



User's Guide

AIX VS COBOL
Compiler/6000



First Edition (March 1990)

This edition of the *User's Guide for IBM AIX VS COBOL Compiler/6000* applies to Version Number 1.1 of the IBM AIX VS COBOL Compiler/6000 Licensed Program and to all subsequent releases of these products until otherwise indicated in new releases or technical newsletters.

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: **INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS MANUAL "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.** Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed program. You can use any functionally equivalent program instead.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 997, 11400 Burnet Road, Austin, Texas 78758-3493. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

IBM is a registered trademark of International Business Machines Corporation.

©Copyright International Business Machines Corporation 1987, 1990. All rights reserved.

©Copyright Micro Focus, Ltd. 1987, 1990. All rights reserved.

Notice to U.S. Government Users - Documentation Related to Restricted Rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

Trademarks and Acknowledgements

The following trademarks apply to this book.

- IBM is a registered trademark of International Business Machines Corporation.
- RT is a registered trademark of International Business Machines Corporation.
- AIX is a trademark of International Business Machines Corporation.
- AIX VS COBOL Compiler/6000 is a trademark of International Business Machines Corporation.
- OS/VS COBOL and VS COBOL II are trademarks of International Business Machines Corporation.
- Micro Focus and VS COBOL Workbench are trademarks of Micro Focus.
- Micro Focus COBOL/2, VS COBOL, FILESHARE, LEVEL II COBOL/ET, Professional COBOL, Professional COBOL/2, and ANIMATOR are trademarks of Micro Focus.
- UNIX is a registered trademark of AT&T.
- RM/COBOL is a trademark of Ryan McFarland Corporation.
- Microsoft is a trademark of Microsoft Corporation.
- Data General is a trademark of Data General Corporation.

About This Book

This book explains how to develop and execute COBOL programs on the IBM AIX Operating System.

Who Should Read This Book

This book is intended for users who have a good understanding of the COBOL programming language. A detailed presentation is provided in the *Language Reference*. Users should also be familiar with the IBM AIX Operating System.

How to Use This Book

The following highlighting and notation conventions are used in this book:

- Commands, keywords, file names, and keys appear in **bold type**. Not all commands are case-sensitive. For example, you can type **b** or **B** for **break**.
- New terms appear in *bold italic* type.
- Examples appear in monospace type.
- Lowercase italics appear when the presence of a variable item is implied, for which you must substitute a particular value.

```
ACCEPT dataname FROM CRT
```

- When items are enclosed in braces { }, you must choose only one of the items.
- When items are enclosed in square brackets [], you can choose one or none of the items.
- An item followed by an ellipsis . . . may be repeated multiple times.

```
CALL procedure USING parameter . . .
```
- In text and in command line formats, the symbol ◀^l is used to represent the **Enter** or **Return** key on your keyboard.

How This Book is Organized

This book contains the following information:

- Chapter 1, “Introduction” provides an overview of using IBM AIX VS COBOL on the IBM AIX Operating System.
- Chapter 2, “Advice on Writing COBOL Programs” provides information about how to write AIX VS COBOL programs as efficiently as possible.
- Chapter 3, “Device- and File-Handling” describes AIX files and devices as they appear to AIX VS COBOL programs.
- Chapter 4, “The COBOL Interface” describes the **cob** command, which provides the user interface to the AIX VS COBOL system.
- Chapter 5, “Compiler Options” describes the system-wide compiler options and their default values.

-
- Chapter 6, “Native Code Generator Options” describes the Native Code Generator options and their default values.
 - Chapter 7, “Running an AIX VS COBOL Program” describes the procedure for running an AIX VS COBOL program.
 - Chapter 8, “File Sharing in the Multi-User Environment” describes how AIX VS COBOL provides independent COBOL programs with the ability to share data files in AIX VS COBOL multiple-user environments.
 - Chapter 9, “Advanced Programming Features” describes three advanced programming features: library subroutines, Run Time Environment (RTE) subprograms for special features, and the file handler.
 - Chapter 10, “Configuring Your AIX VS COBOL System” describes how to alter the default behavior of the AIX VS COBOL ACCEPT and DISPLAY statements.
 - Chapter 11, “Debugging Your Program Using ANIMATOR” describes how to debug your programs interactively using ANIMATOR.
 - Chapter 12, “Designing Display Screens and Programs Using FORMS-2” describes how to use FORMS-2 to create and edit data entry screens and programs in interactive mode.
 - Chapter 13, “Ryan-McFarland COBOL: Conversion Series 3” describes how to use IBM AIX VS COBOL to process Version 2 RM/COBOL source programs.
 - Chapter 14, “Data General COBOL: Conversion Series 5” describes how to migrate from the DGCOBOL environment to an AIX VS COBOL environment.
 - Chapter 15, “Error Messages” lists error messages that you may see in syntax checking, code generation, and system-generated messages.
 - Appendix A, “Environment Variables” describes the environment variables you can set for AIX VS COBOL.
 - Appendix B, “National Language Support” provides information on national language support.
 - Appendix C, “Character Sets and Collating Sequence” contains tables of character sets and the collating sequence for AIX VS COBOL.
 - Appendix D, “Packaging Application Programs” explains how to compile application programs that will be distributed.

Related Publications

The AIX VS COBOL Compiler/6000 documentation is available in hardcopy publications only. Softcopy information to support AIX and other licensed programs is provided with the product. The entire AIX library is available as softcopy on a CD-ROM. Refer to the operating system documentation for more detailed information on the various features of AIX. The following hardcopy documentation is also available.

Language Reference describes how to compile and execute AIX VS COBOL programs.

Ordering Additional Copies of This Book

To order additional copies of this book (without the program media), use form number SC23-2178-00.

Contents

Chapter 1. Introduction	1-1
Contents	1-2
About This Chapter	1-3
Introduction	1-4
AIX VS COBOL System Components	1-4
Compiler	1-5
Native Code Generator	1-5
Run Time Environment	1-5
Linker	1-5
ANIMATOR	1-5
FORMS-2	1-5
Static Linking and Dynamic Loading	1-6
Statically Linked Code	1-6
Dynamically Loaded Code	1-6
Program Development Cycle	1-6
Program Source Conventions	1-7
Checking Out the AIX VS COBOL System with Demonstration Programs	1-8
pi.cbl	1-9
stock1.cbl	1-10
stock2.cbl	1-11
Chapter 2. Advice on Writing COBOL Programs	2-1
Contents	2-2
About This Chapter	2-3
Optimizing COBOL Programs	2-4
Features to be Used with Care	2-4
Programming Restrictions	2-5
Current Restrictions on the Use of Some SAA Functionality	2-6
Using Intermediate or Native Code	2-7
Optimizing Native Code	2-8
Handling Large Programs	2-8
Segmentation (Overlaying)	2-9
Interprogram Communication (Call and Cancel)	2-9
Calling AIX VS COBOL Subprograms	2-11
Search Sequence for Locating File Name	2-11
Multiple Entry-Points	2-13
Calling Non-COBOL Subprograms	2-15
Cancelling Non-COBOL Subprograms	2-18
Mixing C and COBOL Programs	2-18
Passing the Command Line	2-21
Calling Operating System Functions	2-23
AIX VS COBOL Dialect Flagging and Error Reporting	2-24
Chapter 3. Device- and File-Handling	3-1
Contents	3-2
About This Chapter	3-3
Devices	3-4
File Assignment	3-4
Special Characters in Environment Variables	3-9
AIX VS COBOL Disk File Structure under AIX	3-12
Record-Sequential Files	3-12

Line-Sequential Files	3-12
Relative Files	3-12
Indexed Sequential Files	3-13
Library Files	3-16
File Restrictions	3-17
Input-Output Error-Handling (File Status)	3-17
Alternate File Status Table	3-18
Writing Output Directly to a Printer	3-20
Chapter 4. The COBOL Interface	4-1
Contents	4-2
About This Chapter	4-3
COBOL Interface Command	4-4
The Development Cycle	4-5
Option Specification	4-7
System-Wide Default Options	4-7
Optional User Default Options	4-7
Command Line Options	4-8
Embedded Source File Options	4-15
Command Line Conventions	4-16
Command Line Examples	4-17
Chapter 5. Compiler Options	5-1
Contents	5-2
About This Chapter	5-3
Format of Compiler Options	5-4
Permitted Options	5-5
Excluded Combinations	5-23
ANS85 Options	5-24
Default Options	5-24
Mainframe Options	5-27
SAA Options	5-27
Options Permitted in \$SET Statements	5-28
Compiler Messages	5-29
Listing Format	5-30
Chapter 6. Native Code Generator Options	6-1
Contents	6-2
About This Chapter	6-3
Permitted Options	6-4
Default Options	6-5
Native Code Generator Messages	6-6
Chapter 7. Running an AIX VS COBOL Program	7-1
Contents	7-2
About This Chapter	7-3
Command Line Syntax	7-4
Command Line Examples	7-5
Examples	7-5
Switch Parameters	7-6
Run-Time Switches	7-7
ANIMATOR Switch (A)	7-7
Skip Locked Record Switch (B)	7-7
ANSI COBOL Debug Switch (D)	7-8
COBOL Symbol Switch (e)	7-8

Error Switch (E)	7-8
Compatibility Check Switch (F)	7-8
Keyboard Interrupt Switch (i)	7-9
ISAM Files Sequence Check Switch (K)	7-10
Memory Switch (l)	7-10
Null Switch (N)	7-10
Dynamic Linkage Setup Switch (p)	7-10
File Status Error Switch (Q)	7-11
Reread Locked Record Switch (R)	7-11
Sort Memory Switch (s)	7-12
Sort Switch (S)	7-12
Tab Switch (T)	7-12
Examples	7-13
Run Time Environment Error Messages	7-13
COBOL Profiler	7-14
Profiler Directives	7-14
Profiler Output	7-15
Chapter 8. File Sharing in the Multi-User Environment	8-1
Contents	8-2
About This Chapter	8-3
A Typical Multi-User Environment	8-4
Including Multi-User Syntax in Your Program	8-4
Facilities for Multi-User AIX VS COBOL	8-4
Data Locking	8-5
Organization of Shared Files	8-6
The Procedure Division	8-11
File Status	8-11
Demonstration Programs	8-13
Running the Demonstration Programs	8-13
Chapter 9. Advanced Programming Features	9-1
Contents	9-2
About This Chapter	9-3
Library Subroutines	9-4
cobsetjmp and coblongjmp	9-4
cobtidy	9-5
RTE Subprograms	9-5
Put a Character to the Screen	9-6
Read a Character from the Keyboard	9-7
Split/Join a File Name	9-7
File-Related Operations	9-8
Modifying the Behavior of User Attributes	9-9
Modifying the Behavior of ACCEPT/DISPLAY	9-9
Display Screen Input and Output	9-11
Test Keyboard Status	9-13
Sound the Audible Alarm	9-13
Move the Cursor to a Defined Position	9-13
Pack Byte	9-14
Unpack Byte	9-14
CRT Screen Handling	9-14
The ACCEPT and DISPLAY Statements	9-14
Display Attributes	9-15
Screen Handling From C	9-16
Using Escape Sequences to Send Attribute Information to the Screen	9-19

File Handler	9-20
Interface to the COBOL File Handler	9-21
Operation Codes Passed in the Second Byte of the First Parameter	9-21
Information Passed in the FCD at Open Time	9-22
Information Passed for Other Operations	9-22
FCD Information Format	9-23
Key Definitions for Indexed Files	9-25
Global Information	9-25
Key Definitions	9-25
Component Definitions	9-26
CISAM Features	9-26
Chapter 10. Configuring Your AIX VS COBOL System	10-1
Contents	10-2
About This Chapter	10-3
Introduction	10-4
terminfo	10-5
cobkeymp	10-5
ADISCTRL	10-5
Keyboard Conversion Process	10-5
keybcf Utility	10-6
Specifying and Accessing Multiple or Alternate cobkeymp Files	10-7
Invoking the keybcf Utility	10-8
Using the keybcf Utility	10-9
Maximum Size of keybcf Buffers	10-14
adiscf Utility	10-14
Invoking the adiscf Utility	10-14
Using the adiscf Utility	10-14
Chapter 11. Debugging Your Program Using ANIMATOR	11-1
Contents	11-2
About This Chapter	11-3
Introduction	11-4
Facilities Not Supported by ANIMATOR	11-5
Getting Started	11-5
Running ANIMATOR	11-6
Specifying Directives	11-6
ANIMATOR Directives	11-6
ANIMATOR Display Screen	11-7
Using ANIMATOR Commands	11-8
Help Display Screens	11-9
Animating STOCK1	11-9
Using Break Points	11-11
Examining the Contents of Data Items	11-13
Ending Animation	11-14
Animating Your Own Programs	11-15
Using the ANIMATOR Switch	11-15
Command Line Switches	11-16
File Searches	11-16
Animating CALLED Programs	11-17
OS/VS COBOL-Style PERFORMS	11-17
Other Remarks about Animation	11-18
Cursor Control Keys	11-18
ANIMATOR Commands	11-19
Help	11-19

View	11-20
Align	11-20
eXchange	11-20
Where	11-20
looKup	11-21
word-left (<) and word-right (>)	11-21
Escape Key	11-21
Letter Commands	11-21
Step	11-21
Go	11-22
next-If	11-22
Perform	11-23
Reset	11-23
Break	11-24
Env	11-27
Query	11-32
Find	11-37
Locate	11-38
Text	11-39
Do	11-40
ANIMATOR Command Summary	11-41
Chapter 12. Designing Display Screens and Programs Using FORMS-2	12-1
Contents	12-2
About This Chapter	12-3
Introduction	12-4
Outputs	12-5
Phases	12-5
Operator Interface	12-6
FORMS-2 Validation	12-7
Initialization Phase	12-16
Initialization Display Screen I01	12-16
Initialization Display Screen I02	12-17
Work Phase	12-18
Display Screen W01	12-18
Work Display Screen	12-19
Work Phase Completion	12-28
Data Descriptions	12-29
Record Name and Data-Name Generation	12-29
Picture Generation	12-30
Editing the DDS File	12-30
Incorporation of DDS File Contents	12-30
Checkout Program	12-31
Checkout Program Generation	12-31
Checkout Program Compilation	12-31
Checkout Program Running	12-31
Checkout Processing	12-32
Checkout Completion	12-32
Display Screen Image File	12-33
Display Screen Image File Generation	12-33
FORMS-2 Maintenance	12-33
Printed Forms	12-34
Form Images in the Design Process	12-34
FORMS-2 User Display Screen Generation Example	12-35
Index Program	12-39

Index Program Generation	12-40
Index File Generation	12-41
Index Program Compilation	12-41
Index Program Running	12-41
User Index Program Example	12-43
Chapter 13. Ryan-McFarland COBOL: Conversion Series 3	13-1
Contents	13-2
About This Chapter	13-5
Converting RM/COBOL Applications to AIX VS COBOL	13-6
Submitting RM/COBOL Source Programs to AIX VS COBOL	13-6
Converting Data Files	13-6
Enhancing Your Converted Application	13-6
Other Considerations for Conversion	13-7
Submitting an RM/COBOL Application to the AIX VS COBOL System	13-7
Migrating from the RM/COBOL Environment	13-7
tabx Program	13-8
Source Compatibility	13-9
RM Directive	13-9
SPZERO Option	13-9
Perform Statements	13-9
Types of Data	13-10
COMPUTATIONAL-1 (COMP-1) Data Types	13-10
COMPUTATIONAL-6 (COMP-6) Data Types	13-10
COMPUTATIONAL (COMP) Data Types	13-10
Conversion Problem Solving	13-11
Length of Nonnumeric Literals	13-11
Source Code in Columns 73 to 80	13-12
Reserved Words	13-12
Numbering Segments	13-12
Program Identification and Data-Names	13-13
Column Number Specification	13-13
End-of-File Notification	13-13
HIGH-VALUES	13-13
Duplicate Paragraph Names	13-14
Display of Input Data in Concealed ACCEPT Fields	13-14
Executable Code Problems	13-14
Trailing Blanks in Line-Sequential Files	13-15
Undefined Results of MOVE and Arithmetic Operations	13-15
Embedded Control Sequences in DISPLAY Statements	13-15
Redefinition of COMPUTATIONAL or COMPUTATIONAL-6 Data	
Items	13-16
ON SIZE ERROR Clause	13-17
Field Wrap-Around	13-17
COMPUTATIONAL-1 Data Items with a Picture Other Than S9(4)	13-18
File and Record Locking	13-19
Initialization of the WORKING-STORAGE	13-19
Converting Data Files for Use with Converted Programs	13-20
Supported Data File Types	13-20
COMP/COMPUTATIONAL Data	13-20
COMP-3/COMPUTATIONAL-3 Data	13-21
COMP-6/COMPUTATIONAL-6 Data	13-22
DISPLAY Data	13-22
Program Modifications Required by convert3	13-23
Running convert3	13-24

Running convert3 in Interactive Mode	13-24
File Details	13-25
Print File Name	13-25
Record Type Specification	13-26
Binary Sequential Files	13-28
Generate Program	13-28
Escape	13-29
Running convert3 in Batch Mode	13-29
Running convert3 with a Parameter File	13-33
Using the File Conversion Program	13-33
Creating an Executable File Conversion Program	13-34
Running the File Conversion Program	13-34
Indexed Sequential Files with Duplicate Alternate Keys	13-34
convert3 and File Conversion Program Error Messages	13-35
convert3 Error Messages	13-35
File Conversion Program Error Messages	13-36
Chapter 14. Data General COBOL: Conversion Series 5	14-1
Contents	14-2
About This Chapter	14-3
Converting DG Interactive COBOL Applications to AIX VS COBOL	14-4
Submitting Source Programs	14-4
Enhancing Converted Applications	14-4
Source Compatibility	14-5
The DG Directive	14-5
Reserved Words	14-5
DG International Character Set	14-5
DG File Status and Other Exception Values	14-6
Calls	14-6
LINKAGE SECTION Access	14-6
Arithmetic of Group Level Items	14-6
Run-Time Switches	14-6
Program Identification and Data-Names	14-6
Reformatting a DG Source File	14-6
Using reform5	14-7
Reformatting Rules	14-7
Converting Data Files for Use with Converted Programs	14-7
Supported Data File Types	14-8
DG Data Types	14-10
Source File Restrictions	14-11
The File Conversion Process	14-12
Running convert5	14-12
Running convert5 in Interactive Mode	14-13
Running convert5 in Batch Mode	14-17
Example Parameter List	14-19
Running convert5 with a Parameter File	14-20
Using the File Conversion Program	14-20
Creating an Executable File Conversion Program	14-20
Running the File Conversion Program	14-20
Error Messages	14-22
Errors Reported by convert5	14-22
Errors Reported by the Conversion Program	14-23
Chapter 15. Error Messages	15-1
Contents	15-2

About This Chapter	15-3
Introduction	15-4
Compiler Messages	15-4
Severe Compiler Messages	15-7
Compiler Error Messages	15-40
Compiler Warning Messages	15-43
Compiler Information Messages	15-46
Compiler Flags	15-47
Errors Encountered During Code Generation	15-54
Native Code Generator Messages	15-54
Run Time Environment Errors	15-60
Types of Errors	15-60
Run Time Environment Error Messages	15-62
cob Command Errors	15-85
Appendix A. Environment Variables	A-1
Introduction	A-3
COBATTR	A-3
COBCPY	A-4
COBCTRLCHAR	A-4
COBDIR	A-4
COBHELP	A-5
COBIDY	A-5
COBLPFORM	A-5
COBOPT	A-6
COBPATH	A-6
COBPRINTER	A-7
COBSW	A-7
TMPDIR	A-8
Appendix B. National Language Support	B-1
Introduction	B-3
Features Provided by National Language Support	B-3
Compiling Programs with National Language Support	B-4
Running Programs with National Language Support	B-5
Running Your Program	B-5
RTE NLS Initialization	B-6
String Comparisons	B-6
Class Condition Tests	B-6
Indexed Sequential File Operations	B-7
Comparisons Performed as Part of SORT or MERGE Statements	B-7
The NLS Support Routines	B-7
Mixing Programs with and without National Language Support	B-8
Appendix C. Character Sets and Collating Sequence	C-1
Appendix D. Packaging Application Programs	D-1
Introduction	D-3
The Run Time Package	D-3
Preparing Application Packages	D-3
Statically Linkable Native Code (.o)	D-3
Dynamically Loadable Native Code (.gnt)	D-4
Intermediate Code (.int)	D-4
Glossary	G-1

Index	X-1
--------------------	------------

Figures

1-1.	Program Development Cycle	1-7
2-1.	Sample CALL Tree Structure	2-10
8-1.	A Hypothetical Multi-User Environment	8-6
8-2.	FILE-CONTROL Paragraph Syntax for Record and Line-Sequential Files	8-7
8-3.	FILE-CONTROL Paragraph Syntax for Relative Files	8-8
8-4.	FILE-CONTROL Paragraph Syntax for Indexed Sequential Files	8-10
8-5.	Initial Display Screen of the Demonstration Program	8-14
10-1.	Character Conversion Process	10-6
10-2.	Main keybcf Display Screen	10-9
10-3.	Alter Function Key Options	10-11
10-4.	Main adiscf Command Menu	10-15
10-5.	Load Option	10-27
11-1.	ANIMATOR Display Screen	11-7
11-2.	Example of CALL Statement/PERFORM Level Relationship	11-29
14-1.	Example Program to Reformat DG Interactive COBOL Relative Data File	14-9
14-2.	An Example Parameter File	14-19

Tables

3-1.	File Name Mapping	3-8
4-1.	Development Cycle of Input File to cob Command	4-6
5-1.	Excluded Combinations of Options	5-24
9-1.	Default File Handlers	9-20
9-2.	Operation Codes Passed in the Second Byte of the First Parameter	9-21
10-1.	Default ADIS Control Keys	10-7
10-2.	Hexadecimal Sequences for Key Functions Not on Your Keyboard	10-12
10-3.	Default Mappings of ADIS Function Keys	10-26
11-1.	ANIMATOR Command Summary	11-41
12-1.	Cursor Control Keys	12-6
13-1.	Error Message Identification	13-35
14-1.	Error Message Identification	14-22
C-1.	Character Set and Collating Sequence	C-1

Chapter 1. Introduction

Contents

About This Chapter	1-3
Introduction	1-4
AIX VS COBOL System Components	1-4
Compiler	1-5
Native Code Generator	1-5
Run Time Environment	1-5
Linker	1-5
ANIMATOR	1-5
FORMS-2	1-5
Static Linking and Dynamic Loading	1-6
Statically Linked Code	1-6
Dynamically Loaded Code	1-6
Program Development Cycle	1-6
Program Source Conventions	1-7
Checking Out the AIX VS COBOL System with Demonstration Programs	1-8
pi.cbl	1-9
stock1.cbl	1-10
stock2.cbl	1-11

About This Chapter

This chapter discusses the components of the IBM AIX VS COBOL Compiler/6000 Version 1.1, and provides a program development cycle overview, installation instructions, and demonstration programs that illustrate the use of the AIX VS COBOL compiler.

Introduction

IBM AIX VS COBOL Compiler/6000 Version 1.1 provides a high performance optimizing compiler that produces object code for execution on the IBM RISC System/6000 under the AIX Operating System. AIX VS COBOL accepts COBOL source code as defined by the following standards:

- ANSI COBOL X3.23 1985 High
- ANSI COBOL X3.23 1974 High
- IBM SAA Level 1 COBOL
- FIPS PUB 21-2.

Conversion utilities are provided to migrate Data General COBOL and Ryan-McFarland COBOL source to AIX VS COBOL source.

In addition, AIX VS COBOL offers these enhanced functions:

- Source compatibility with the following:
 - IBM AIX PS/2 VS COBOL
 - IBM AIX/RT VS COBOL
 - IBM OS/VS COBOL (Release 2.4 and earlier)
 - IBM VS COBOL II (Release 1, December 1984)
 - IBM COBOL/2 (Release 1)
 - Micro Focus Extensions (Level 4 and earlier)
 - Data General Interactive COBOL Revision 1.30
 - Ryan-McFarland COBOL 2.0
 - Microsoft COBOL 1.0 and 2.2
- Automated installation
- Optimized code
- Operating system interface library
- Interlanguage linkages with C
- Detailed on-screen messages
- Development and debugging environment
- Interactive design of application screen layouts
- National Language Support.

AIX VS COBOL System Components

The AIX VS COBOL system is a compact, interactive system. The major components of the AIX VS COBOL system are as follows:

- Compiler
- Native Code Generator
- Run Time Environment (RTE)
- ANIMATOR debugging tool
- FORMS-2 screen/program facility.

The **cob** command provides access to the compiler, the Native Code Generator, the **cc** command and the AIX system linker. Files specified to this command can be any mixture of COBOL source, intermediate code, native code, linkable object code, assembler source files or C source files. By default, the **cob** command converts the specified files into intermediate code files that are suitable for animation. The **cob** command can output native code, intermediate code, or statically linked executable modules, depending on the options you specify. The AIX system linker is used to link the object files to the RTE which creates one executable file.

Compiler

The AIX VS COBOL compiler translates COBOL source code into an intermediate code. This intermediate code is a sequence of instructions to the machine.

Native Code Generator

The Native Code Generator translates the intermediate code produced by the compiler into the native code of the IBM RISC System/6000.

Run Time Environment

The Run Time Environment (RTE) loads files of intermediate or native code. By default, the RTE passes native code to the IBM RISC System/6000 processor for direct execution, although it can execute code interactively. The RTE also acts as the interface between your COBOL program and such operating system functions as file and device-handling and memory management.

Note: You may see the phrase Run Time Environment referred to as Run Time System in various places in the AIX VS COBOL publications. The phrases are synonymous and are interchangeable. The same holds true for the acronyms RTE and RTS.

Linker

The linker links the object files to the RTE and creates one executable file.

ANIMATOR

ANIMATOR is an interactive tool that allows for quick and easy program debugging. ANIMATOR allows you to do the following:

- See in what sequence the statements of your program are executed.
- Halt the program at any time.
- Display the contents of data items and change them.
- Alter the sequence in which statements are executed.
- Cause statements to be skipped.

FORMS-2

The FORMS-2 package is an extension to the software development system that enables you to create and edit data entry screens for application programs at a console.

Static Linking and Dynamic Loading

The AIX VS COBOL system allows you to execute statically linked or dynamically loaded code, both of which can be output by the **cob** command, depending on the options specified.

To design COBOL application programs that make the most efficient use of available memory, use a mixture of static and dynamic modules in your programs.

Statically Linked Code

Statically linked code is in the form of a standard AIX **a.out** executable object module that has all of its overlays or other procedures linked into memory. To execute such modules, specify the name of the executable object module on the AIX command line. The name of the module that you would specify is dependent upon how you instructed the **cob** command to create the module. See Chapter 4, "The COBOL Interface" for more information.

The advantage of statically linking programs in a multi-user environment is that it allows users to share programs. It also allows programs to call other programs within the same suite with the maximum possible speed. Static linking can result in large executable files; however, the size of available memory on the AIX system should not be a constraint.

If you want your program to **CALL** a program that is written in a different language, the called program must be statically linked to the AIX VS COBOL RTE.

Dynamically Loaded Code

Dynamically loaded code can be either COBOL intermediate files (**.int**) or COBOL native files (**.gnt**). This code loads modules as needed. Use the **cobrun** command to execute dynamically loadable code. See Chapter 7, "Running an AIX VS COBOL Program" for more information.

Dynamically loaded programs are especially suitable for development environments. It allows you to debug parts of your program while the rest of it is running, and to reload the amended part of your code without needing to reload the entire application. An advantage of dynamically loaded programs is that they require less available memory.

Program Development Cycle

Figure 1-1 on page 1-7 illustrates the typical cycle of creating a COBOL source program and submitting it to the **cob** command.

The type of program output by the **cob** command depends upon command line options specified when you submit your input files. See Chapter 4, "The COBOL Interface" for full details about available options.

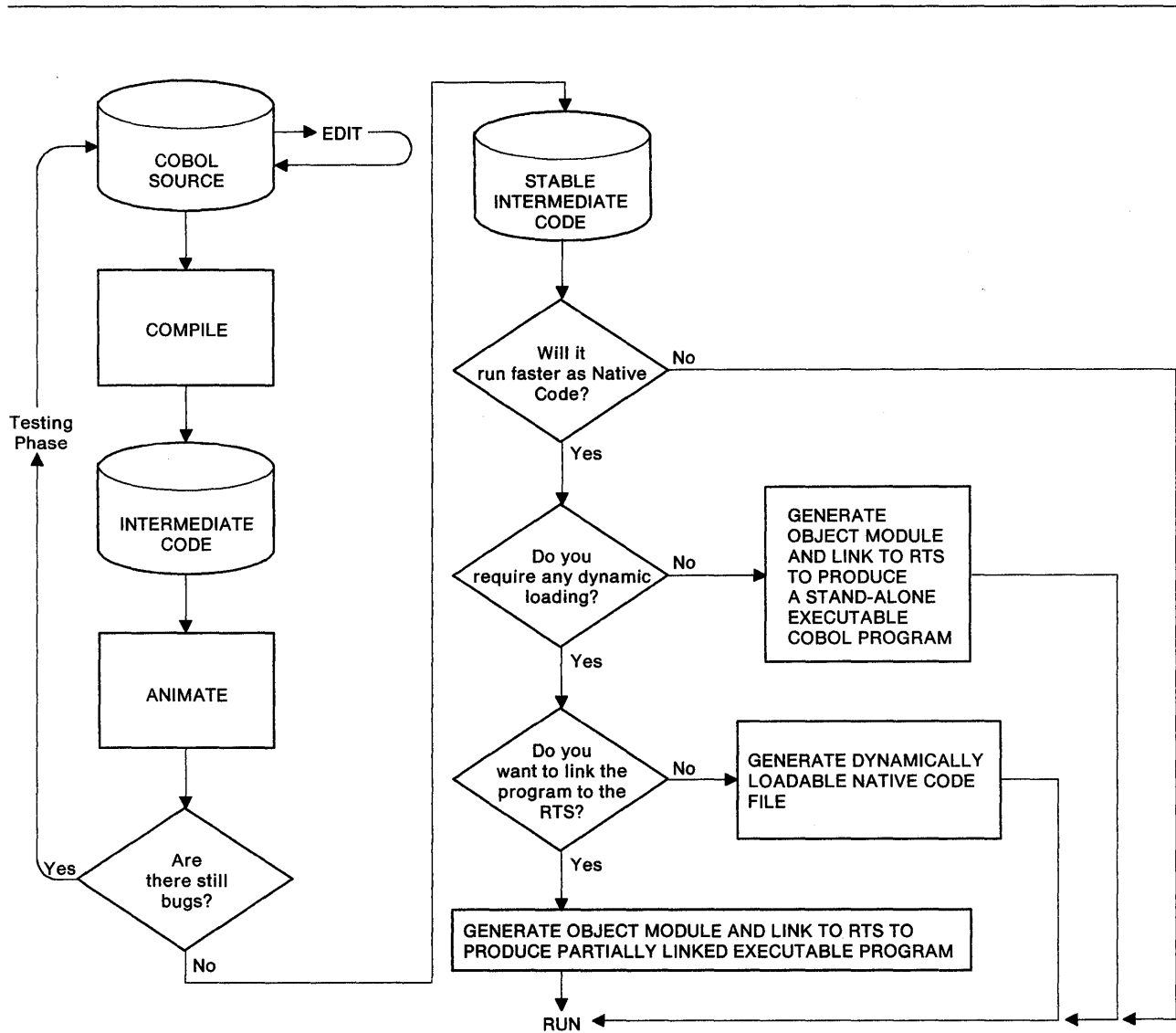


Figure 1-1. Program Development Cycle

Program Source Conventions

The AIX VS COBOL compiler accepts source code from a standard AIX text file (as created by an AIX editor such as vi). The format of the AIX text file is the same as for standard COBOL and is described in the *Language Reference*.

Each line of your COBOL source programs, including the last line, must be terminated by a new line character.

Your COBOL source programs must not contain any control characters (characters with hexadecimal values 00 to 1F inclusive, or 7F) except the tab character, unless they are embedded in literal strings. The tab is expanded with spaces to the next character position that is a multiple of 8.

Checking Out the AIX VS COBOL System with Demonstration Programs

A number of demonstration programs have been provided. The source code for these programs is in the following files of the `/usr/lpp/COBOL/lib/demo` directory:

- `pi.cbl`
- `stock1.cbl`
- `stock2.cbl`
- `mudemo.cbl`
- `stockin.cbl`
- `stockioa.cbl`
- `stockiom.cbl`
- `stockout.cbl`

Each file has a specific function to help you verify that your AIX VS COBOL system is accurately configured to your console, as follows:

File	Description
<code>pi.cbl</code>	This program displays on the screen the mathematical constant pi to 12 decimal places and is the basic screen test for AIX VS COBOL DISPLAY.
<code>stock1.cbl</code>	Do not run this program until you are confident that <code>pi.cbl</code> is working correctly. It is the test for AIX VS COBOL ACCEPT, which provides the basic interactive functions, and indexed-sequential file input-output.
<code>stock2.cbl</code>	This program uses a data file created by running <code>stock1.cbl</code> and is dependent on having run that program successfully. The source code contains a deliberate error that does not affect the program's execution, but provides an example of an AIX VS COBOL error message.
<code>mudemo.cbl</code> <code>stockin.cbl</code> <code>stockioa.cbl</code> <code>stockiom.cbl</code> <code>stockout.cbl</code>	These programs show how the file and record locking syntax of AIX VS COBOL allows a number of programs to have simultaneous access to the same set of indexed-sequential files without interfering with one another. These programs are explained in Chapter 8, "File Sharing in the Multi-User Environment."

The `pi.cbl`, `stock1.cbl`, and `stock2.cbl` programs introduce you to the program development cycle. They also indicate how simple COBOL programs can have sophisticated screen and file-handling features.

Copy these programs into one of your work directories before walking through the examples given in the rest of this section.

pi.cbl

Type the following command line:

```
cob -vxP pi.cbl ◀
```

The **cob** command recognizes the **.cbl** file extension and invokes the compiler. Specifying the **-v** option causes any messages output by **cob** to be displayed on the screen. The **-x** option causes the input file to be processed to a statically linked executable module. By default, the name of this module is the base name of the first file input to the **cob** command. In this case, as there is only one input file, the statically linked module takes its base name, **pi**. The **-P** option forces **cob** to create a listing file, **pi.lst**. Full details on the use of the **cob** command and its options can be found in Chapter 4, "The COBOL Interface."

The first lines displayed immediately tell you that the compiler has been loaded and is executing.

```
* IBM AIX VS COBOL Compiler/6000 LP
* 5601-258 (C) Copyright IBM Corp. 1987, 1990
* Copyright (C) 1984, 1987 Micro Focus, Ltd.
* All Rights Reserved
* Licensed Materials - Property of IBM
* Accepted-verbose
* Accepted-list (pi.lst)
* Compiling pi.cbl
```

When the compilation is finished, the compiler reports the results as follows:

```
* Total messages: 1
* Unrecoverable: 0      Severe: 0
* Errors: 0          Warnings: 0
* Information: 1      Flags: 0
```

It also outputs a message giving the sizes of the code, the data areas, and the compiler dictionary.

Next, the Native Code Generator is invoked. Messages are displayed to tell you that the Native Code Generator is loaded and is executing.

When code generation is completed, the Native Code Generator outputs a message giving the sizes of the data and code areas, the literals, and the Native Code Generator dictionary.

Then, a single executable file (**pi**) is created, which contains the RTE support libraries required by the program **pi**, with **pi** linked to them.

In addition to the executable file, **pi**, the compiler generates the following two files:

- **pi.lst**, which contains the list file
- **pi.int**, which contains the intermediate code.

Running the Linked RTE

To run **pi**, enter the name of the file containing the linked RTE, as follows:

```
pi ↵
```

The display screen clears, the cursor appears at the top left, and the **pi** screen is displayed as illustrated below for the final term:

```
CALCULATION OF PI  
  
NEXT TERM IS 0.000000000000  
  
PI IS 3.141592653589
```

Problem Diagnosis

If the screen is not displayed or is displayed incorrectly, the terminfo entry for your console type may be incorrect. See Chapter 10, “Configuring Your AIX VS COBOL System” for further information.

stock1.cbl

To submit **stock1.cbl** to the **cob** command and output a single statically linked executable module named **stock1**, enter the following:

```
cob -vxP stock1.cbl ↵
```

This command outputs a listing in the file **stock1.lst**, and causes the **cob** command to display any messages it outputs on the display screen.

Running stock1

After the **cob** process has finished successfully, you can run the single executable file **stock1** it produced by entering:

```
stock1 ↵
```

The display screen clears and the following screen is displayed:

```
STOCK CODE < >  
DESCRIPTION < >  
UNIT SIZE < >
```

The program waits for you to enter data through the keyboard using the **Tab** key, the **Backspace** key, and the **Return** (↵) key.

Before entering any data, try moving the cursor from data item to data item using the cursor control keys. Note that AIX VS COBOL does not allow you to position the cursor outside the bounds of the data to be entered. While entering data, check the following two functions:

- Left zero-fill, which can be tested using the ‘.’ on data entered into UNIT SIZE. Keying **1.** should result in **0001**.
- **Return** (↵), to enter your first display screen full of data.

In the following two cases, pressing ↵ will not result in data being written to the file.

1. When UNIT SIZE is not numeric
2. When a record with this STOCK CODE number already exists on the file.

Reference to the listing of the source program shows why. The relevant statements are as follows:

```
IF CRT-UNIT-SIZE NOT NUMERIC GO TO CORRECT-ERROR.
```

and

```
WRITE STOCK-ITEM; INVALID GO TO CORRECT-ERROR.
```

Case 1 is the result of an explicit test by the programmer for valid data input. Case 2 arises from the fact that the `STOCK CODE` is being used as the Record key, and duplicate keys are not permitted in the indexed sequential file to which these records are being written.

To terminate the run cleanly, you must key spaces into the `STOCK CODE` field and press `↵`. The program tests for this end-of-run signal in the line:

```
IF CRT-STOCK-CODE = SPACE GO TO END-IT.
```

Problem Diagnosis

Typical problems that may be experienced are as follows:

- Cursor fails to move or moves incorrectly, with one or more of the cursor movement keys. If you are using the keys correctly, the terminfo entry for your console type may be incorrect.
- A run-time error may occur if the files `STOCK.IT` and `STOCK.IT.idx`, generated and referenced by `stock1`, have been damaged; for example, by a previous run of `stock1` that was incorrectly terminated. To recover, delete the two files `STOCK.IT` and `STOCK.IT.idx`, and start again with the `stock1` program.

`stock2.cbl`

Compile `stock2.cbl` using the same command line options used to compile `stock1.cbl`.

Run the program, and retrieve the records you entered to the file using `stock1` by entering into the `STOCK CODE` field the values you previously used. Again, spaces in the `STOCK CODE` field must be used to terminate the run.

Chapter 2. Advice on Writing COBOL Programs

Contents

About This Chapter	2-3
Optimizing COBOL Programs	2-4
Features to be Used with Care	2-4
Programming Restrictions	2-5
Current Restrictions on the Use of Some SAA Functionality	2-6
Using Intermediate or Native Code	2-7
Optimizing Native Code	2-8
Handling Large Programs	2-8
Segmentation (Overlaying)	2-9
Interprogram Communication (Call and Cancel)	2-9
Calling AIX VS COBOL Subprograms	2-11
Search Sequence for Locating File Name	2-11
Multiple Entry-Points	2-13
Calling Non-COBOL Subprograms	2-15
Cancelling Non-COBOL Subprograms	2-18
Mixing C and COBOL Programs	2-18
Passing the Command Line	2-21
Calling Operating System Functions	2-23
AIX VS COBOL Dialect Flagging and Error Reporting	2-24

About This Chapter

This chapter highlights a number of IBM AIX VS COBOL procedures for which the AIX VS COBOL system produces particularly efficient code. It also describes a number of implementation restrictions that you must be aware of when writing AIX VS COBOL programs. This chapter describes considerations for handling large programs and explains how to call subprograms written in COBOL and other languages.

Optimizing COBOL Programs

The following information will increase the performance of your COBOL programs:

- The CALL statement executes faster when parameters appear in the same order in the CALL statement as they do in the Procedure Division header and in the LINKAGE SECTION of the called program.
- Place any LINKAGE SECTION items that are not referenced in the Procedure Division header after those that are.
- All parameters are 01 or 77 level items.
- Parameter passing is faster when an out-of-line PERFORM statement is a single section, and if the system is sure that the section is entered and left only under the control of a PERFORM statement. Try to ensure that any section that is to be PERFORMed has no jumps in or out of it, or any alterable GO TO statements. You must also ensure that every preceding section is either entered or left under the control of a PERFORM statement, or finishes with a STOP RUN. An EXIT PROGRAM statement is not sufficient.
- Access to a table is fastest if every occurrence of a given item is aligned in the same way, and if the length of each entry is a power of 2 or 4. You can achieve this by placing FILLERs in the table.
- Items in the WORKING-STORAGE SECTION are accessed more quickly than those in the LINKAGE SECTION.
- The fastest type of comparison is a comparison for equality with binary zero.
- In complex conditions, you should place first those tests that will give the quickest decisions or are most likely to be true.

Features to be Used with Care

The following information clarifies a number of features that can be misunderstood and cause programming errors:

- A MOVE operation caused by an INTO phrase in either a READ or RETURN statement is executed even if the READ or RETURN operation is unsuccessful.
- Bound checking on a variable-length table operates if the subscript or index points outside the maximum length of the table. It does not take into account the current length of the table (that is, the value of the item specified in the DEPENDING phrase).
- If you attempt a MOVE between two numeric-edited items the result will be undefined, although no error status is returned.
- If you use the DECIMAL POINT IS COMMA clause, you must ensure that any commas separating two numeric literals are followed by a space. Any commas which are not followed by a space are treated as decimal points.
- The **comp** option can alter the results of certain arithmetic statements. In particular, the **comp** option can alter the result of a SUBTRACT statement. This is because the **comp** option allows true unsigned overflow. For example, if you subtract 1 from 0, where both digits are unsigned **comp** values, a large positive number results when the **comp** option is set on. With the default **nocomp**, the result is 1.

-
- When the OSVS option is set, all statements are treated as a comment until a DIVISION heading is read. This means copy statements are not recognized before a DIVISION heading is encountered.
 - When porting your program, be sure numeric parameters stored in binary format passed to another language have the correct byte order. AIX VS COBOL Compiler/6000 byte order is HIGH/LOW.

Programming Restrictions

You should be aware of the following limits while using the AIX VS COBOL system:

- If you are calling COBOL code, you can have a maximum of 59 parameters. If you are calling C code, you can have a maximum of 254 parameters.
- According to the ANSI standard, numbers are limited to 18 significant decimal digits, and all significant digits are within 18 digits of the decimal point.
- In AIX VS COBOL the result of a multiplication or division that is greater than 36 digits gives a SIZE ERROR, as will the result of an addition or subtraction that is greater than 37 digits.
- The maximum number of indexed files that can be open simultaneously is 64.
- The maximum indexed sequential record size is 8 Kbytes.
- The maximum number of keys in an indexed sequential file is 64 (63 alternate).
- The maximum length of an indexed sequential record key is 120 bytes. If this limit is exceeded, a run time error message is issued and the program aborts. The length of all keys cannot exceed 7680 bytes.
- The maximum number of parts for a split key is 8.
- Maximum nesting of PERFORMS in .int programs is 100. (This does not apply to .gnt programs.)
- The maximum number of file or record locks that may be held depends upon the system parameter **nflocks** in the **/etc/master** file. See the AIX operating system documentation for details.
- If your program contains a CHAIN statement that includes a subscripted item greater than 8 Kbytes as a USING field, unpredictable results will occur at run time. The following Native Code Generator error message will be output:
 ILLEGAL INTERMEDIATE CODE
- The maximum number of USING parameters per entry point is 62.
- The maximum number of nested IF statements in a source program is 64.
- The maximum record length is 65 535 bytes.
- The ISAM block size is 1 Kbyte.

- Where a CHAIN statement or a CALL PROGRAM statement includes USING parameters which are defined in the Linkage Section or File Section of the chaining program, results at run time can be unpredictable. Data should be moved from these sections into Working-Storage items for use as a USING parameter. The CALL PROGRAM statement refers to Data General Interactive COBOL syntax. The AIX VS COBOL CALL statement is not included in this restriction.
- In order to use reference modification within an IF statement or an EVALUATE statement, the compiler directive `osvs` must not be set when you compile your source code under AIX VS COBOL.
- AIX VS COBOL does not support tape. Syntax related to tape manipulation is supported for compatibility. Such operations will behave as described for non-reel media.
- In order to be able to WRITE output directly to a printer, you must be a member of the "system" group. This is due to permissions for the files `/dev/lp0` and `/dev/lp1` on the AIX file system.
- If you call the AIX system routine `load` from a module that is statically bound into the RTE, you will not be able to dynamically load `.int` or `.gnt` COBOL routines.
- When running programs compiled with the `SIGN=EBCDIC` compiler option, comparisons with numeric literals do not function correctly in `.gnt` and executable code.

Current Restrictions on the Use of Some SAA Functionality

The following restrictions apply to the use of some of the functionality found in the COBOL SAA Reference.

- OPEN WITH NO REWIND should return a file status of 07 when the physical device is not a tape. This operation currently returns a file status of 00.
- If a record key number is too large to be contained in the variable declared as the relative key, then the WRITE for that record should fail. Currently, that WRITE will create a record on the file.
- A READ statement with an improper INTO phrase does not currently produce an error. Given:

```

FILE SECTION.
FD A.
01 a-1 pic aa.
01 a-2.
   02 a-3 pic aa.
WORKING-STORAGE SECTION.
77 A-4 PIC AA.
PROCEDURE DIVISION.
...
READ A RECORD INTO a-4.

```

This should produce an error since the following rules in SAA are not met regarding when an INTO phrase is allowed:

1. Only one record description is subordinate to the file description entry.
 2. All record-names associated with file-name-1 and the data item referenced by identifier-1 describe a group item or an elementary alphanumeric item.
- When a file fills up a file system, the file status code returned is 9/28 (No space on device) instead of 24. If the file exceeds the ulimit that is set, the file status will be 9/194 (File size too large).
 - Currently, the compiler wrongly issues the message 232-S Numeric-edited picture string too large for an ALPHANUMERIC-EDITED data item larger than 32767 bytes.
 - The message 232-S Numeric-edited PICTURE string is too large is issued for some declarations such as:

```
77 over-replication-edit PIC 9B(512) VALUE ALL SPACES.
```

The message text implies that the number of characters in the source file for the PICTURE specification exceeds the limit of 30 characters. The message should say that the PICTURE specifies a storage area that is too large for the data item. For SAA, the maximum number of characters for a data item of type NUMERIC EDITED is 127.

Using Intermediate or Native Code

If you wish to run your code in an unlinked environment, you can choose whether you wish the `cob` command to output intermediate or native code. See Chapter 4, “The COBOL Interface,” for details of how you can do this.

To maximize performance, compile your programs to native code, since native code programs execute much faster than intermediate code programs. However, a program that is *I-O bound* (that is, spends most of its time moving data to and from files and devices rather than performing arithmetic) derives less benefit from faster code generation. Programs that are *processor bound* (that is, spend most of their time operating on data rather than transferring it) are likely to increase their run-time speed significantly as a result of code generation.

Although native code gives better performance than intermediate code, you should be aware that the native code version of a program takes up more space than its intermediate code equivalent, since intermediate code is very compact. In fact, the only difference between intermediate and native code files lies in the code area; the data areas are identical. Typically, the code area of a native code program is a little less than twice the size of the code area of the equivalent intermediate code program.

Native code cannot be animated. The ANIMATOR operates only on the intermediate code versions of your programs. AIX system debuggers can be used to debug your native code. See the information on the `-g` option in Chapter 4, “The COBOL Interface.”

If your program is divided into a main COBOL program and a number of subprograms called from the main program, some of the programs can be processed to intermediate code and others to native code. You can mix the two freely.

Optimizing Native Code

The following information increases the performance of the COBOL programs compiled to native code.

- Operations on items of USAGE COMP-X are the fastest types of operations; operations on items of USAGE COMP are the next fastest type of operation; operations on items of USAGE DISPLAY the next; while those on items of USAGE COMP-3 are the slowest.
- MOVEs between items of USAGE COMP are faster if you specify **notrunc** or **trunc** "ANSI" rather than **trunc**. See Chapter 5, "Compiler Options" for details of these options.
- Operations on items which have the same PICTURE, USAGE, and alignment in memory are very fast. See details on the SYNC clause in *Language Reference*.
- Operations on items are faster if the operands are of nine digits or less.
- Performance is improved when operations are used on items of 1, 2, 4 or 8 bytes in length.
- You will find the following statements, phrases, and operations relatively slow:
 - MULTIPLY
 - DIVIDE
 - Exponentiation
 - COMPUTE
 - ON SIZE ERROR
 - ROUNDED

In particular, decimal operations such as DISPLAY and COMP-3 are slower because of the computations that are needed for decimal precision.

- Specifying the **ibmcomp** compiler option may speed up many operations, including complex arithmetic operations, such as COMPUTE statements. See Chapter 5, "Compiler Options" for details of the **ibmcomp** option.

Handling Large Programs

The AIX VS COBOL system allows you to execute statically linked or dynamically loaded code. Statically linked code is a standard AIX **a.out** executable object module which has all of its overlays linked into memory. Dynamically loaded code, either a COBOL intermediate code file or a COBOL native code file, loads its overlays as needed.

When designing a COBOL application program that is to be dynamically loaded, you can make efficient use of the available memory. This chapter describes the ways in which even quite large applications can make use of limited memory space. These techniques are:

- *Segmentation*, in which you divide the Procedure Division code into segments.
- *Interprogram communication*, in which you design an application as a set of separately compiled programs passing control to one another by means of CALL statements.

Segmentation (Overlaying)

AIX VS COBOL provides syntax to divide a COBOL program with a large Procedure Division into a COBOL program with a small Procedure Division and a number of segments containing the remainder of the Procedure Division. This was a feature on older systems to make effective use of small memory sizes.

The AIX system is a virtual memory system. This means that the operating system itself has a very robust system of memory management, and that programmers do not need to manage segments themselves. Nonsegmented code is more efficient in this environment.

AIX VS COBOL will accept COBOL programs either with or without segmenting syntax. If you are creating .int code (intermediate code) files to be executed, you have a choice of using either real or simulated segments (by selecting either the **seg** or **noseg** option during compilation). When segmentation is used, extra intermediate code files are generated by the AIX VS COBOL system as follows:

```
filename.inn
```

where:

filename is the name without the extension of the principal intermediate code file, and

nn is a segment number that identifies the particular segment.

A separate intermediate code file is generated for each independent segment. Due to the efficiency of the AIX virtual memory system, when native code is being generated, either .gnt or .o code, simulated segments will always be used. This means that for native code, individual, small modules for each segment are not produced. Instead, the entire program is created as a single large code module. See Chapter 5, "Compiler Options" for the options to use to determine which kind of code to produce (.int, .gnt, or .o).

Interprogram Communication (Call and Cancel)

You can design an application as a group of independently compiled subprograms that pass control to one another by means of the **CALL** statement. The main programs and the subprograms in your application can be written in the COBOL language or in some other language, such as C. You may mix these programs freely in an application. However, before you call a non-COBOL subprogram from an AIX VS COBOL program, you must link it to the COBOL libraries. An intermediate code program can call a native code program and vice versa.

Figure 2-1 on page 2-10 shows a sample application using interprogram communication.

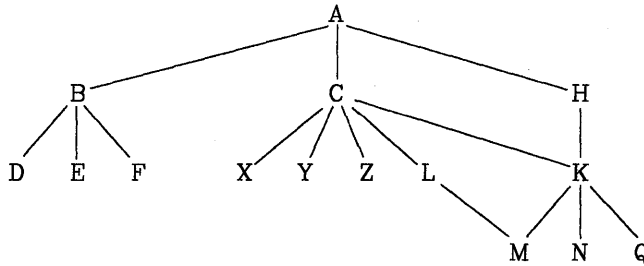


Figure 2-1. Sample CALL Tree Structure

The main program A, which is permanently resident in memory, calls B, C, or H, which are subsidiary and stand-alone functions within the application. These programs call other specific functions as follows:

- B calls D, E, and F
- C calls X, Y, Z, L, or K conditionally
- H calls K
- K calls M, N, or Q conditionally
- L calls M if it needs to

As the functions B, C, and H are stand-alone, they do not need to be permanently resident in memory together, and can therefore be called as necessary, using the same physical memory when called. The same applies to their subfunctions in the tree structure.

The CANCEL statement releases dynamically allocated memory occupied by the cancelled program and closes any files opened by it. The memory is released to either the dynamic loader for later use or to the AIX operating system for use by other processes. This is dependent on the setting of the memory run-time switch. See "Memory Switch (1)" on page 7-10 for information on that run-time switch.

In the example shown in Figure 2-1, you would plan the use of CALL and CANCEL so that a frequently called subroutine such as K would be kept in memory to save load time. However, because it is called by C or H, K cannot be initially called without C or H in memory. Thus, the larger of C or H should call K initially so as to allot space. It is also important to avoid overflow of programs. At the level of X, Y, and Z it does not matter in which order loading takes place; these programs do not make calls at a lower level.

Called programs that open other files should be left in memory so that files do not have to be reopened on every call. Note that EXIT PROGRAM does not close files, but CANCEL does.

The CANCEL statement releases dynamically allocated memory occupied by the canceled program and closes any files opened by it. The format of this statement is described in the *Language Reference*.

The considerations mentioned above about memory usage and load time are only potentially involved when dynamically loaded modules are used. Even then, the AIX Operating System can handle any memory management requirements.

Calling AIX VS COBOL Subprograms

You can call an AIX VS COBOL subprogram by using either of the following formats of the CALL statement:

```
CALL "literal" USING ...
```

or

```
CALL dataname USING ...
```

For COBOL programs, the literal string or the dataname is the PROGRAM-ID, the entry-point name, or the base name of the source file (that is, the file name without any extension). For statically linked modules, the Native Code Generator converts calls with literal strings to subroutine calls which refer to external symbols. If the symbol has not been defined when linking is performed, it is assumed to be the name of a file to be dynamically loaded, provided you specify the `-U` option on the `cob` command line. See Chapter 4, “The COBOL Interface” for details. The AIX VS COBOL system automatically creates a routine to define the symbol and to load the associated file if it is entered at run time.

A CALL literal statement to a statically linked program results in the relevant program being called directly.

When you make a CALL using a literal to represent a user-defined subprogram, the literal must be numeric and in the range of “01” to “127”. Calls to user-defined subprograms are supported for compatibility with older code. In some previous implementations, this was the only method available to call C code programs. You can now call C code programs by using the name of the C program. Making calls using numeric literals to represent user-defined subprograms is not recommended.

Calls to user-defined subprograms are only supported for .int code. Neither form of native code (.gnt or statically bound) will support this method of calling. The mechanism for passing parameters using user-defined subprograms is very inefficient. (It uses *calargc* and *calargv* and sets up the parameters on the stack.) Native code is intended to produce the best performance possible. Therefore, using the user-defined subprogram method of passing parameters is not appropriate for native code.

When you make a call using a literal to represent a user-defined subprogram, you must add code to the file “usercall.c”, which is delivered with AIX VS COBOL. It is in the \$COBDIR/src directory. This file contains a routine, “xequcall”, which must be set up to recognize the number of the call you wish to make, and to take the appropriate action. This is usually just to code a normal call to your C code. There are more details on “usercall.c” in the comments at the top of that file.

See the *Language Reference* for more information on CALLs.

Search Sequence for Locating File Name

A CALL literal statement to a dynamically loaded program or a CALL dataname statement causes the RTE to search for the called program. The search sequence followed by the RTE to find the named file is as follows:

1. The RTE searches through the entry-points of all COBOL programs which have already been CALLED but have not yet been CANCELED.
2. If it cannot find the named file, the RTE searches through the entry-points of all statically linked programs (both COBOL and non-COBOL).

-
3. If it still cannot find the named file, the RTE searches the fixed-disk and tries to find a suitable file from which the program could be loaded. It searches for the file directory by directory, then extension by extension in each directory. If you specify a directory path, the RTE searches for the file only in the named directory.

Before completing the search operation, the RTE splits the required program name into its component parts: directory, base name, and extension. The RTE does not use the directory portion of the program name in its search of loaded programs. It compares the base name with the entry-point names of all loaded programs. If you specify no extension, the first matching name that the RTE finds is assumed to be the program you wish to CALL. If you do specify an extension, the extension of the loaded program must be the same if a match is to be made. Alternate entry-points to programs are treated as if they had the extension of the file from which they were loaded.

When the RTE searches the table of programs linked to it, it uses only the base name of the specified file.

When CALL specifies a file with no path name:

1. The RTE first searches for the named file in the directory from which the calling program was loaded.
2. If no match is found, the RTE searches the directories specified by the COBPATH environment variable. (See Appendix A, "Environment Variables.")
3. If a match is still not found, the RTE searches the directory specified by the COBDIR environment variable. (See Appendix A, "Environment Variables.")

If you specify a file extension, the RTE will search only for a file with a matching extension. However, it is not recommended that you include the extensions `.int` and `.gnt` in the file names you specify to the CALL statement. If you specify a file without an extension, the RTE uses the following algorithm to search for it:

1. It searches for the named statically linked file in memory.
2. If the file is not linked with the COBOL libraries, the RTE adds the extension `.gnt` to the base name of the file and tries to find the corresponding native code file on fixed-disk.
3. If it cannot find the native code file on fixed-disk, it adds the extension `.int` to the base name of the file and searches the fixed-disk for the corresponding intermediate code file.

The RTE always assumes that the first matching program name which it finds is the program you want to CALL.

If no matching program is found, a run-time error occurs.

Note that if the first character of a file name that is to be dynamically loaded at run time is "\$", the string of characters from the "\$" to the first "/" character is treated as an AIX environment variable, and is replaced by the value of that variable. See Chapter 3, "Device- and File-Handling," for details.

For example, if the statement:

```
CALL "$COBDIR/A"
```

is found in the source program, A is loaded from the path given in COBDIR at run time.

Multiple Entry-Points

AIX VS COBOL allows you to define multiple entry-points in a COBOL program using the ENTRY statement. You can CALL a program either via the main entry-point (at the start of the Procedure Division) or via one of the points in the program marked by an ENTRY statement. See the *Language Reference* for a description of the ENTRY statement.

Multiple Entry-Points in Dynamically Loaded Programs

You can CALL a dynamically loaded program via an entry-point in the same way that you would call it via its main entry-point. For example:

```
Procedure Division using param-1,param-2.  
first-para.  
:  
:  
:  
entry "other" using param-3,param-4,param-5.  
:  
:  
:
```

Using multiple entry-points in programs is regarded in many circles as bad programming practice. If you do use multiple entry-points, avoid entering a program for the first time via an entry-point other than the main entry-point.

If you compile the above program into an intermediate code file **mainprog.int** (or dynamically loaded code file, **mainprog.gnt**), then you can CALL it via its main entry-point, as follows:

```
CALL "mainprog" USING PAR-1,PAR-2.
```

At some later point CALL the same program via its other entry-point, as follows:

```
CALL "other" USING PAR-3,PAR-4,PAR-5.
```

It is recommended that if you do use multiple entry-points, you avoid entering a program for the first time via an entry-point other than the main entry-point. You need to be aware of the following features when calling dynamically loaded programs via entry-points.

When the RTE loads the program called via its main entry-point, it notes the names of any other entry-points within the program. When you subsequently CALL the same program via its other entry-point, the RTE can detect that the program containing this entry-point is already loaded, provided that you have not used the CANCEL statement to release the memory occupied by the program after the first CALL.

If, on the other hand, your first entry to a program is via the entry-point "other" rather than by the main entry-point, the RTE will not be able to associate the entry-point "other" with the program **mainprog.int**, and the RTE will be unable to load the program. You can solve this problem by creating a link between **mainprog.int** and the entry-point "other" using the AIX command **ln**, as follows:

```
In mainprog.int other.int ◀
```

If you are calling via an entry-point in an overlay segment, you must also establish a link between the intermediate code overlay file and the entry-point. For example, if you are calling via an entry-point "other" in **mainprog.int** that is located in a section with segment number 52, you must create a link as follows:

```
In mainprog.i52 other.i52 ◀
```

If you are animating a program that is being entered initially by an entry-point other than the main entry-point, you must establish a link between the **.idy** file used by ANIMATOR and the entry-point. For example:

```
In mainprog.idy other.idy ◀
```

However, you may still experience problems if you want to **CALL** the program again later using the main entry-point (rather than "other"). When you **CALL** a program via any of its entry-points, the RTE picks up the references to all its other entry-points. Unless your program has a program name assigned to it in the **PROGRAM-ID** paragraph, the main entry-point will not have a name associated with it. The consequence is that if you **CALL** the above program by:

```
CALL "other" USING...
```

and later call the same program again by:

```
CALL "mainprog" USING...
```

the RTE, unaware of a main entry-point, does not detect the already loaded program and loads a duplicate copy. This can cause a severe problem by duplicating the data in the program.

When the RTE loads a program, it initializes the area of memory holding the program data so that the data is initially either undefined or has the initial values assigned to it by the **VALUE** clauses in the **Data Division**. If you exit from the program without **CANCEL**ing the memory it occupies, when you reenter the program its data will be in the state in which the program left it.

If the RTE loads a duplicate copy of the above program because it is not aware that the program has already been loaded, it will initialize the data in the program. This means that when you enter the program the second time (via the main entry-point), the data in the program will reflect none of the changes made to it during the first entry to the program (via the "other" entry-point).

You can make the RTE aware of the main entry-point of a program that you enter via another entry-point by including a program name in the **PROGRAM-ID** paragraph in the **Identification Division** of your program. For example:

```
identification division.  
program-id. mainprog.
```

Now when you CALL the program, the RTE will be aware of the entry-point "mainprog". Consequently, to reenter the program successfully via the main entry-point after having entered it first via the "other" entry-point, you would have to use:

```
CALL "MAINPROG" USING...
```

In order to ensure complete portability of your applications, use only digits and letters in your PROGRAM-ID and entry-point names. The first character of a program name (that is, the file name of the source code, PROGRAM-ID, name, and any ENTRY "... USING names) must be alphabetic. If it is numeric it may be converted to an alphabetic character as follows:

```
0 converts to J  
1-9 converts to A-I
```

This applies only if the digit is in the first character position of the program name. However, if your program name contains a hyphen, it is converted to zero, regardless of its position in the name.

For intermediate code and dynamically loadable native code, the RTE maps the names of all calls before searching for the program in memory. If the program is not found in memory, an attempt is made to load it from the filesystem using the unmapped name. After loading, the program name and entry-point names are held by the RTE in their mapped format.

For statically bound native code, the PROGRAM-ID, all alternate entry-points, and all CALL *literal* names are mapped. The RTE searches for the mapped name in memory. If it is not found in memory, the RTE searches for the mapped name in the filesystem. If the mapped name is not found in the filesystem, no attempt is made to unmap the name and search for it. A statically bound native code CALL *identifier* behaves the same as described for intermediate code.

You must not use entirely numeric call names as COBOL PROGRAM-IDs, since these are reserved for user calls.

Calling Non-COBOL Subprograms

You can access non-COBOL (C and Assembler) subprograms using the standard COBOL CALL ... USING statement. The address of each USING parameter is passed to the argument in the non-COBOL subprogram that has the same ordinal position in the formal parameter declarations. You must ensure that all formal parameter declarations are pointers.

In the following example, C functions are accessed from a COBOL program.

```
$ SET OSVS
*
* Enables reserved word RETURN-CODE which is an
* OSVS special-register
*
WORKING-STORAGE SECTION.
01 STR.
   03 STR-TEXT      PIC X(10).
   03 FILLER        PIC X VALUE X "00".
*
* NULL TERMINATE STRING FOR C FUNCTION
*

01 COUNTER  PIC 9(8) COMP VALUE ZERO.

PROCEDURE DIVISION.
CALL-C SECTION.
   CALL "cfunc" USING STR, COUNTER.
   IF RETURN-CODE NOT = ZERO

*
* RETURN-CODE SET FROM RETURN () IN C
*

   DISPLAY "ERROR"
ELSE
   DISPLAY "OK".
STOP RUN.
```

```
-----

cfunc (st, c)
char  *st;
int   *c;
{
.
.
.
return (0);
}
```

All non-COBOL subprograms you wish to call from COBOL must be statically linked to the RTE using the **cob** command. The format of the **cob** command you use determines whether all COBOL programs invoked with this RTE, or only the specified COBOL program, can access the non-COBOL programs linked to it. For example:

```
cob -xe "" cprog.c -o rts ◀
```

allows all COBOL programs invoked with **rts** to access the C functions linked to **rts** since the entry-point is null and can be supplied at run time, while:

```
cob -x cobprog.cbl cprog.c -o cobprog ◀
```

allows only the specified COBOL program (**cobprog.cbl**) and any of its called subprograms to access the C functions linked to the RTE. See Chapter 4, “The COBOL Interface,” for information on the use of the **cob** command.

When you use the **CALL** statement from within a COBOL program to access a non-COBOL module as described above, you must ensure that the COBOL run environment is not accidentally damaged. This means you must ensure that:

- The called module preserves the local COBOL run environment (that is, the registers) according to C calling conventions. Refer to the information on subroutine linkage and system calls in the AIX operating system documentation for allocation of registers over calls and a definition of which registers should be preserved and which can be used as work registers.
- The global COBOL run environment (that is, data areas allocated by the COBOL system, such as open file, buffers, and environment variables) should only be destroyed or altered under the direct control of the COBOL system. The routine **cobtidy()** is provided to tidy up the global COBOL run environment. You can call this routine from non-COBOL modules to empty buffers, close files, and free any data areas allocated by the COBOL system. Call **cobtidy()** when all COBOL modules have been exited and you do not intend to reenter them. You may use this routine if you wish to close down the COBOL system but are not yet ready to exit to the operating system -- for example, before you execute the **exec()** routine. Do not call **cobtidy()** directly from COBOL, as this gives undefined results. See Chapter 9, “Advanced Programming Features,” for more information about **cobtidy**.
- The COBOL run environment (that is, the memory map image of the current terminal screen) is not aware of any changes to the screen which the non-COBOL module may make. Also, the stty settings required by the COBOL run environment may be different from those required by the non-COBOL module. The non-COBOL module is responsible for saving and restoring the COBOL run environment. The demo programs **call_sys.c** and **call_sys.cbl** illustrate how this can be accomplished. All demo programs are located in the **\$COBDIR/demo** directory.

When you wish to shut down the current COBOL environment and start another, you should use the **CHAIN** statement. While you are in the COBOL system but not within a COBOL module, you should use the **cobexit()** routine to return to the operating system.

Canceling Non-COBOL Subprograms

The CANCEL statement has no effect when it references a non-COBOL program.

Mixing C and COBOL Programs

A C program can call a COBOL program in the same way as it would call another C program. In the following example the COBOL program name is called using the arguments a and b:

```
name (a, b);
```

The following functions are also provided to allow you to mix C and COBOL programs in an application. Parameters are passed by reference:

```
cobcancel (name)  
char *name;
```

The above function cancels the COBOL program name previously called. It leaves the data contained in this program in the initial state as defined for the COBOL CANCEL verb. See the *Language Reference* for more information.

```
cobfunc (name, argc, argv)  
char *name;  
int argc;  
char **argv;
```

This function has the same effect as specifying the previous two examples. However, this function, unlike the previous example, causes the program to behave as if it had been called using C function rules, not COBOL CALL rules.

```
cobexit (exitstatus)  
int exitstatus;
```

This function allows the terminal to be reset if ADIS was used in a called COBOL program. It also terminates the program's run in the same way as if a COBOL STOP RUN statement had been executed.

Example

The following example shows a C program calling a COBOL program. The example demonstrates how to:

- Pass a string from C to COBOL
- Pass a number from C to COBOL
- How a called COBOL program can keep its data active
- Use the powerful COBOL editing facilities from C
- Animate a COBOL program called from C
- Use the symbolic debugger **dbx** to debug a C program calling COBOL.

account.cbl

\$set OSVS

program-id.
account.

data division.

working-storage section.

78 account-name-len value 80.

01 account-name pic x(account-name-len).

01 total pic 9(9) value zero.

01 result pic \$\$\$,\$\$\$,\$\$9.

linkage section.

01 strlen pic x(4) comp-5.

01 newname pic x(80).

01 next-item pic x(4) comp-5.

procedure division.

display spaces upon crt.

exit program.

entry "validate" using strlen newname.

if strlen > account-name-len

display "account name exceeds ", account-name-len,
"characters."

move 1 to return-code

else

move newname(1:strlen) to account-name.

exit program.

entry "tally" using next-item.

add next-item to total

on size error

display "numeric overflow"

move 2 to return-code.

exit program.

entry "showaccount".

display spaces upon crt.

display account-name

move total to result.

display result.

exit program.

```

cmain.c
-----
#include <stdio.h>

#define BUFFSZ          80                /* temp buffer size */

extern int account();                    /* Cobol program - initialization */
extern int validate();                   /* Cobol program - takes account name */
extern int tally();                       /* Cobol program - increments total */
extern int showaccount();                 /* Cobol program - controls displays */
extern void cobexit();                    /* close down Cobol system and exit */
extern int cobprintf();                   /* COBOL display from C */
extern int cobgetch();                    /* COBOL character get */

main()
{
    int status;
    long num;
    char buf[BUFFSZ];
    int strlen();

    if (status = account())                /* Call COBOL to initialise */
        cobexit(status);

    cobprintf("account: ");                /* select account code */
    get_string(buf);

    num = strlen(buf);
    if (status = validate(&num, buf))
        cobexit(status);

    do
    {
        cobprintf("cost [0 to end]: ");
        get_string(buf);
        num = atoi(buf);                    /* tally items */
        if (status = tally(&num))
            cobexit(status);
    } while (num != 0);

    showaccount();                          /* display total */

    cobexit(status);
}

get_string(buffer)
char buffer[];
{
    int=0;

    while (((buffer[i] = cobgetch()) != '\n') && i < BUFFSZ)
        cobprintf("%c",buffer[i++]);
    cobprintf("\n");
    buffer[i] = 0;
}

```

If you wish to statically link and run the programs used in the above example you would type:

```
cob -x cmain.c account.cbl <|
```

To link the C program, **cmain.c**, to the COBOL libraries and run the above programs you would type:

```
cob -Uo crts cmain.c account.cbl <|  
crts <|
```

To allow animation of the dynamically loaded COBOL modules you would type:

```
cob -Uo crts cmain.c account.cbl <|  
COBSW=+A <|  
export COBSW <|  
crts <|
```

To use the symbolic debugger **dbx** to debug the C program you would type:

```
cob -gx cmain.c account.cbl <|  
dbx cmain <|
```

To use the symbolic debugger **dbx** to debug the C program and animate the COBOL program, you would type:

```
cob -gU cmain.c account.cbl <|  
COBSW=+A <|  
export COBSW <|  
dbx cmain <|
```

Passing the Command Line

The AIX VS COBOL system allows you to call a program and pass the command line to the main program as a parameter to be accessed via the Linkage Section. The main program in a run-unit is the first program within it; that is, the one which is called directly by the AIX system. The command line parameter, in the format shown below, is passed to the Linkage Section of the main program:

```
01 CMD-LINE.  
  02 ARGC PIC 9(4) COMP.  
  02 ARG.  
  10 ARGS PIC X OCCURS 0 TO 65535 DEPENDING ON ARGC.
```

To be able to access this example parameter, the main program must declare the above area in its Linkage Section and must have the following Procedure Division header:

```
PROCEDURE DIVISION USING CMD-LINE.
```

This causes the main program to be invoked as though the system program which had invoked it were a COBOL program calling with a CALL statement of the form:

```
CALL "program name" USING CMD-LINE.
```

You can substitute your own names for the items shown in the above example, but you must use a format which is similar to that shown here.

ARGC contains a count of the actual number of occurrences of ARGV, that is, the number of characters on the command line, and you must not access data beyond this. It is recommended that you test that the length field contains a non-zero value which does not exceed the maximum limit of the occurs. You should take care not to access data beyond the end of the command line (for example, by defining a fixed length field and then MOVEing it) as this would be an illegal reference, and could give you a hardware error on some systems.

Consider the following example:

```

WORKING STORAGE SECTION.
01  ARGV          PIC X(20).
01  ARGV-LENGTH  PIC 9(4) COMP.
01  ARGV-MAX-LENGTH PIC 9(4) COMP VALUE 20.
01  NEXT-ARGV    PIC 9(4) COMP VALUE 1.
LINKAGE SECTION.
01  CMD-PARAM.
03  CMD-LENGTH  PIC 9(4) COMP-X.
03  CMD-LINE.
05  CMD-CHAR    PIC X OCCURS 1 TO 999 DEPENDING ON CMD-LENGTH.
PROCEDURE DIVISION USING CMD-PARAM.
A000 SECTION.
  IF CMD-LENGTH = 0
    DISPLAY "No command line  ".
  IF CMD-LENGTH > 999
    DISPLAY "Command line too long" STOP RUN.
  PERFORM UNTIL NEXT-ARGV > CMD-LENGTH
    UNSTRING CMD-LINE DELIMITED BY ALL " " INTO ARGV
    COUNT IN ARGV-LENGTH WITH POINTER NEXT-ARGV
    IF ARGV-LENGTH > ARGV-MAX-LENGTH
      DISPLAY "Argument too long"
    ELSE
      PERFORM PROCESS-ARGV
  END-PERFORM.
  ...
PROCESS-ARGV.
  ...

```

To ensure that your program is portable, you must use the OCCURS DEPENDING clause. If you do not use this clause, characters after the end of the specified command line length may be accessed, which may give a memory validation error on some systems.

The length of the command line is held as a two-byte integer which can hold values larger than the COBOL picture.

Calling Operating System Functions

The following example shows how parameters can be passed to AIX system service routines.

- * Example of direct calling of C routines from a COBOL program using
- * sleep (), an operating system service routine, and getenv(), a general
- * library routine.

```
$set rtncode-size(4)
```

- * Note that the return code size of 4 bytes is required for
- * returning a pointer so set the compiler directive. This
- * is the default for the AIX VS COBOL system.

```
working-storage section.
```

```
01 errno is external pic 9(9) comp-5.
```

- * errno is the external AIX data item to which the error number
- * returned by an AIX system service routine is assigned.

```
01 sleep-time          pic 9(9) comp-5.
```

```
01 term                pic x(100) value spaces.
```

```
01 env-name            pic x(100).
```

```
linkage section.
```

```
01 namebuf             pic x(100).
```

```
01 return-code2.
```

```
    05 return-pointer usage is pointer.
```

- * Linkage items have no physical storage but the names can
- * be used to reference addresses given by the SET verb.
- * Return-code2 is used to reference return-code and redefine it
- * as a character pointer named as return-pointer.
- * Return-pointer is then used to dereference the pointer and set
- * namebuf to point to the character string associated with the
- * pointer returned by the call to getenv().

```
procedure division.
```

```
get-cobdir section.
```

```
*
```

```
    set address of return-code2 to address of return-code.
```

```
    move 0 to errno.
```

```
*
```

- * "getenv()" expects a pointer to an array of characters terminated
- * by a low-value.

- * Cobol can pass its parameters by REFERENCE, CONTENT, or VALUE:-

- * BY REFERENCE will pass to the function the address of the parameter

- * (in C a PIC X(n) would look like a char*, except it would not be

- * NULL terminated).

```

* BY CONTENT will pass to the function the address of a temporary data
* item (to which there is an implied move from the parameters before
* the call is made). The only difference between BY CONTENT and BY
* REFERENCE is that the called module cannot effect the value of the
* parameter as seen from the calling module.
* BY VALUE will pass to the function the actual value of the data item
* rather than its address. BY VALUE should only be used to call non-COBOL
* modules (because the PROCEDURE DIVISION USING statement has no way of
* specifying that a VALUE parameter is to be expected). Note that if
* the size of the parameter being passed exceeds 4 bytes then it will
* be passed as if BY REFERENCE has been specified, also any numeric
* literals passed in this way will be passed to the called module
* as a 4 byte comp numeric in machine byte order (in C as a long on
* a 32 bit machine).
* EG:
    display "about to sleep".
    move 10 to sleep-time.
    call "sleep" using by value sleep-time.
    display "have had a very nice sleep thanks".
*
* Now back to demonstrate how to find the value of the environment
* variable TERM and display it.
*
* Ensure that parameter to "getenv()" is NULL terminated.
    string "TERM" low-values delimited by size into env-name.
*
    call "getenv" using env-name.
    if return-code = 0
        display "TERM not found"
    stop run.
*
    set address of namebuf to return-pointer.
*
* Function result of "getenv()" is a NULL terminated string. Cobol
* requires SPACE termination.
    string namebuf delimited by low-values into term.
*
    display term.
    stop run.

```

AIX VS COBOL Dialect Flagging and Error Reporting

When you select a particular COBOL dialect to use when you compile your program, for example, **vsc2** or **mf**, the additional reserved words in that dialect are enabled. When a particular dialect is turned off, for example, **novsc2** or **nomf**, the reserved words associated uniquely with that dialect are disabled. If a reserved word is used in some other context in a dialect that is still enabled, all functionality for that reserved word will still be available, that is, no error will be issued for any use of that reserved word.

For example, the reserved word ON is used in many contexts. One of these is the ON statement, which is part of the `osvs` dialect of COBOL. However, the reserved word ON is used elsewhere in COBOL, so that even when the option `noosvs` is used, all uses of the reserved word ON will be accepted by AIX VS COBOL. For instance, you will still be able to code the ON statement even when the `noosvs` option is used, since the reserved word ON is not unique to the dialect of COBOL that is disabled.

On the other hand, the reserved word OTHERWISE is unique to the dialect `osvs`. Therefore, the option `osvs` must be set for any syntax using the word OTHERWISE to be accepted. If the word OTHERWISE is coded and the `osvs` option is not on, (either by default or by explicit action), you will get a severe error reported.

Other examples of being able to use syntactic constructions even though they belong to a dialect that is not selected are:

- Omitting optional reserved words. Since no reserved word is used wrongly, there is no error to report.
- Using syntactic constructions that do not involve reserved words, for example, reference modification.

You can always determine if you are coding within the bounds of a particular dialect through the use of flagging. Consider the example discussed above regarding the reserved word ON. If you wanted to code only to the ANSI 1985 COBOL standard and you used `flag(ans85)` on your compilation, the use of the ON statement would be flagged as being outside of the chosen dialect. The additional examples cited above would also be flagged if the flagging dialect selected did not allow the syntax you used.

The choice of a dialect of COBOL should be viewed as a means of enabling or disabling specific reserved words. It will not necessarily report errors for coding outside of the chosen dialects, unless the syntax used does not exist in any dialect. In order to keep your coding within a specific dialect or standard, you should use flagging in addition to the proper dialect options.

See Chapter 5, "Compiler Options" for more information on the available options and how to set them for compilation.

Most of the time, the error messages reported for your compilation are interspersed into your source code when seen on the listing. However, some errors cannot be detected until the entire program has been scanned. Since the AIX VS COBOL compiler is a one-pass compiler, it is not possible for errors such as these to be reported interspersed in the source. These errors will be reported at the end of the source listing.

Chapter 3. Device- and File-Handling

Contents

About This Chapter	3-3
Devices	3-4
File Assignment	3-4
Special Characters in Environment Variables	3-9
AIX VS COBOL Disk File Structure under AIX	3-12
Record-Sequential Files	3-12
Line-Sequential Files	3-12
Relative Files	3-12
Indexed Sequential Files	3-13
Library Files	3-16
File Restrictions	3-17
Input-Output Error-Handling (File Status)	3-17
Alternate File Status Table	3-18
Writing Output Directly to a Printer	3-20

About This Chapter

This chapter describes some aspects of AIX files and devices as they appear to AIX VS COBOL programs. In particular, it describes:

- How to use special devices recognized by the RTE
- How to assign files in a program, statically and dynamically, to files and devices
- How AIX VS COBOL file structures map onto AIX file structures
- How the COBOL COPY statement operates in an AIX environment
- How to handle input-output errors.

Devices

The compiler and the RTE are programmed to use certain devices. A program that reads from the standard input device **stdin** will access the standard AIX input. A program that writes to the standard output device **stdout** will access the standard AIX output. A program that writes to the standard error device **stderr** will access the standard AIX error output. All AIX VS COBOL utilities write error messages to error output, and not to standard output. At run time, the RTE recognizes sequential or line-sequential files opened with these names and directs output to the appropriate target. Note that if **stdin** is line-sequential, the first READ is from the command line tail.

The system emulates printer channels C01 through C12 by line feeds and form feeds. If you want to write to these channels, set the environment variable **COBLPFORM** to define the line numbers on the form. See Appendix A, "Environment Variables," for details on how to set this variable. The format consists of a series of numbers separated by colons, as in the following example:

```
COBLPFORM = "1:.....:60"
```

This sets channel 1 to line 1 (the beginning of the page) and channel 12 to line 60. You can specify only a single line number for each channel. Those channels which have line number zero; mnemonics S01, S02, CSP; or are undefined, are set to line 1.

Any **WRITE BEFORE/AFTER PAGE** statements cause positioning at line 1. Each line that is advanced increases the line number by 1. A request to skip to a line number less than or equal to the current line causes a new page to begin. The appropriate number of line feeds are then generated.

Any **WRITE BEFORE/AFTER TAB** statements generate a form feed and cause any subsequent skips to a channel number to start a new page.

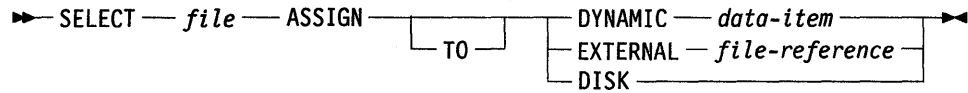
File Assignment

The AIX VS COBOL system offers three types of file assignments:

- **Fixed file assignment**, in which you assign the internal user file name to a literal operating system file name when you write the program.
- **Dynamic file assignment**, in which you assign the internal user file name to a data item defined within your program. You can store the name of an operating system file in this variable at run time and assign the internal user file name to this file.
- **File name mapping**, which allows you to assign files to AIX pipes or to assign the index and data files of indexed-sequential files to different directories.

In the interest of program portability it is advised not to build the full path names of files into your programs. You should use logical file names instead.

You do this in the SELECT clause for the file, which has the form:



where:

data-item is the name of a data item declared in the WORKING-STORAGE SECTION of your program. You must declare this item as PIC X(*n*), where *n* is the maximum length of the file name you want to use. If you do not declare this data item, the compiler declares it for you automatically as PIC X(21).

file-reference is a user-defined name. Before you run a program that contains such a file assignment, you must ensure that you have declared an environment variable with the same name as the name that follows EXTERNAL. You must also ensure that the value of this environment variable includes the file name for the appropriate file. If the *file-reference* contains hyphens, only the positions to the right of the rightmost hyphen have any significance.

If no environment variable is set, the file name used for the AIX file is *file-reference*.

If you use the DISK option in the SELECT clause, you specify the name of the data item that holds the file name in the VALUE OF FILE-ID clause in the FD entry of the file.

The following example shows how to use DYNAMIC:

```
SELECT MYFILE ASSIGN TO DYNAMIC FILE-NAME
:
DATA DIVISION.
:
WORKING-STORAGE SECTION.
:
01 FILE-NAME PIC X(25).
:
PROCEDURE DIVISION.
:
MOVE "mylib/file3" TO FILE-NAME.
OPEN MYFILE...
```

DYNAMIC associates the path and file name `mylib/file3` with the internal file name `myfile`. You can achieve the same effect using DISK:

```
SELECT MYFILE ASSIGN TO DISK
:
DATA DIVISION.
:
FILE SECTION.
:
FD MYFILE VALUE OF FILE-ID IS FILE-NAME.
:
WORKING-STORAGE SECTION.
:
01 FILE-NAME PIC X(25).
:
PROCEDURE DIVISION.
:
MOVE "mylib/file3" TO FILE-NAME.
OPEN MYFILE...
```

The following example shows how you can use EXTERNAL:

```
SELECT MYFILE ASSIGN TO EXTERNAL MYENV
:
PROCEDURE DIVISION.
:
OPEN MYFILE...
```

Before running this program, you would create an AIX environment variable called `dd_MYENV` and give it a value, such as `mylib/file3`. When you run the program, the file with the internal file name `MYFILE` is associated with the AIX file, `mylib/file3` (the current value of the environment variable `dd_MYENV`).

If you use the EXTERNAL feature, you must be careful how you specify file and environment names. The file reference that follows EXTERNAL is not a literal in the usual COBOL sense; the compiler treats it as a user name. One consequence of this is that if the name following EXTERNAL is in lowercase, the compiler converts it internally to uppercase. Therefore, when you create the environment variable its name must be in uppercase.

File Name Mapping

AIX VS COBOL allows file names to be mapped or changed at run time through environment variables. This allows the physical file name to be changed by the user each time the COBOL program is run.

This use of environment variables to re-map a file's name at run time is distinct from the use of environment variables to associate the name of a file declared EXTERNAL with a physical file. If both types of environment variables are used, the one specifying a `dd_` name has precedence.

When a file is opened by a COBOL program, the system checks to see if there is an environment variable defined that will cause a different file to be used. If no environment variable is defined for the ASSIGNED file name, substitution does not take place, and the file is opened as usual.

If an environment variable exists for the ASSIGNED file name, the value of the environment variable is used for the physical file name. The environment variable name searched for is constructed with the first element of the ASSIGNED file name, prefixed with *dd_*. For example, if you try to open a file named *dir/file*, the system searches for the environment variable *dd_dir*. If you try opening a file named *dir1/dir2/file1*, the system searches for *dd_dir1*. And, if you try opening a file named *file1*, the system searches for *dd_file1*.

After the system finds an environment variable name, it takes the value of that environment variable and adds it to the beginning of the remaining elements of the original file name. This name is then the physical file name that the RTE searches for. Consider the examples in Table 3-1:

File Name ASSIGNED in Program	Environment Variable Searched for	Environment Variable Contents	Physical File Name
<i>dir/file1</i>	<i>dd_dir</i>	<i>d2</i>	<i>d2/file1</i>
<i>dir/file1</i>	<i>dd_dir</i>	<i>d4</i>	<i>d4/file1</i>
<i>dir/file1</i>	<i>dd_dir</i>	<i>d2/d4</i>	<i>d2/d4/file1</i>
<i>dir1/dir2/file1</i>	<i>dd_dir1</i>	<i>d2</i>	<i>d2/dir2/file1</i>
<i>dir1/dir2/file1</i>	<i>dd_dir1</i>	<i>d4</i>	<i>d4/dir2/file1</i>
<i>dir1/dir2/file1</i>	<i>dd_dir1</i>	<i>d2/d4</i>	<i>d2/d4/dir2/file1</i>
<i>file1</i>	<i>dd_file1</i>	<i>d2</i>	<i>d2</i>
<i>/dir3/dir4/file1</i>	<i>dd_</i>	<i>d2</i>	<i>d2/dir3/dir4/file1</i>

If you try to open a file whose name begins with the slash (/) character, the system searches for *dd_*.

Do not start any file name with the characters **cob**.

dd_ can be uppercase, lowercase, or mixed case; the RTE recognizes the combinations *dd_*, *DD_*, *dD_*, and *Dd_*. However, you must be careful not to define multiple variables for the same file using different case combinations, because the RTE may select the wrong file name. For example:

```
SELECT FILE1 ASSIGN TO "myfile"
```

When FILE1 is opened, the RTE searches for the environment variable *dd_mYfile*. If it is defined, as in *dd_mYfile = another.file*, the physical file name *another.file* is used for FILE1. Otherwise, *mYfile* is used as the physical file name.

The rules for file name mapping described above allow you to put all the files connected with one application in the same directory and be certain that the AIX VS COBOL system will be able to find them all. You achieve this by defining each file as *application-name/file-name* and by setting up the environment variable *dd_application*, which points to the name of the application directory containing all these files.

Be aware that if you dynamically change any environment variable names, they are not accessed again. These environment variables are accessed only at the start of a run. However, external variables are accessed again.

Special Characters in Environment Variables

Not all special characters used in COBOL file names can be used in environment variables. Four characters, the greater-than symbol (>), less-than symbol (<), colon (:), and ampersand (&), have a special meaning in this context. You cannot map a filename containing these characters using environment variable logical filename mapping. These characters are described in the following sections.

A period (.) in an environment variable must be replaced by an underscore (_) so that file mapping can proceed. For example, to map the COBOL file *file.lst* to the file *my_file.list*, define the environment variable as follows:

```
dd_file_lst=my_file.list
```

Indexed Files

COBOL indexed files are implemented as two files: a data file and an index file. For example, the COBOL indexed file *id_file* has a data file named *id_file* and an index file named *id_file.idx*. An environment variable *dd_id_file = x* has a data file *x* and an index file *x.idx*. If you want to change the names of the data and index files independently, you must use the ampersand (&) character, as follows:

```
dd_id_file="&datafile&indexfile"
```

In the preceding example, the data file is now named *datafile* and the index file is named *indexfile*. When you use & to rename files, you can place data and index files in separate directories to increase performance capabilities.

Notes:

1. Double quotation marks (“ ”) must be used with the & character.
2. The & character can only be used with indexed files.
3. You cannot use the & character when specifying multiple paths. (See “Multiple Files” on page 3-10).

The "\$" Character in Filenames

If the physical filename in your program starts with a "\$" character, this forces the system to attempt to map the specified file. If no mapping exists a "file not found" condition is returned; the system does not search for the unmapped filename. Consider the following example contents of a SELECT statement:

```
select filename assign "$file1"
```

This causes the system to search for the environment variable "dd_file1". If this is found the system follows the rules for filename mapping given in the previous sections. If this is not found, a "file not found" condition is returned; the system does not attempt to search for "file1".

Multiple Files

You can set up a search path for files that are opened for reading by using colons (:) to separate alternate paths in a manner similar to the AIX PATH environment variable. For example, the environment variable `dd_my_file = ":file1:dir/file2"` causes the COBOL file `my_file` to use `file1` if it exists at the time the file is opened. Otherwise, `dir/file2` is used.

Notes:

1. The initial colon is required. If the colon is absent, the program searches for the file "`file1:dir/file2`".
2. Do not use this technique for indexed files. Otherwise, an environment variable containing colons (:) will be interpreted as a single file name, and the colons will be interpreted as part of the name of the indexed files.

Warning: This technique works only for files opened for input. If you attempt to use the technique for files opened for output or for I-O, a fatal RTE error will occur.

Redirection and Pipes

When you are specifying the contents of an environment variable you can use the following three characters to set up pipes:

- >
- <
- |

The meanings of these characters when used in the value of an environment variable are described in the remainder of this section. These special characters are only recognized by the AIX VS COBOL system if they appear at the start of the environment variable. You cannot use these characters while specifying multiple paths. You also cannot use these special characters with variable length records, except in line sequential files that have no record varying syntax in the FD; no multiple 01 level data items; and no RECORDING MODE clause.

The < character defines the specified file as a pipe connected to the standard output of the given command. The file ASSIGNED within your program must be either sequential or line sequential, and it must be OPENED for INPUT. Its name can be only one element long, this is, it must not contain a / character.

Consider the following example contents of an environment variable named *dd_dir*:

```
dd_dir="<ls -l ..."
```

This causes every READ in the program of the original ASSIGNED file to return the value of the next line of the output from the `ls -l` command.

The pipe is set up using the AIX `popen()` library routine.

The `>` character defines the specified file as a pipe connected to the standard input of the command. The file ASSIGNED within your program must be either sequential or line sequential, and it must be OPENED for OUTPUT. Its name can only be one element long, that is, it must not contain a `/` character.

Consider the following example contents of an environment variable named *dd_dir*:

```
dd_dir=">pr -h Title ..."
```

This causes every WRITE to the ASSIGNED file in the program to be passed to the standard input of the `pr -h Title` command.

The `|` character defines the specified file as a two-way pipe to the specified process. The file ASSIGNED within your program must be either sequential or line sequential, and it must be OPENED for I-O.

Consider the following example contents of an environment variable named *dd_file*:

```
dd_file="|proc"
```

This defines the file *file* as a two-way pipe to the process "proc". That is, all the read operations on that file will read the standard output of the process "proc", while all the WRITE operations to that file will write to the standard input of the process "proc".

Also consider the following example contents of an environment variable named *dd_UPPERCS*:

```
dd_UPPERCS="|toupper.sh"
```

Then, create a file called **toupper.sh** that contains:

```
tr '[a-z]' '[A-Z]' >out.file
```

and make this file executable.

After you have done these steps, execute your COBOL program that does a WRITE to the file that is ASSIGNED to *UPPERCS*. The shell script **toupper.sh** receives the program's output as input which causes the shell command `tr` to convert the lower-case letters to uppercase and write the results to the file **out.file**.

You receive an error if you attempt to WRITE to a line sequential file or a sequential file OPENED for I-O unless the file has been mapped using the `|` character.

As these three characters do have a special meaning in environment variables, if you wish to reference any file whose name begins with any of these characters, you must precede the name with a `\` character.

AIX VS COBOL Disk File Structure under AIX

AIX VS COBOL offers four types of file organization for use by the COBOL programmer: record-sequential, line-sequential, relative, and indexed sequential.

Record-Sequential Files

Record-sequential files consist of a series of fixed-length records. The length of a record-sequential file record is the length of the longest FD entry for the file in the FILE SECTION of the program.

Normally, the space occupied by a record-sequential file record is the same as the record length as defined in the FD entry. However, if records are written to the file using WRITE BEFORE ADVANCING or WRITE AFTER ADVANCING, extra control characters are written to the file. Programs then will be unable to read the data correctly.

Line-Sequential Files

Line-sequential files are intended to cater to text (ASCII) files created by text editors and similar utilities.

A line-sequential file consists of a series of variable-length records, each of which is terminated by the character hex 0A. From the point of view of a program accessing a line-sequential file, the file record has a maximum length as specified by the FD entry of the file in the FILE SECTION. When your program reads a record from a line-sequential file:

- The record area is padded on the right with spaces if the record is shorter than the maximum record length.
- The record area is filled if the record is longer than the maximum record length. Subsequent READs will fill the record area until the record terminator is read. If the next character after a READ is hex 0A, it is omitted; that is, the READ will not return a blank line.

In both cases the 0A character is stripped from the record; it is not present in the file record area.

If records are written to a line-sequential file using ADVANCING phrases (except for BEFORE 1), the records will contain extra device control characters. Such files cannot be read by a program.

You should store only legal ASCII characters in a line-sequential file.

Relative Files

Relative files allow you to access data randomly by specifying its position within the file.

A relative file consists of a series of fixed-length records, where the length is given by the longest FD entry for the file in the FILE SECTION of the program.

Each record is uniquely identified by a record number. The first record in the file is record number one, the second is record number two, and so on.

At the end of each relative file record there is a one-byte control field (not included in the record length as defined in the FD entry), whose value indicates whether that record logically exists in the file. This control field can have either of two values:

- **hex 0A**

The record exists and can be accessed by a program.

- **hex 00**

The record has been deleted and cannot be accessed by a program.

When you DELETE a record from a relative file all that happens is that the control field of the record is changed from 0A to 00. The data itself remains in the file in its original position.

You can read a relative file by declaring it in a program as a sequential file with record length $n + 1$ (where n is the record length of the relative file). This will allow you to read records that have been deleted. If, for security purposes, you need to ensure that deleted data can never be accessed, you must overwrite the record before deleting it.

Indexed Sequential Files

An indexed sequential file is implemented as two separate files: the data file and the key or index file. The data file is in relative file format.

The name that you supply is the name of the data file; the name of the associated key file is produced by using the extension **.idx** with the root of the data file name. For example:

Data File	Key File
myfile	myfile.idx
clock.fle	clock.fle.idx

It is advisable to avoid using the extension **.idx** in other contexts, and to limit the data-name portion of the file name to 10 characters or less.

The index is built up as an inverted tree structure that grows in height as records are added. The number of key file accesses required to locate a randomly selected record depends primarily on the number of records in the file and the key length.

Faster response times are obtainable when reading the file sequentially, but only if other indexed sequential operations do not intervene.

The necessity of making regular backup copies of all types of files cannot be emphasized too strongly, and this should always be regarded as the main safeguard. There are situations with indexed sequential files (for example, media corruption) that can lead to only one of the two files becoming unusable. If the index file is lost in this way, you can recover data records from just the data file (although not in key sequence) and thus reduce the time lost due to an error. As an aid to this, all unused data records are marked as deleted at the relative file level by appending one byte to each record that contains **LOW-VALUES**. For undeleted records this byte contains the character hex 0A.

The recovery operation may therefore be done with a simple COBOL program by defining the data file as *organization sequential access sequential* with records defined as one byte longer than in the indexed sequential file description. The records are then read sequentially, the data **MOVED** from the sequential file record area into the indexed (sequential) file record area, and written to a new version of the indexed sequential file. Those records with **LOW-VALUES** in the last (extra) byte are discarded. Note that this byte (containing line feed characters in a required record) is not written to the indexed sequential file on recovery because of the record length discrepancy of one byte in the record definitions.

Another way to recover indexed sequential files that you suspect are corrupt is to use the **bcheck** utility. This is described in "The bcheck Utility."

Indexed Sequential File Format

The size of an indexed sequential file depends on the number of records it contains, as follows:

$$m + (n*m) + (K*m) + (F*m)$$

where:

m is the default block size.

n is the number of keys.

K is $508 / (\text{total key length} + (8*n))$.

F is the number of records/126.

This formula gives the minimum size of the index file in bytes. The value of *K* is the number of blocks required to hold the index information. The value of *F* is the number of free list blocks required for the file. These blocks are needed only if the file is fragmented.

The bcheck Utility

You can run the **bcheck** utility to check the consistency of an indexed sequential file. If the index is found to be corrupt, **bcheck** can construct a new index for the file.

To run **bcheck**, enter a command of the form:

```
bcheck [options] file-list ␣
```

where *options* is a string of one or more of the following:

- i Checks the index file only (the default is to check both the **.idx** and **.dat** files that make up an indexed sequential file)
- l Lists the entries in the index binary tree
- n Answers "no" to all questions
- y Answers "yes" to all questions

file-list is a list of the names of the indexed sequential files to be checked.

Unless you specify **-n** or **-y**, **bcheck** is entirely interactive. Each time it finds an error, it asks you whether or not to delete the index.

In order for the **bcheck** utility to reconstruct a file, the index file must exist and cannot be empty. The reconstructed index file produced by **bcheck** is functionally equivalent to the original file.

For example:

```
bcheck -n sale.ship.idx <|
```

checks the index file **sale.ship.idx**. If errors are found, all requests to delete the corrupt index are answered “no”. If such errors are found, you could then delete and rebuild the index as follows:

```
bcheck -y sale.ship.idx <|
```

You should limit the data-name portion of index file names to 10 characters or less. This is because the **bcheck** utility removes any **.idx** extension from a file name, truncates the name to 10 characters, and then adds its own **.idx** extension.

For example, if you enter the following command:

```
bcheck -n mustock.dat
```

bcheck uses *mustock.dat* as the physical file name and the index file name is converted to *mustock.da.idx*. However, if you pass the file *mustock.da.idx* to **bcheck**:

```
bcheck -n mustock.da.idx
```

mustock.da is used for the physical file name, but this file does not exist.

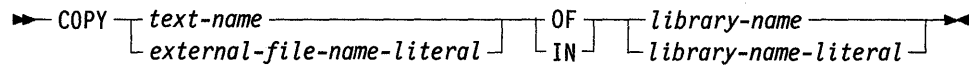
The following error messages may be issued by the **bcheck** utility.

255	Duplicate record	
256	File not open	
257	Illegal argument	
258	Bad key descriptor	(illegal key descriptor)
259	Too many files	(too many files open)
260	Corrupted isam file	(bad isam file format)
261	Need exclusive access	
262	Record or file locked	
263	Index already exists	(key already exists)
264	Primary index	(is primary key)
265	End of file	(end/begin of file)
266	Record not found	
267	No current record	
268	File is in use	(file locked)
269	File name is too long	
270	Back lock device	
271	Can't allocate memory	
272	Bad collating table	
275	NLS Language mismatch	(wrong language nl_init(ed))

Library Files

The COBOL COPY statement allows you to specify the name of a file from which COBOL source programs are read by the compiler when the COPY statement is executed. See the *Language Reference* for a description of how the COPY statement works. This section is concerned with how the specification of the copy file in the COPY statement maps onto the AIX file system.

The copy file in a COPY statement is identified as follows:



where:

text-name is the name of a file without an extension. The compiler searches for this file in the current directory. This name will be converted to uppercase before searching.

external-file-name-literal is the name of a file in quotation marks. This file name may have an extension, and may include a path name if there is no library name or library name literal. The case of this name will be exactly as written in the quoted string.

library-name is a single letter and must be the name of a directory within the current directory. The compiler searches for the file specified in *text-name* or *external-file-name-literal* within this subdirectory.

library-name-literal is a path name in quotation marks. The compiler searches for the file specified in *text-name* or *external-file-name-literal* in the context of this path name.

For example:

```
COPY prog OF A
```

is converted to **A/PROG** (relative to the current directory).

```
COPY "prog.cbl" OF D
```

is converted to **D/prog.cbl** (relative to the current directory).

If the system cannot find the required COPY file, it searches for the environment variable COBCPY. You can use this environment variable to specify a path, or multiple paths, for COBOL COPY libraries. If multiple paths are specified, the first character must be a colon. For example:

```
COBCPY=":/usr/group/sharedcpy:/usr/mydir/mycpy"
```

See Appendix A, "Environment Variables" for more details on COBCPY.

If the system still cannot find the specified COPY file, then the extension specified by the **osex** compiler option is appended to the filename, and a search is made for that name. The default for the **osex** option is ".cbl". See Chapter 5, "Compiler Options" for information about the **osex** option.

File Restrictions

The maximum size of any file you can create is limited by the system parameter **ulimit**. You may find that the default limit, as supplied with your AIX system, is not large enough for your code. However, this limit can be increased by a superuser.

The following remarks indicate some uses of files by AIX VS COBOL:

- Each indexed sequential file counts as two files while it is open.
- Up to five files may be required while a **SORT** or **MERGE** statement is executing, depending on the number of records to be sorted. However, in most cases no files are required at all.
- One file is required for loading an overlay or calling a subprogram. This file will be open only during execution of the **GO TO**, **PERFORM**, or **CALL** statement that causes the load.
- If you are using **ANIMATOR**, it requires two more open files.

Input-Output Error-Handling (File Status)

If you have specified the **STATUS** clause in the **FILE-CONTROL** paragraph in a program and an error occurs during an operation on the file, the status value returned will contain a character "9" in the first byte and the error number in binary (**COMP**) in the second byte. If the operation is successful, the first byte will contain "0". It is your responsibility to check for error conditions and to take appropriate corrective action or to terminate the program run. See the *Language Reference* for a list of file status errors.

If you have not specified the **FILE STATUS** clause in the **FILE-CONTROL** paragraph in a program, file errors with a first byte value of "9" will result in a run-time error being output.

If you wish to display this status with its correct decimal value, careful redefinition of data items is required in order to avoid truncation of the value. This is because the facility that enables the storage of a nonnumeric value greater than decimal 99 as a hexadecimal value is an extension to the ANSI COBOL standard X3.23 (1974), but the rules for moving or manipulating such data are restricted by the standard to a maximum of decimal 99.

The example that follows illustrates one method of retrieving the value of status key 2 for display purposes. Note how truncation has been avoided by redefining the two status bytes as one numeric data item (length two bytes) capable of storing up to four decimal digits.

```

000010 ENVIRONMENT DIVISION.
000020 INPUT-OUTPUT SECTION.
000030 FILE-CONTROL.
000040 SELECT FILE1
000050     ASSIGN "TST.FIL"
000060     STATUS IS FILE1-STAT.
000070 DATA DIVISION.
000080 FILE SECTION.
000090 FD FILE1.
000100 01 F1-REC                PIC X(80).
000110 WORKING-STORAGE SECTION.
000120 01 FILE1-STAT.
000130     02 S1                  PIC X.
000140     02 S2                  PIC X.
000150 01 STAT-BIN REDEFINES FILE1-STAT PIC 9(4) COMP.
000160 01 DISPLY-STAT.
000170     02 S1-DISPL            PIC X.
000180     02 FILLER              PIC X(3).
000190     02 S2-DISPL            PIC 9999.
000200 PROCEDURE DIVISION.
000210 START-TEST.
000220     OPEN INPUT FILE1.
000230     IF S1 NOT = 9
000240         GO TO END-TEST.
000250     MOVE S1 TO S1-DISPL.
000260     MOVE LOW-VALUES TO S1.
000270     MOVE STAT-BIN TO S2-DISPL.
000280     DISPLAY DISPLY-STAT.
000290 END-TEST.
000300 STOP RUN.

```

USE Procedures

If you declare USE procedures in the DECLARATIVE SECTION to handle input-output errors, these procedures are only executed if a FILE STATUS data item is also declared.

Alternate File Status Table

The AIX VS COBOL system comes supplied with a C source file *filestat.c*, which contains tables of the file status values defined by the ANSI 74 and ANSI 85 standards. We recommend that you not alter these two tables in any way. However, this file also contains a table giving an alternate set of file status values for those input-output error conditions which return a value of "9" in the first byte. You can alter this table if you wish. By default, this alternate set of error numbers is that output by RM/COBOL. If you want the AIX VS COBOL system to output error messages from this list, rather than from its standard list of run-time error messages as defined in Chapter 15, "Error Messages," you must either:

- Compile your program with the RM option set (see Chapter 5, "Compiler Options" for details)
- Specify the +Q run-time switch when you execute your program (see Chapter 7, "Running an AIX VS COBOL Program" for details).

If you wish to alter the default table of alternate file status values to a set of values which conform with the statuses returned in the COBOL dialect of your choice by editing the file *filestat.c* in `$COBDIR/src`, you must index the table using the second byte of any status “9” items. The table entry then contains the new value for that file status in Binary Coded Decimal (BCD) format. Any undefined or unrecognized status values are mapped onto status “30”: “permanent I-O error”.

Once you have altered the table you must rebuild the RTE so that it uses your altered version of *filestat.c* rather than the original version. You can either do this globally or individually for each RTE; see “Globally Altering File Status Values” and “Altering File Status Values for Individual Run Time Environments” for details.

Globally Altering File Status Values

To globally rebuild the RTE so that it uses your altered version of *filestat.c* when outputting file status error messages, you must first compile your new version of the module by entering the command:

```
cc -c filestat.c ◀
```

As any further Run Time Environments that you build will use the new version of *filestat.c*, we recommend that you keep a copy of the original version.

Once you have compiled your new version of the module, you must replace *filestat.o* in the COBOL library by entering:

```
ar rv /usr/lpp/COBOL/lib/coblib/libcobol.a filestat.o ◀
```

You must replace `/usr/lpp/COBOL/lib/coblib` with the correct directory if this is different.

You must then rebuild the RTE using the `cob` command. For example:

```
cd /usr/lpp/COBOL/lib ◀  
cob -xvo rts32 ◀
```

See Chapter 4, “The COBOL Interface” for full details on the `cob` command.

Altering File Status Values for Individual Run Time Environments

You can use the `cob` command to rebuild a single RTE so that it uses your altered version of *filestat.c* when outputting file status error messages. For example, enter:

```
cc -c filestat.c  
ln filestat.o filestat  
cob -xvo rts filestat
```

As usual, you can include any COBOL or C programs or other object modules in the command line.

See Chapter 4, “The COBOL Interface” for a full description of the `cob` command.

Writing Output Directly to a Printer

You can code your COBOL program so that it will WRITE output directly to the printer. The writing will go through the print spooler, which means that the output will be buffered until the CLOSE statement. At that time, the entire file is released to the printer.

To get this effect, code the SELECT statement as follows:

```
SELECT myprfile ASSIGN TO EXTERNAL mypr
```

The internal and external file names *myprfile* and *mypr* are arbitrary names created by the user. Notice that *mypr* will be converted to uppercase since it is not a quoted string. In order to be able to write directly to the printer, EXTERNAL is required in the SELECT statement. Alternatively, you can compile with the -C compiler option **assign = external** to get the same effect as EXTERNAL in the SELECT statement.

To run this program and have the output sent to the printer using the spooler, you must use an environment variable to redirect the output to the printer device to be used. This is done as follows:

```
cob -uv prtr.cbl  
dd_MYPR="> /bin/print"  
export dd_MYPR  
cobrun prtr.gnt
```

The above setting of the environment variable is for the ksh shell. To set it under csh, do:

```
setenv dd_MYPR "> /bin/print"
```

You can also specify which printer to use in the dd name:

```
dd_MYPR="> /bin/print lp1"
```

This will direct all of the output from the WRITE directly to the chosen printer.

Example

```
Identification Division.
Program-id. prtr.
*
* Example to write a file directly to the printer.
*
Environment Division.
Input-Output Section.
File-control.
    select myprfile assign to EXTERNAL MYPR
    organization is line sequential
    access is sequential
    file status is filestat.
*
Data Division.
File Section.
FD myprfile.
01 myrec.
    02 info pic x(80).
Working-Storage Section.
01 filestat  pic xx.
01 mydata    pic x.
*
Procedure division.
Action section.
*
* Dummy accept to grab the empty command line to prepare for next
* REAL accept.
*
    accept mydata.
    display "Starting the print test.".
write-it.
    open output myprfile.
    move "This " to info.
    write myrec.
    move "is  " to info.
    write myrec.
    move "my  " to info.
    write myrec.
    move "output" to info.
    write myrec.
*
* Show that the writing is delayed until the CLOSE.
*
    display "We have written 4 records but none should"
    display "have printed yet. Now hit enter to CLOSE"
    display "the file and get the whole file printed.".
    accept mydata.
*
    close myprfile.
*
conclusion.
    display "Finished the print test.".
    stop run.
end program prtr.
```

Chapter 4. The COBOL Interface

Contents

About This Chapter	4-3
COBOL Interface Command	4-4
The Development Cycle	4-5
Option Specification	4-7
System-Wide Default Options	4-7
Optional User Default Options	4-7
Command Line Options	4-8
Embedded Source File Options	4-15
Command Line Conventions	4-16
Command Line Examples	4-17

About This Chapter

The **cob** command provides the interface to the IBM AIX VS COBOL system. This chapter describes options to the **cob** command and how to use the **cob** command to compile and link source files to produce an executable module.

COBOL Interface Command

The **cob** command handles all phases involved in the production of an executable module. These phases range from checking COBOL source syntax to generating native code object modules and linking them with the COBOL and system libraries.

The result of the **cob** command can be any combination of statically linked and dynamically loaded executable files, depending on the input and options used. This allows flexibility in the development of a COBOL application.

One consideration when writing source code is the compile time involved in getting the source code into a form that can be debugged. The **cob -a** option enables fast compilation to intermediate code suitable for source level debugging on ANIMATOR. Another consideration is execution speed. Debugged code can be compiled with **cob -u** or **cob -x** to produce native code, which requires more time to compile but results in a module that executes faster.

The AIX VS COBOL system is installed in the **/usr/lpp/COBOL/lib** directory with the **installp** procedure. This directory is searched first for the various components when you issue the **cob** command, unless the **COBDIR** environment variable has been used to change the search directory. See Appendix A, "Environment Variables" for more information.

The files created by the **cob** command are placed in the current directory. Any temporary files are created in the system temporary directory, **/tmp**, unless you set and export the environment variable **TMPDIR** and specify a valid path name. See Appendix A, "Environment Variables" for more information.

The **cob** command recognizes the following file types:

File Type	Description
.cbl, .CBL, or .cob	COBOL source text file
.int	Intermediate code file
.gnt	Dynamically loaded native code file
.c	C source text file
.o	Object module file
.a	Archive file
.s	Assembler source file

You can force the **cob** command to recognize files with extensions other than those listed above by specifying the **-k** option on the command line. See "Option Specification" on page 4-7 for more information.

To invoke the **cob** command type the following on the AIX VS COBOL system:

```
cob [options] filename ←
```

where:

options is one or more of the options or flags described in "Option Specification" on page 4-7.

filename is any mixture of COBOL source, intermediate code, native code, linkable object code, C source, assembler source, or archive files. These files are recognized by their extension. Any unrecognized files are saved to be used at link time. The system assumes that they are either valid linker options or input files.

Any archive files supplied to the **cob** command are passed to the linker. The entry-point for a COBOL file is derived by taking the base name of the file without the extension. If the first character of the entry-point is numeric, it is converted as follows:

0 to J
1 through 9 to A through I

filename must not contain a hyphen. Any hyphens in *filename* are converted to zeros.

The Development Cycle

Use the **cob** command to do the following:

- Check COBOL source file syntax.
- Generate intermediate code files suitable for interpretation by **cobrun**.
- Code-generate the resulting intermediate code files into native code.
- Link native code with COBOL libraries.
- Output any mixture of statically linked or dynamically loaded executable files.

The type of file created by the **cob** command depends on the options you specify on the command line. These are described in “Option Specification” on page 4-7. By default, the **cob** command creates a dynamically loadable intermediate code file (with the extension **.int**), which is suitable for animation.

The **cob** command passes each input file through a series of steps. Each step transforms one file type into another file type. These types are characterized by the file suffixes, and each type is available depending on the options you specify on the command line.

Table 4-1 on page 4-6 shows the development cycle of an input file to the **cob** command.

Table 4-1. Development Cycle of Input File to cob Command		
Input File Type	Output File Type	Action
.cbl .CBL .cob	.int	Checked by compiler
.int	.gnt	Code generated for dynamic loading
.int	.o	Code generated for static linking
.gnt		No further action possible
.s	.o	Passed to system assembler
.c	.o	Passed to C compiler
.o [[.o],..]	a.out	Linked with RTE

File type determines the point in the development cycle at which an input file starts. The default end point (the point at which the **cob** command terminates) creates an intermediate code file suitable for animation from the input COBOL source files. To process files beyond the intermediate code stage, specify the relevant option to the **cob** command (see "Option Specification" on page 4-7).

For example, to obtain a dynamically loadable native code file instead of an intermediate code file, specify **-u** on the **cob** command line. Under the **-u** option, source files with the extension **.cbl** are compiled and then code-generated. Files with the extension **.int** are just code-generated. The development cycle for a source file named **myfile.cbl** given to the **cob** command with the **-u** flag specified is as follows:

```
myfile.cbl → myfile.int → myfile.gnt
```

To obtain a single, statically linked executable module, specify **-x** on the **cob** command line. Under the **-x** option, COBOL source files with the extension **.cbl** are compiled, code-generated, and then linked with the COBOL libraries to form a single executable module. The development cycle for a source file named **myfile.cbl** given to the **cob** command with the **-x** flag specified is as follows:

```
myfile.cbl → myfile.int → myfile.o → myfile
```

where **myfile** is an **a.out** format file.

If you are producing a dynamically loadable file, any file names with the extension **.o** supplied on the **cob** command line are linked to the dynamic loader to produce a statically linked RTE library. The RTE library takes the base name of the first object module file supplied on the command line. A dynamically loadable program can access any modules in the static RTE library, or any other valid dynamically loadable program, using the **CALL** statement.

A statically linked program can access any other statically linked program written in a language that compiles to AIX **a.out** and follows C calling conventions. A statically linked program can also access any valid dynamically loadable program.

Option Specification

The order in which options are passed to the various AIX VS COBOL tools determines their precedence, with later options overriding previous options. This is important if you plan to specify additional options to the defaults of the AIX VS COBOL system. The higher numbers override previous defaults.

The **cob** command processes options in the following order:

1. System-wide defaults, as defined in **\$COBDIR/cobopt**
2. Optional user defaults, as defined in the COBOPT environment variable
3. Command line options
4. Embedded source file options.

System-Wide Default Options

System-wide default options are defined in the file **\$COBDIR/cobopt**. This is the file the **cob** command reads first when invoked. This file has the following format:

```
compiler: [ [COBOL-COMPILER-OPTION] ... ]
ncg: [ [NCG-OPTION] ... ]
[SET environment-variable=value ... ]
[; comment-entry ... ]
```

Notes:

1. Any lines in this file which begin with a semicolon (;) are treated as comment lines by the **cob** command.
2. Since the contents of this file affect the operation of the entire AIX VS COBOL system, you should only alter the file after careful consideration of the effect you might have on the default COBOL options in your system.

See Chapter 5, “Compiler Options” and Chapter 6, “Native Code Generator Options” for complete information on the default compiler and Native Code Generator options.

Optional User Default Options

Use the COBOPT environment variable to do the following:

- Supply options which supplement or override the system-wide default options defined in **\$COBDIR/cobopt**
- Specify the path of a file which contains user options.

When using COBOPT to point to a file which contains user options, that file must have the same format as **\$COBDIR/cobopt**. If COBOPT itself contains the options, it has the following format:

```
COBOPT="compiler:[ [COBOL-COMPILER-OPTION]...]
ncg:[ [NCG-OPTION]...]
[SET environment-variable=value ... ]
[; comment-entry ... ]"
```

Note: You must include the quotation marks. There cannot be any spaces between the colons following each component name.

You can use the SET statement in COBOPT to force the **cob** command to set the specified environment variable to the given value. For example:

```
SET COBCPY=:$COBDIR/src1ib:$HOME/mylib::
```

See Chapter 5, “Compiler Options” and Chapter 6, “Native Code Generator Options” for complete information on permitted options.

Command Line Options

Options and flags specified on the command line override user default options set up in COBOPT and the system-wide default options specified in **\$COBDIR/cobopt**.

The **cob** command supports the following flags:

Flag	Description
-a	Compile for animation (default) when no other options are specified.
-c	Compile to object module (.o).
-d <i>symb</i>	Dynamically load <i>symb</i> .
-e <i>epsym</i>	Set initial entry-point to <i>epsym</i> .
-g	Create information for symbolic debugger (dbx). Code source is unaffected.
-i	Compile for unlinked environment (.int).
-k <i>ext</i>	Recognize extra source COBOL file extensions.
-l <i>key</i>	Pass -l <i>key</i> to system linker (ld) maintaining relative ordering.
+l <i>key</i>	Pass -l <i>key</i> to system linker after all other options.
-m <i>symb=newsym</i>	Map text <i>symb</i> onto <i>newsym</i> .
-o <i>filename</i>	Specify output file name.
-p	Compile and link with AIX profiling routines.
-pg	Compile and link with AIX Berkeley profiling routines.
-u	Compile for unlinked environment (.gnt).
-v	Set verbose mode.
-x	Process to statically linked executable module.
-A <i>option</i>	Pass <i>option</i> to assembler (as).
-CC <i>option</i>	Pass <i>option</i> to C compiler.
-C <i>option</i>	Pass <i>option</i> to COBOL compiler.
-D	Show each command line step involved in compilation.
-F	Create an RTE quickly.
+F <i>symb</i>	Create an RTE quickly and add <i>symb</i> to a linked data table.
-L <i>dir</i>	Pass -L <i>dir</i> option to system linker changing search algorithm.
-N <i>option</i>	Pass <i>option</i> to Native Code Generator.

-O	Turn optimization on.
-P	Produce COBOL compilation listing file.
-Q <i>option</i>	Pass <i>option</i> to system linker (ld).
-S	Do not assemble the <code>.s</code> file.
-T	Put only text symbols into the “loaded” table.
-U	Pass unresolved reference to linker (ld).
-V	Report version number.
-W <i>err-level</i>	Control error level for cob termination.
-X <i>symb</i>	Exclude text <i>symb</i> from the executable output file.

Options and flags specified on the command line override any user default options set up in COBOPT and the system-wide default options defined in **\$COBDIR/cobopt**. The following subsections describe each flag.

Compile for Animation (-a)

-a compiles the source file input to the **cob** command ready for animation. This is the default end point of the **cob** command. The **cob** command outputs intermediate code files (with the suffix `.int`) and ANIMATOR files (with the suffix `.idy`). Both of these are used by ANIMATOR when you debug your code. See Chapter 11, “Debugging Your Program Using ANIMATOR” for details on how to use ANIMATOR. If you supply any `.o` files to the **cob** command, they are linked with the COBOL libraries to form a single executable file. The executable file is the one you need to use when you run ANIMATOR.

In this way it is possible to animate programs that call, or are called by, programs written in languages other than COBOL.

For example:

```
cob -a myfile.cbl c.o <|
```

creates the files `myfile.int`, `myfile.idy`, and `c`. The file `c` contains the RTE and the file `c.o`. The command:

```
cob myfile.cbl c.o <|
```

has the same effect. You do not need to specify **-a** because by default the **cob** command processes each input file as though this flag had been set.

If you then want to animate `myfile.int`, use the commands:

```
COBSW=+A
export COBSW
c myfile.int
```

Compile to Statically Linkable Object Module (-c)

-c compiles source text files and code generates them no further than `.o` modules. If you supply intermediate files instead of source files, these files are just code-generated to statically linkable `.o` modules. This flag has an effect only if specified with the **-x** or **-u** options.

Dynamically Load *symb* (-d *symb*)

-d *symb* causes *symb* to be dynamically loaded if it is referenced. This option allows certain parts of the RTE to be loaded as necessary rather than to be loaded permanently. The ADIS module is by default dynamically loaded with your program.

Set Initial Entry-Point (-e *epsym*)

By default, the entry-point address for a statically linked module is the base name of the first file input to the **cob** command. This option allows you to override the default and set the default entry-point address to be that of the symbol *epsym*. For this option to take effect *epsym* must be defined in a COBOL module. *epsym* can also be null, in which case the entry-point address is read from the command line at run time. If you wish the entry-point to be null, use the command:

```
cob -xe "" -o rts
```

Create Information for Symbolic Debugger (-g)

The compiler creates additional information needed for the use of the symbolic debugger **dbx**. This debugger is used to debug native code that has been statically bound. The **dbx** debugger can be used for C code or for COBOL code. See the documentation on **dbx** on how to use its features.

When using **dbx** to debug code, you can, for example, do the following actions:

- Set breakpoints
- See call tracebacks
- Step through source code lines or native instructions
- See the declarations of variables
- See the value of variables.

When debugging C code, the full features of **dbx** can be used. When debugging COBOL code, all of the features are not implemented. For example, you cannot evaluate COBOL expressions under **dbx**.

You can mix the use of **dbx** with the use of the animator. That is, you can debug code that is statically bound at the same time as you debug **.int** code using the animator. See the example in "Mixing C and COBOL Programs" on page 2-18.

When the **-g** flag is given, a lookahead optimization feature in the native code generator is suppressed. This lookahead suppression is needed to make the source line numbers used by the debugger correspond correctly to the line numbers that the generated native code references. This effect, combined with the additional code needed to reference symbolic debug information, will result in reduced performance for code compiled with **-g**. This is typical behavior for code compiled for debugging.

Compile for Unlinked Environment (-i)

This compiles the source files input to the **cob** command into dynamically loadable intermediate code files.

Recognize Extra COBOL Source File Extensions (-k *ext*)

The **cob** command recognizes file types which have the following extensions: **.cbl**, **.CBL**, **.cob**, **.int**, **.gnt**, **.c**, **.a**, **.s** and **.o**. You can submit COBOL source files with other extensions to the **cob** command provided you specify **-k** on the command line before each of the filenames which has the non-conventional extension.

Pass -l key to System Linker Maintaining Relative Ordering (-l key)

-l *key* is passed to the system linker (**ld**) maintaining the relative ordering. The system linker searches the library **libkey.a** for any external routines. The system linker searches a library when its name is encountered, so where you place -l is significant. By default, libraries are searched for in **\$COBDIR/coblib** and then in the **/lib** and **/usr/lib** directories. See “Pass -L *dir* to System Linker Changing Search Algorithms (-L *dir*)” on page 4-13 for details on how you can specify alternative search paths.

Pass -l key to System Linker after All Other Options (+l key)

-l *key* is passed to the system linker after all other linker options and the COBOL libraries have been passed to it. The system linker searches the library **libkey.a** for any external routines. By default, libraries are searched for in **\$COBDIR/coblib** and then in the **/lib** and **/usr/lib** directories. See “Pass -L *dir* to System Linker Changing Search Algorithms (-L *dir*)” on page 4-13 for details on how you can specify alternative search paths.

Map *syml* to *newsym* (-m *syml* = *newsym*)

-m maps unresolved symbol *syml* to *newsym*. This creates a routine to satisfy any references to *syml*. If this routine is called, control passes to the routine which has the entry name *newsym*. *newsym* must be defined. You could use this flag to dummy out unwritten optimized routines into one general purpose routine, provided the calling sequence is the same. For example,

```
cob -x -m s1=x1 -m s2=x2 myprog.cbl mylib.a ◀
```

maps the unwritten routines *s1()* and *s2()* to the functionally similar *x1()* and *x2()* routines, which must already have been coded. You can also use this option to substitute your own file handler for indexed file operations, in place of the one supplied with your AIX VS COBOL system. You can do this only if your file handler conforms to the Callable File Handler Interface standard. See Chapter 9, “Advanced Programming Features” for details.

Specify Output filename (-o filename)

By default, the name of the final executable module created by the **cob** command, if the -x option is specified, is the base name without the suffix of the first file entered to the **cob** command. This option allows you to change the name of this module.

Compile and Link with AIX Profiling Routines (-p)

-p prepares the AIX VS COBOL program so that the **prof** command can generate an execution profile. The -p causes the compiler to produce code that counts the number of times each procedure is called. The -p is also passed to the C compiler if any C source files are specified on the **cob** command line. If you are using **cob** to output a statically linked executable module, this option causes **cob** to include the system library **libc_p.a** instead of **libc.a**. It also binds in the startup module **mcrt0.o** instead of **crt0.o**. See the AIX system documentation for more details.

Compile and Link with AIX Berkeley Profiling Routines (-pg)

-pg is similar to the -p option, but the -pg uses the AIX Berkeley Profiling Routines. It invokes a run time recorder that keeps extensive statistics on the running process. The -pg is also passed to the C compiler if any C source files are specified on the **cob** command line. If you are using **cob** to output a statically linked executable module, this option causes **cob** to include the system library **libc_p.a** instead of **libc.a**. It also binds in the startup module **gcrt0.o** instead of **crt0.o**. See the AIX system documentation for more details.

Compile for Unlinked Environment (-u)

-u compiles source text files and code generates them to dynamically loadable native code. The intermediate code file will be created in the current directory. You can supply intermediate code files instead of source text files; these are just code-generated. If you supply any **.o** files as input files to the **cob** command, they are statically linked to the dynamic loader to produce the static RTE library for dynamically loadable files. Dynamically loaded programs can access (via **CALL**) any of the modules in the statically linked RTE library and also any other valid dynamically loadable program.

Verbose Module (-v)

-v sends the verbose option to the compiler and the native code generator.

Process to Statically Linked Executable Module (-x)

-x creates a single statically linked executable module from the files input to the **cob** command. By default, the name of this module is the base name without the extension of the first file input to the **cob** command. You can use this option to produce a full RTE. For example,

```
cob -xo rts.new ◀l
```

Warning: Do not try to use any other method to create a full RTE.

Pass option to Assembler (-A option)

-A option passes the specified option to the assembler.

Pass option to the COBOL Compiler (-C option)

-C option passes the specified option to the COBOL compiler. Chapter 5, "Compiler Options" contains full details on the options you may use with this flag.

Pass option to the C Compiler (-CC option)

-CC option passes the specified option to the C compiler. See the AIX commands documentation for valid **-CC** options.

Show Each Command Line Step Involved in Compilation (-D)

The **-D** flag will show a detailed expansion of each step that is taken for the compilation processing. This flag only has effect when it is used with the **-v** flag. It will show the exact invocation and the full path names of files and all arguments given to each of the commands to do COBOL and C compilations, assembly, and binding.

If you give the flag **-DDD**, the temporary work files that are created as part of compilation processing will not be erased. The **cob** command will issue a message when it is finished telling you what the name of the work directory is under **/tmp**. The **-DDD** flag can be used with or without the **-v** flag. If the **-v** flag is not used, the only information shown by the **cob** command is the name of the temporary work directory.

Create an RTE Quickly (-F)

Allows the fast creation of a statically linked executable module with dynamic load support. When a statically linked executable module is being produced, all the entry-points and external data (from both COBOL and C modules) are made available to dynamically loaded programs. This is achieved by **cob** creating an entry for each entry-point and all external data in a “loaded table” – *ldtab*. If you specify the **-F** flag, **cob** only creates entries for modules named on the command line plus a default list found in `$COBDIR/coblib/cobfsym`. Any symbols for objects in archive files are not included in *ldtab* if you specify the **-F** option, which reduces the time taken to create a statically linked executable module. If a reference from a dynamically loaded module is made to any name which is not in *ldtab*, then you will receive RTE error 173:

Called program file not found in drive/directory

See “Create a RTE Quickly and Add *symb* to a Linked Data Table (+F)” for details of how to add symbols to *ldtab*.

You cannot specify this option if you set the **-U** option.

Create a RTE Quickly and Add *symb* to a Linked Data Table (+F)

Specifying the **+F** option has exactly the same effect as specifying the **-F** option as described in “Create an RTE Quickly (-F),” with the exception that the **+F** option allows you to specify a symbol which you wish to add to the linked data table, *ldtab*.

You cannot specify this option if you set the **-U** option.

Pass **-L dir** to System Linker Changing Search Algorithms (-L dir)

-L dir is passed to the system linker maintaining the relative ordering. This option changes the search algorithm for libraries which do not have an absolute path name. By default, **cob** searches the `$COBDIR/coblib` directory first and then the `/lib` and `/usr/lib` directories next, but if you specify this option, **cob** searches the specified directory first instead.

Pass *option* to NCG (-N option)

-N option passes the specified option to the Native Code Generator. Chapter 6, “Native Code Generator Options” contains full details on the options you may use with this flag.

Turn Optimization On (-O)

-O enables maximum performance at run time because minimum run-time checks are carried out. It is recommended that use of this flag be limited to debugged code. At a minimum this flag passes the **nobound** option to the compiler and Native Code Generator.

Produce Listing File (-P)

-P causes the compiler to produce a listing file (with the extension `.lst`) for each COBOL source file.

Pass option to System Linker (-Q option)

-Q option passes the specified option to the system linker. When you use the **-Q** flag on the **cob** command line to pass options to the system linker, you must use a separate **-Q** flag for each option. Options that begin with hyphens or have embedded spaces in them must be enclosed in quotation marks.

Do Not Assemble the .s File (-S)

This option suppresses the assembly of the **.s** file if the **asm** NCG option is given. It will also suppress the binding step if the **obj** NCG option is given. The combinations of relevant options and their effects are:

asm noobj -S Produces a **.s** file but does not assemble it. No **.o** is produced.

asm obj -S Produces a **.s** and an **.o** file. No assembly or binding is done.

noasm obj -S Produces a **.o** file but does not bind it. No **.s** file is produced.

Put Only Text Symbols into the "Loaded Table" - ldtab (-T)

-T causes the **cob** command to put only entry-point symbols into the "loaded table"
- *ldtab*. External data items are not put into the loaded table if the **-T** option is set.

Unresolved Reference (-U)

Any unresolved reference found at link time causes the code to call the dynamic loader to be included with the name of the unresolved symbol. This allows the **cob** command to attempt to load and execute any valid dynamically loadable file of that name that may exist.

Report Version Number (-V)

-V reports the version number of any of the invoked components. This implies that you have also set the **-v** (verbose) option.

Control Error Level for cob Termination (-W err-level)

-W err-level specifies the level of COBOL compiler error which causes the **cob** command to stop processing. *err-level* is a single alphabetic character representing the following possible levels of error:

u	Unrecoverable
s	Severe
e	Error
w	Warning
i	Informational

The **cob** command terminates if your code contains an error at the specified level or higher, provided such errors are reported to the **cob** command by the compiler. This is dependent upon the setting of the **warning** compiler option, which controls the level of error reported by the compiler. For example, if you set the **warning** option to force the compiler to report only unrecoverable, severe, and error level errors, and you set **-W** to abort the **cob** command should any errors at the information level (or above) be reported, only errors in the categories unrecoverable, severe, or error will actually cause the **cob** command to terminate.

By default, the **cob** command terminates if your code contains reported errors in the severe category or above.

Exclude *symbol* from the Executable Output File (-X *symbol*)

-X *symbol* excludes the unresolved text symbol *symbol* from the executable output file. It can be used to satisfy undefined symbols to modules which are not required. This will allow an executable file to be produced. This option can also be used to exclude from the RTE those parts of it which the program does not need. Those parts and the symbols to represent them are:

Symbol	Description
ANIM	ANIMATOR
DYNLOAD	Dynamic loader
INTERPRETER	COBOL interpreter for .int code
PROFILE	COBOL profiler
RTECALL	RTE call by number routines
USERCALL	User call support
syms	Run Time support for dynamically loaded .gnt files

If you attempt to call any undefined symbol, the following RTE error message appears:

```
164 Run Time subprogram not found
```

When you use the -X flag to exclude certain modules from the executable output file, the resulting file is not actually any smaller than it would have been if you had not specified the flag. Setting this flag does ensure that you will receive a meaningful error message if you attempt to call any excluded module.

Embedded Source File Options

One or more compiler options can be specified within your COBOL source code. The compiler will use these options each time you compile that code. In order to embed options within the source code, use the \$SET statement. This has the form:

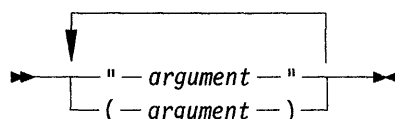
```
$SET [COBOL-COMPILER-OPTION]...
```

where:

COBOL-COMPILER-OPTION can be one or more of the compiler options specified in Chapter 5, "Compiler Options." It cannot contain a native code generator option. Each item in the option list must be separated by spaces. An option list can be no longer than one line. To specify additional options that exceed the space available on one line, use another \$SET statement, as follows:

```
$SET noalter bound nocomp list  
$SET errlist
```

To modify a compiler option in a \$SET statement by an argument, enclose the argument within either double quotation marks or parentheses:



Note that you cannot precede an argument by an equal sign in a \$SET statement, as you can when you specify argument with a compiler option on the **cob** command line or in the **\$COBOPT** file.

argument can contain spaces if enclosed in quotation marks, but not if enclosed in parentheses.

Both of the following examples have the same effect: They cause the compiler to assume all file assignments to data-names will be resolved externally, and to flag features in your program which are not in the ANS85 dialect of the COBOL language.

- \$SET assign(external) flag(ans85)
- \$SET assign "external" flag "ans85"

\$ must be in column 7; if it is not, the compiler will not recognize it and will produce errors. Failure to place the \$ character in column 7 may also "hang" the compiler. The same is true for any character at the start of a source file which the compiler does not recognize.

You can specify multiple \$SET statements within your source code, and these can appear anywhere in the code. However, if you want to specify any dialect-controlling compiler options, for example **ans85**, these must appear as the first line of your source code, as shown below:

```
$ SET ANS85.  
IDENTIFICATION DIVISION.  
.  
.  
.
```

Once you have set a dialect-controlling option at the beginning of your source code, you cannot unset it later in the program. Refer to "Options Permitted in \$SET Statements" on page 5-28 to know which options are permitted with the \$SET statement and where in the source file they are permitted.

Options specified within COBOL source files by means of the \$SET command have the highest precedence of all the options specified to the compiler. They override those specified on the **cob** command, with the environment variable **COBOPT**, and the system default options as defined in **\$COBDIR/cobopt**.

Command Line Conventions

You should observe the following rules while using the **cob** command:

- All flags must be delimited by the hyphen (-).
- Flags which have no arguments can be grouped behind one delimiter. For example:

```
cob -Pa pi.cbl ◀
```

compiles the COBOL source contained in **pi.cbl** into a file that is suitable for animation and produces a listing file, **pi.lst**.

- The first argument following a flag must be preceded by at least one space.

-
- Groups of arguments following a flag must be separated by at least one space, and they must be enclosed in quotation marks. For example:

```
cob -C "list noalter" pi.cbl ◀
```

has the same effect as:

```
cob -C list -C noalter pi.cbl ◀
```

Both pass the list and noalter options to the compiler.

- All flags must precede operands.
- You may use two hyphens (--) to delimit the end of the flags.

Command Line Examples

The following examples demonstrate how to use the **cob** command:

- `cob -a pi.cbl ◀`

This is the default case. It compiles the program in **pi.cbl** into a file called **pi.int**, which is suitable for animation.

- `cob -x pi.cbl ◀`

The COBOL source file **pi.cbl** is compiled, code-generated, and then linked to the RTE to form a statically linked **a.out** format file named **pi**.

- `cob -i pi.cbl ◀`

The COBOL source file **pi.cbl** is compiled for the unlinked environment to produce an intermediate code file, **pi.int**.

- `cob -u pi.cbl ◀`

This command compiles and code generates the program in **pi.cbl** into a dynamically loadable native code file called **pi.gnt**.

- `cob -x -e "" pi.cbl ◀`

This command compiles, code generates and links the program contained in **pi.cbl** to form a statically linked executable file named **pi**. Specifying the **-e** option with a null argument ensures that the entry-point is read from the command line at run time.

See Chapter 7, "Running an AIX VS COBOL Program" for details of the commands you can use to execute the files produced by the above examples of the **cob** command.

Chapter 5. Compiler Options

Contents

About This Chapter	5-3
Format of Compiler Options	5-4
Permitted Options	5-5
Excluded Combinations	5-23
ANS85 Options	5-24
Default Options	5-24
Mainframe Options	5-27
SAA Options	5-27
Options Permitted in \$SET Statements	5-28
Compiler Messages	5-29
Listing Format	5-30

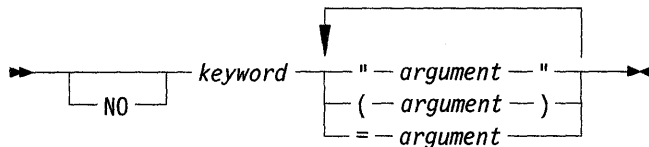
About This Chapter

This chapter details the system-wide default compiler options as defined in **\$COBDIR/cobopt**. You can supplement or override these default options by setting up a user-defined option variable: **\$COBOPT**. Chapter 4, “The COBOL Interface,” contains full details on the format of these files. You can also override the default options by using the **-C** command line flag or a **\$SET** statement. See Chapter 4 for details.

Throughout this chapter, all references to **\$COBOPT** refer to the contents of the **COBOPT** environment variable, or the file to which it points.

Format of Compiler Options

Compiler options, whether they appear in the system default **\$COBLIB/cobopt** file, the user-defined options file **\$COBOPT**, or following the **-C** flag in the **cob** command line, have the general form:



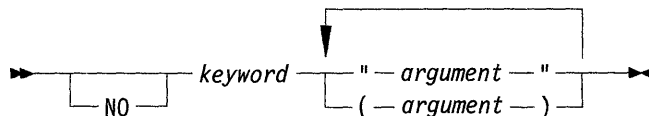
where:

keyword is one of the keywords described in “Permitted Options” on page 5-5.

no, if specified, switches off the effect of the option, and may adjoin *keyword* or be separated from it by one or more spaces. A particular option may be on or off by default.

argument, where applicable, qualifies the action of the option in some way and may adjoin *keyword* or be separated from it by one or more spaces. *argument* must be preceded by an equal sign, or it must be enclosed within either double quotation marks or parentheses.

Compiler options, where they appear in \$SET statements, have the general form:



where **no**, *keyword*, and *argument* are as described above, with the exception that *argument* must be enclosed within either double quotation marks or parentheses; it cannot be preceded by an equal sign.

Wherever possible, you should use the format of the option that contains an equal sign before any modifying argument. This is because if you use either of the other possible formats, you must escape the quotation marks or the parentheses whenever they might be misinterpreted by the AIX shell. Compiler options whose argument contains an embedded blank must use the *keyword* “*argument*” format to pass the option to the compiler.

For example, we recommend you use either:

```
cob -C assign=external -C flag=ans85 pi.cbl ␣
```

or

```
cob -C "assign=external flag=ans85" pi.cbl ␣
```

Both of these examples cause the compiler to assume that all file assignments to data-names will be resolved externally, and to flag features in your program which are not in the ANSI 85 dialect of the COBOL language. The quotation marks in the second example are necessary as they inform AIX that all the material within them is grouped behind the **-C** flag. If you omit the quotation marks, second and subsequent options are ignored. A warning is given to this effect.

Permitted Options

Options must be separated by one or more spaces. Options default to values that yield the highest performance when appropriate. The following are the permitted options:

[no] align

align [= *integer*]

Specifies the byte boundaries on which data items are aligned. 01 and 77 items are aligned on addresses which are multiples of *integer*.

Default: **align** = 4

[no] alter

Controls the use of **alter** statements within the program being compiled. **no alter** allows the compiler to operate more efficiently.

Default: **alter**

[no] analyze

Is reserved for use with other AIX VS COBOL products that can be added to the AIX VS COBOL system.

Default: **no analyze**

[no] anim

Causes the program to be compiled in a manner suitable for animation. See Chapter 11, "Debugging Your Program Using ANIMATOR" for more details.

Default: **no anim**

[no] ans85

ans85 = **syntax**

Specifies that those reserved words that are specific to the ANSI 85 COBOL standard (other than those in ANSI 74 COBOL) should be regarded as reserved words. This also alters the behavior of certain statements to conform to the ANSI 85 COBOL standard. See the *Language Reference* for details. To ensure full compatibility with the ANSI 85 COBOL standard, you must also set the **no optional-file** option.

When used with the optional *syntax* parameter, this option enables ANSI 85 syntax. However, it retains ANSI 74 behavior for those elements of the COBOL language that occur in both the ANSI 85 and ANSI 74 standards with different behavior. These features are:

- Definition of I-O status values
- Behavior of the **alphabetic** class test
- Order in which control variables in **performs** are initialized
- Behavior of **moves** to a variable-length group item.

Default: **ans85**

assign = { **external** }
 { **dynamic** }

Specifies the default value for an ASSIGN clause that does not specify either **external** or **dynamic**.

Default: **assign = dynamic**

[no] autolock

Causes the default locking for files opened I-O or EXTEND to be **automatic** rather than **exclusive**. This option does not appear in the list produced by the **setting** option if its state is the same as the **writelock** option. If this is the case, the state of the **fileshare** option in the **setting** list also indicates the state of the **autolock** and **writelock** options.

Default: **no autolock**

[no] bell

bell [= *integer*]

Defines the character used to cause the bell (the audible warning of the terminal) to sound. *integer* is the ASCII character in decimal.

Turning the option off (**no bell** or **bell = 0**) causes no bell character to be set.

Specifying **bell** with no *integer* indicates that the character to be used is the character specified in the terminfo entry for your terminal type. See the AIX operating system documentation for more details.

Default: **no bell**

[no] bound

Specifies that on each table access during execution of intermediate code the subscript value is checked to ensure that it is within the limits implied by the associated OCCURS clause. **no bound** turns off this run-time checking.

For a constant subscript that is out of bounds, a message will always be issued at compile time. If the **bound** option is given, this message will be an Error; if **no bound** is given, then this compile-time message will be a Warning level message.

This option **bound** is synonymous with the option **check**.

Default: **no bound**

[no] brief

Produces error numbers only on the listing and **stderr**. The text of error messages is suppressed.

Default: **no brief** (unless no error message file can be found)

charset = *character-set*

Defines the *character-set* of your environment. All literals and collating sequences will be handled in the specified *character-set*, which must be either ASCII or EBCDIC.

Default: **charset = ASCII**

[no] check

Specifies that on each table access during execution of intermediate code the subscript value is checked to ensure that it is within the limits implied by the associated OCCURS clause. **no check** turns off this run-time checking.

For a constant subscript that is out of bounds, a message will always be issued at compile time. If the **check** option is given, this message will be an Error; if **nocheck** is given, then this compile-time message will be a Warning level message.

This option **check** is synonymous with the option **bound**.

Default: **no check**

[no] comp

Is supported for compatibility purposes only. Unsigned integer USAGE COMP items are compiled as COMP-X items, and signed integer USAGE COMP items are compiled as DECIMAL items.

Default: **no comp**

[no] coms85

Alters the behavior of communications syntax to be as specified in the ANSI 85 COBOL standard. See the *Language Reference* for details.

The syntax for the Communications module can be compiled; however, the Communications module is not supported at run time.

Default: **coms85**

[no] confirm

The compiler options specified after this option are echoed to the display screen.

Default: **confirm**, but this is visible only if you specified verbose (the -v flag) to the **cob** command.

[no] copylbr

This is reserved for use by the AIX VS COBOL system.

Default: **no copylbr**

[no] copylist [= integer]

Causes the contents of any files named in COPY statements to be listed. Whatever the state of this option, the name of any copy file open at the time a page heading is output is listed as part of the heading.

The optional *integer*, which must be 0 or in the range 50 to 99 inclusive, allows the selection of particular segments with this option. A 0 means all root segments. For example:

copylist = 53 causes **copylist** to be set only in the Identification Division and in segment 53.

no copylist = 53 causes **copylist** to be set in segment 53 only.

Default: **no copylist**

currency-sign = *integer*

Kanji feature. Causes the compiler to recognize *integer* as the currency-sign character. *integer* must be a 2-digit decimal number specifying the ASCII value of the currency sign required. Values not allowed in the CURRENCY-SIGN clause in the Special-Names paragraph are also not allowed as values for *integer*. See the *Language Reference* for a list of these values. You can override the currency sign specified by this option by specifying a CURRENCY-SIGN clause within the Special-Names paragraph of your source code. If you set both the **currency-sign** and the **nls** options when compiling your program, the currency sign specified in the CURRENCY-SIGN compiler option is overridden by that supplied by AIX (from an environment variable) during the execution of your program.

Default: **currency-sign** = 36 (the \$ character)

[no] **date**

date [= *string*]

date causes the system date to be entered into the comment entry in the DATE-COMPILED paragraph (if present). To provide your own date, specify **date** = *string*. The **date** option also causes the system date or *string* to be output at the top of each page of the listing.

no date causes spaces to be used in place of date.

Default: **date**

[no] **dbcs**

Kanji feature. Causes the compiler to accept characters of the Double Byte Character Set (DBCS) for use in ideographic languages such as Japanese, Chinese, and Korean.

For this option to have effect, you must have installed the DBCS variety of the AIX VS COBOL compiler. See Chapter 1, "Introduction" for more information on installing the AIX VS COBOL system.

For the **dbcs** option to have complete effect (all the needed additional reserved words, for example), you must also use the **vsc2** option.

The option **dbcs** cannot be used together with the **nls** option.

Default in the DBCS-variety of the compiler: **dbcs**

Default in the non-DBCS-variety of the compiler: **nodbcs**

nodbcssosi

dbcssosi(*integer*)(*integer*)

Kanji feature. Defines the 2 characters used as the shift-out and shift-in delimiters in DBCS literals. The 2 integers are the ASCII codes, given in decimal, of the characters. If shift-out and shift-in characters are specified, each DBCS literal must have the shift-out character immediately after the opening quotation mark and the shift-in character immediately before the closing quotation mark. They act as additional delimiters to the literal, and are not part of its value. If **nodbcssosi** is specified, then no shift-out and shift-in characters are needed or recognized in the DBCS literals.

If **dbcssosi** is specified, then the two integer parameters are required.

Default in the non-DBCS variety of the compiler: **nodbcssosi**

Default in the DBCS variety of the compiler: **nodbcssosi**

[no] defaultbyte

defaultbyte [= *integer*]

Initializes the Data Division to the specified byte; by default this is to spaces.

Default: **defaultbyte** = 32

[no] dg

Specifies that those reserved words and features specific to the Data General Interactive COBOL language are enabled. See the *Language Reference* for details.

Default: **no dg**

[no] directives

directives = { *filename* }
{ *filename* }

Enables you to load preset options from the file named in *filename*. If you specify this option on the **cob** command line, it must be the last option on the line. If you specify it in a \$SET statement, it must be at the beginning of your source file. See Chapter 4, “The COBOL Interface” for details. Options within the file specified by *filename* must be separated by a space, and no option can be broken across two lines. The options are read from the file until the end-of-file (EOF) is reached or another **directives** option is specified. You can specify more than one options file in a program by specifying **directives** = *filename* within an options file or by writing more than one \$SET statement at the beginning of your program. If you specify **directives** within an options file, the compiler switches to the new options file, but does not return to the original options file. The setting of the **directives** option does not appear in the list produced by the **setting** option.

Default: **no directives**

[no] echo

Causes error lines and flags to be echoed to **stderr**. Each error message shows the source line of the error, the error number, and (unless **brief** is set) an explanatory message.

Default: **echo**

[no] echoall

Ensures that a full listing is sent to **stdout**, if the list or print option is specified.

Default: **no echoall**

[no] errlist

Causes the listing to be restricted to those COBOL lines containing syntax errors or flags, together with associated error messages.

Default: **no errlist**

[no] errq

Asks whether you want compilation to stop or continue when an error occurs.

Default: **no errq**

[no] filecase

Specifies whether the compiler is to be case-sensitive. **no filecase** means that the compiler is sensitive to case. It is recommended that you do not alter the default setting of this option since it has been set up for your AIX environment.

Default: **no filecase**

[no] fileshare

Has the same effect as specifying both the **autolock** and **writelock** options. It is provided for compatibility with earlier FILESHARE products. It is recommended that you do not use it in new programs.

Default: **no fileshare**

[no] flag

flag = $\left. \begin{array}{l} \mathbf{ans74} \\ \mathbf{ans85} \\ \mathbf{mf} \\ \mathbf{osvs} \\ \mathbf{saa} \\ \mathbf{vsc2} \end{array} \right\}$

Causes the compiler to flag every feature used in the program that is outside a given dialect. These dialects are:

ans74 Full implementation of ANSI standard X3.23 - 1974 COBOL.

ans85 Full implementation of ANSI standard X3.23 - 1985 COBOL.

mf Micro Focus COBOL extensions.

osvs As for **ans74**, plus features taken from IBM OS/VS COBOL syntax. (See the *Language Reference* for details.)

saa Full implementation of IBM System Application Architecture definition of COBOL.

vsc2 As for **ans74**, plus features taken from IBM VS COBOL II syntax. (See the *Language Reference* for details.)

The message output by the compiler lists the dialect in which the feature you have used would be acceptable.

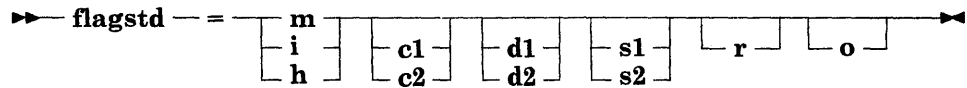
Default: **no flag**

[no] flagq

Specifies whether you want the compiler to terminate its run should it output a flag.

Default: **no flagq**

[no] flagstd



Causes ANSI 85 language level certification flags to be output as the source program is checked. Flags are issued to the selected COBOL subset, optional modules, and obsolete elements arguments of the **flagstd** option.

If the **flagstd** is set, it must include at least one of the following 3 arguments:

- m** ANSI 85 defined Minimum COBOL subset
- i** ANSI 85 defined Intermediate COBOL subset
- h** ANSI 85 defined High COBOL subset

Optionally, you may narrow the scope of flagging by adding any of the following arguments. These arguments may be in any order but must be separated by at least one space.

- c1** Communications optional module level 1
- c2** Communications optional module level 2
- d1** Debug optional module level 1
- d2** Debug optional module level 2
- s1** Segmentation optional module level 1
- s2** Segmentation optional module level 2
- r** Report Writer optional module
- o** All Obsolete language elements

Note that the **flag** and **flagstd** options provide similar functionality and thus only one may be used at any time.

Default: **noflagstd**

[no] form

form = *integer*

Specifies the number of lines per page of the listing. *integer* must be at least 3 and not greater than 255.

A form feed character is always produced at the head of the listing file, unless **no form** is used. **no form** specifies that no form feed characters or page headings are produced anywhere in the listing.

If the listing is directed to **stdout** (by use of the **list** option), interpretation of the form feed character is dependent on the type of your display screen.

Default: **form** = 60

[no] ibmcomp

Causes data items with USAGE COMP to be compiled in IBM synchronized (SYNC) format. This improves performance at the cost of increasing the required memory by a small amount. (See the *Language Reference* for details.)

Default: **no ibmcomp**

**[no] { ibm-ms
pc1 }**

Specifies that those reserved words and features that are specific to the IBM-Microsoft COBOL Language are enabled under AIX VS COBOL. See the *Language Reference* for details. Note that **ibm-ms** and **pc1** are synonymous.

The **ibm-ms** option provides compatibility with Microsoft COBOL 1.0. See also the option **ms(2)** for compatibility with Microsoft COBOL 2.2.

Default: **no ibm-ms**

[no] int

int [=file-name]

Specifies the file to be used to hold the intermediate code output by the compiler. If the specified file already exists, it is overwritten. This filename may include a path.

no int suppresses the production of an intermediate code file (the compiler is used for syntax checking only).

Note that if *file-name* is specified without a “.” extension, then an extension must be added later, or the Run Time Environment (RTE) will be unable to find the file and will respond with an error. The RTE assumes that an intermediate code file has the extension **.int**.

You cannot specify this option in a \$SET statement.

Default: the compiler adds **.int** to the source file name, replacing any existing file name extension. The intermediate code file is written to the same directory as the source file.

linkcount = integer

Specifies the maximum number of LINKAGE SECTION items, external data items, and external files allowed in the compilation of nested programs.

integer must not be less than the total number of LINKAGE SECTION items, external data items, and external files in the compilation after the end of the first LINKAGE SECTION, external data item, or external file, whichever appears first.

Default: **linkcount = 64**

[no] { list
 print }

{ list
 print } [= destination]

Specifies the destination of the listing file. If an existing file is specified it is overwritten. You can omit *destination*, in which case the listing is directed to **stdout**. If *destination* is **stdout**, the listing is directed to **stdout**. Note that **list** and **print** are synonymous.

Provided that the listing facility is turned on (by specifying either the **-P** or the **list** option on the **cob** command line, or by specifying the **list** option in \$COBOPT) you can repeatedly use the \$SET statement to set this facility on or off for specified portions of your source code. Include a \$SET NOLIST statement at the start of the portion of source code for which you do not wish a listing to be produced. A \$SET LIST statement later in your source code will turn the listing facility back on. The resulting list file will not contain a listing of the code contained within the two \$SET statements.

Default: **no list**

{ listwidth
 lw } = integer

Sets the width of the listing. *integer* is the number of character positions across the listing page; the value must be in the range 72 to 132.

Default: **listwidth = 80**

[no] **mfcomment**

Specifies whether those lines within COBOL source programs which contain an asterisk (*) in column 1, or a form feed character in columns 1 and 2 followed by an asterisk, are processed by the compiler and appear in the list file. If you set **mfcomment**, such lines are ignored by the compiler and will not appear within the list file. If you set **nomfcomment**, these lines are processed by the compiler and appear within the list file. If ANIMATOR is invoked for programs input to the **cobol** process with **nomfcomment** specified, the results will be unpredictable.

Default: **mfcomment**

[no] mf [level]

mf [level] = *integer*

Specifies that those reserved words that are specific to Micro Focus extensions to the ANSI 74 COBOL standard should be regarded as reserved words. *integer* is used to specify which version of Micro Focus COBOL is to be treated in this way, as follows:

- 1 Professional COBOL
- 2 As 1 plus additional features in VS COBOL Workbench, Version 1.2
- 3 As 2 plus additional features in VS COBOL Workbench, Version 1.3; VS COBOL Workbench, Version 2.0; Professional COBOL Version 2.0; and VS COBOL Version 1.5
- 4 As 3 plus additional features in Micro Focus COBOL/2 and Professional COBOL/2

Default: **mf [level]** = 4

[no] ms(2)

Specifies that those reserved words and features that are specific to the IBM-Microsoft COBOL language are enabled under AIX VS COBOL. See the *Language Reference* for details.

The **ms(2)** option provides compatibility with Microsoft COBOL 2.2. See also the option **ibm-ms** for compatibility with Microsoft COBOL 1.0.

Default: **no ms(2)**

native [= *collating-sequence*]

Specifies the default collating sequence to be used for comparisons. *collating-sequence* must be either ASCII or EBCDIC.

Default: **native** = ASCII

[no] nestcall

Indicates whether the source program includes nested programs.

Default: **nestcall**

[no] nls

Specifies that special National Language Support (NLS) operations are to be done for the following:

- Explicit string comparisons, class condition tests, and numeric editing
- Key comparisons performed on indexed sequential files
- Comparisons performed as part of SORT or MERGE operations.

Selecting **nonls** causes normal operations for these comparisons.

See Appendix B, "National Language Support" for more details on using the NLS facility.

The option **nls** cannot be used together with the **dbcs** option.

Default: **nonls**

[no] odoslide

Determines the location of the second of two data items in the same group, where the first has an OCCURS DEPENDING ON clause and the second follows the first but is not subordinate to it. When you set **odoslide**, the second item is located immediately after the current size of the OCCURS DEPENDING ON table. If you set **no odoslide**, the second item is located after the maximum size of the OCCURS DEPENDING ON table.

Default: **no odoslide**

[no] oldcopy

Causes the COPY statement to operate according to the ANSI 68 COBOL standard (see the *Language Reference* for details).

Default: **no oldcopy**

[no] oldfileio

This is for use by the AIX VS COBOL system.

Default: **no oldfileio**

[no] oldindex

Causes indexes to be generated as subscripts. This option allows IBM OS/VS COBOL comparisons of index data items with arithmetic expressions. It is turned off by default because it may reduce the performance of code.

Default: **no oldindex**

[no] oldvsc2

When used with the **vsc2** option, this alters certain features from the ANSI 85 COBOL standard to make them compatible with Issue 1 of IBM VS COBOL II. This means that:

- No explicit scope delimiter is allowed in a statement without a conditional phrase (for example, AT END, ON SIZE ERROR).
- The word ALSO in an EVALUATE statement can be omitted.
- The CLASS clause in the SPECIAL-NAMES paragraph is not allowed.
- Conditional phrases with NOT (for example, NOT AT END, NOT ON SIZE ERROR) are not allowed.

Default: **no oldvsc2**

[no] optional-file

Causes the compiler to treat all SELECT statements in files opened for I-O or EXTEND as if they were OPTIONAL. Under ANSI 85 standard COBOL, SELECT statements are treated by default as NOT OPTIONAL. To ensure complete compatibility with the ANSI 85 standard you must thus specify both the **ans85** and the **no optional-file** options.

Default: **no optional-file**

[no] **osex**t

osext = *ext*

Causes the compiler to search by default for a file name with the specified extension. This will only affect the search for COPY files. *ext* can be up to 3 characters long. Use this option with caution.

Default: **osex**t = cbl

[no] **os**vs

Specifies whether those reserved words that are specific to OS/VS COBOL language extensions should be treated as reserved words. This also alters the behavior of certain statements in the OS/VS COBOL language (for example, COPY). See the *Language Reference* for details. Note that no bounds checking is carried out on subscripts for programs compiled with this option set.

Default: **no os**vs

override(*reserved-word*) = = (*user-defined-word*)

Changes the COBOL reserved word to the specified user-defined word. See the *Language Reference* for a list of words which are reserved in the COBOL language. Although you should precede arguments to options with an equal sign when you specify them on the **cob** command line or in **\$COBOPT**, you cannot do this with the **override** option. You must enclose both the reserved word and its replacement in parentheses. There must be one space before the first equal sign and another after the second, although there must not be a space between them. This option does not appear in the list produced by the **setting** option.

If you set this option in your source code using the \$SET mechanism, there is only one equal sign between the old and the new word.

Default: no change of reserved words takes place.

perform-type = { **mf**
osvs
rm }

Causes PERFORM statements to behave as in the specified language:

mf The AIX VS COBOL standard type of PERFORM statement. See the *Language Reference* for details.

osvs The IBM OS/VS COBOL type of PERFORM statement. Under IBM OS/VS COBOL all the exit points of the PERFORM statements currently being executed are active simultaneously, unlike AIX VS COBOL type PERFORM statements which are strictly nested so only the exit point of the innermost PERFORM statement is active. Thus, under OS/VS COBOL, if control reaches any of the exit points of the current PERFORM statement, a return jump will occur. Under AIX VS COBOL, if control reaches any of the exit points of the outer levels of the current PERFORM, these will be ignored.

rm The **rm** type of PERFORM statement. See Chapter 13, "Ryan-McFarland COBOL: Conversion Series 3" for details.

Default: **perform-type = mf**

[no] profile

Allows the compiler to include code in your program to produce detailed performance statistics each time you run the program. See Chapter 7, "Running an AIX VS COBOL Program" for a description of the profile facility.

Default: **no profile**

[no] qual

no qual prohibits qualified data-names or procedure names in the program being compiled. This allows the compiler to operate more efficiently.

Default: **qual**

[no] query

Prompts you to supply the path name for a COPY file when the compiler is unable to find it. By default, this condition causes the compiler to output a severe compiler message. See Chapter 15, "Error Messages" for full details on compiler messages.

You are prompted as follows:

```
FILE BELOW NOT FOUND - Stop run Retry Continue Alter path  
  
test.cbl
```

Respond by entering one of the following:

- S** Terminate the compilation with errors.
- R** Retry to copy the file. Before typing **R**, you can place the copy file where the computer is searching for it so that it is found.
- C** Continue the compilation without including the specified copy file.
- A** Prompts you with Please input new path name. Respond by specifying the path in which the COPY file resides. If a correct path name is specified, the compiler prints OK and continues compilation. Otherwise, copy prompting starts over.

Default: **no query**

recmode = format

Determines the format of all the files in your source program unless you have specified a different format for a file in the FD statement. *format* can be either F, to denote fixed-length records, or V, to denote variable-length records.

Default: **recmode = F**

[no] ref

Causes four-digit location addresses to be included on the right-hand side of the listing file. Note that you may need a listing with location addresses in order to identify the locations reported in RTE error messages.

Default: **no ref**

remove = reserved-word

Disables the specified reserved word, allowing you to use it as a user-defined word within your source code. Note that if you wish to disable more than one reserved word you must specify a separate **remove** option for each word.

You can only **remove** words which are already enabled. For example, if you wish to disable the reserved word **ACTUAL**, you can only **remove** it if the **osvs** option has already been specified. This is because **ACTUAL** is a reserved word in the IBM OS/VS COBOL language and is only treated as a reserved word by AIX VS COBOL if the **osvs** option is set.

You must specify the **remove** option after any other options that affect reserved words have been specified. For example, if you specify the **remove** option in the **cobopt** file, you cannot specify any dialect-controlling options on the **cob** command line, since these are processed after options found in **cobopt**. See Chapter 4, "The COBOL Interface" for details on the order in which options are processed. This option does not appear in the list output by using the **setting** option.

You cannot use **remove** to remove the reserved words used to name and reference special registers from the reserved word list.

Default: No reserved words are removed.

[no] reseq

Causes the compiler to generate COBOL line sequence numbers, starting at 1, in increments of 1.

Default: **no reseq**

[no] retrylock

Specifies that a record found to be locked is retried until the record is released. This option is only effective if the **+R** and **+Q** run-time switches are set. See Chapter 7, "Running an AIX VS COBOL Program" for details on these switches.

Default: **noretrylock**

[no] rewrite-ls

Specifies whether **rewrite** of line-sequential files is permitted. If you use this facility it is your responsibility to ensure that the record written is the same size as the one replaced.

Default: **rewrite-ls**

[no] rm
rm = ansi

Changes the behavior of certain features so that they are compatible with RM/COBOL V2.0. If the **ansi** option is specified, these features behave as they do when a program is compiled in that system with the ANSI switch set. See the *Language Reference* for details.

Setting the **rm** option causes the table of alternate file status values described in the "RM Appendix" of the *Language Reference* to be used in file operations. By default, this table contains the status "9" file status values returned by RM/COBOL.

Setting **rm** automatically sets the **notrunc**, **oldindex**, **nooptional-file**, **retrylock**, **align = 2**, and **sequential = line** options. It also causes compiler behavior as if your program contained the syntax:

```
sign trailing separate
for signed numeric data items, and
lock mode is automatic
for each file with no explicit locking syntax.
```

Setting **rm = ansi** automatically sets the **notrunc**, **oldindex**, **nooptional-file**, **retrylock**, **align = 2**, and **sequential = record** options. It also causes compiler behavior as if your program contained the syntax:

```
sign trailing included
for signed numeric data items, and
lock mode is automatic
for each file with no explicit locking syntax.
```

Setting **norm** automatically sets the **trunc = ansi**, **nooldindex**, **nooptional-file**, **noretrylock**, **align = 8**, and **sequential = record** options. It also causes compiler behavior as if your program contained the syntax:

```
sign trailing included
for signed numeric data items. It does not set any locking for those files
which have no explicit locking specified.
```

Note: See also the option **perform-type**.

Default: Each of the options set by this one has its own default value. See the individual entries for each option in this chapter.

rtncode-size = *integer*

Specifies the size of the RETURN-CODE special register and its alignment in the computer memory. *integer* can be either 2 or 4. A value of 2 implies a data description of PIC S9(4) COMP for a register of 2 bytes which is aligned on a 2-byte boundary. A value of 4 implies a data description of PIC S9(9) COMP for a register of 4 bytes which is aligned on a 4-byte boundary.

If a program with a 4-byte return code returns control to a program with a 2-byte return code, binary truncation of the return code will take place. If a program with a 2-byte return code returns control to a program with a 4-byte return code, the top two bytes of the return code will be undefined.

Default: **rtncode-size** = 4

[no] **rw**

Specifies whether those reserved words that are specific to the ANSI COBOL standard Report Writer module should be treated as reserved words. See the *Language Reference* for details.

Default: **rw**

[no] **seg**

no seg causes the compiler to ignore segmentation by treating all section numbers as if they were zero. A monolithic program is produced.

Default: **no seg**

[no] **seqchk**

Checks the sequence numbers in columns one to six and flags lines whose sequence numbers are out of order.

Default: **no seqchk**

sequential = $\left\{ \begin{array}{l} \text{record} \\ \text{line} \end{array} \right\}$

Causes all files whose organization is implicitly or explicitly **sequential** to default to either **record-sequential** (a standard sequential file) or **line-sequential**. See Chapter 3, "Device- and File-Handling" for more information on file structures.

Default: **sequential** = record

[no] **setting**

Causes the compiler to include a list of the current settings of the majority of the compiler options in the listing file **.lst**. The settings of a few options are not shown in this list. See the descriptions of the individual options to determine which these are.

Default: **no setting**

sign = convention

Indicates whether included signs for numeric display fields are to be interpreted according to the ASCII or EBCDIC convention.

Default: **sign = ASCII**

[no] struct

Is reserved for use by COBOL products which can be added to the AIX VS COBOL system. Do not change its setting.

Default: **no struct**

[no] supff

Suppresses form feed characters in the output listing. This only has an effect with the **list** option.

Default: **no supff**

[no] time

Can only be used with the **date** option. Where **date** inserts the system date into the source program and listing, **time** adds the current system time.

Default: **time**

[no] trace

Specifies whether the **ready trace** and **reset trace** syntax should be enabled. **ready trace** and **reset trace** cause code to be inserted at each paragraph and section heading to display the name of that paragraph or section each time it is executed at run time. See the *Language Reference* for details.

Default: **no trace**

[no] trunc

trunc = ANSI

Controls the behavior of data moved into USAGE COMP items. **trunc** truncates decimal values to the number of digits specified by the PICTURE clause. **no trunc** truncates binary values to the capacity of the allocated memory (for small systems). **trunc = ANSI** truncates the decimal values of data moved by nonarithmetic statements to the number of digits specified by the PICTURE clause. For moves involving arithmetic statements when the size error condition occurs, if you specify **trunc = ANSI** but no **on size error** phrase, the value stored in the **usage comp** item is undefined.

Default: **trunc = ANSI**

[no] verbose

Sends messages output by the compiler concerning accepted options and the size of the code and data areas of your programs to **stdout**.

Default: **no verbose**

[no] **vsc2**

vsc2 = *integer*

Specifies that those reserved words that are specific to the IBM VS COBOL II language extensions should be treated as reserved words. See the *Language Reference* for details. It also enables or disables subscript array bound checking.

The possible values for *integer* are:

1 VS COBOL II Release 1.0

This replaces the options **oldvsc2** and **vsc2**.

- No explicit scope delimiter is allowed in a statement without a conditional phrase (AT END, ON SIZE ERROR, and so on).
- The word ALSO in an EVALUATE statement can be omitted.
- The CLASS and SYMBOLIC CHARACTERS clauses in the SPECIAL-NAMES paragraph are not allowed.
- Conditional phrases with NOT (NOT AT END, NOT ON SIZE ERROR, and so on) are not allowed.

2 VS COBOL II Release 2.0

- The CLASS and SYMBOLIC CHARACTERS clauses in the SPECIAL-NAMES paragraph are not allowed.
- Conditional phrases with NOT (NOT AT END, NOT ON SIZE ERROR, and so on) are not allowed.
- When used in conjunction with the **flag** = **vsc2** it provides similar functionality to the VS COBOL II Release 2.
- Also sets the compiler option **dbcs** = 1.

3 VS COBOL II Release 3.0

- When used in conjunction with the **flag** = **vsc2** it provides similar functionality to the VS COBOL II Release 3.

Notes:

1. **ans85** status codes are used when **vsc2** = 3 option is selected.
 2. Do not use the **noans85** option after specifying **vsc2** = 3 since this will turn off some of the **ans85** behavior supported by **vsc2** = 3.
- The CLASS and SYMBOLIC CHARACTERS clauses in the SPECIAL-NAMES paragraph are not allowed.
 - Conditional phrases with NOT (NOT AT END, NOT ON SIZE ERROR, and so on) are not allowed.
 - Also sets the compiler option **dbcs** = 2.

When **vsc2** is specified without *integer*, **vsc2** = 3 is assumed.

Default: **no vsc2**

warning = integer

Controls the level of compiler error messages output by the compiler. *integer* must be 1, 2, or 3. Unrecoverable-level and Severe-level errors are always output. Specifying *integer* as 1 causes Error-level errors to be output; 2 causes Error- and Warning-level errors to be output; and 3 causes all five levels of error messages to be output (Unrecoverable, Severe, Error, Warning, and Informational).

Default: **warning = 3**

[no] writelock

Causes WRITE and REWRITE statements to acquire a record lock when the program is locking multiple records in a file (see Chapter 8, "File Sharing in the Multi-User Environment"). This option does not appear in the list produced by the **setting** option if its state is the same as the **autolock** option. If this is the case, the state of the **fileshare** option shown in the **setting** list also indicates the state of the **autolock** and **writelock** options.

Default: **no writelock**

[no] xref

Produces a cross-referenced listing, consisting of a list of all data items in alphabetical order and an associated sequence number, which shows the line where the item is defined. This reference number is marked with a #. Further sequence numbers show each time the item is used. The listing also shows the data item type and the length (in bytes) of group items. The listing continues with a similar description of paragraph names.

Default: **no xref**

[no] zeroseq

Causes zero suppression in the sequence numbers in columns one to six.

Default: **no zeroseq**

Excluded Combinations

Certain options may not be used in combination with other options. Table 5-1 on page 5-24 shows the options that are excluded if the option shown adjacent in the left-hand column is specified.

Table 5-1. Excluded Combinations of Options	
Option	Excluded Options
nolist	list print [no] form reseq copylist errlist [no] ref echoall
errlist	reseq copylist [no] ref

ANS85 Options

The ANS85 options are as follows:

- ans85
- coms85
- nestcall
- nooptional-file
- trunc = ansi

Default Options

The default compiler options set by **cob** are as follows:

- align = 4
- alter
- noanalyze
- noanim
- ans85
- assign = dynamic
- noautolock
- nobell
- nobound
- nobrief
- charset = ASCII
- nocomp
- coms85
- confirm
- nocopylbr
- nocopylist
- currency-sign = 36
- date
- nodbcs
- nodbcssosi
- defaultbyte = 32
- nodg
- nodirectives
- echo

-
- noechoall
 - noerrlist
 - noerrq
 - nofilecase
 - nofileshare
 - noflag
 - noflagq
 - noflagstd
 - form = 60
 - noibmcomp
 - noibm-ms
 - int = *filename.int*
 - linkcount = 64
 - nolist
 - listwidth = 80
 - mfcomment
 - mf[level] = 4
 - noms(2)
 - native = ASCII
 - nestcall
 - nonls
 - noodoslide
 - nooldcopy
 - nooldfileio
 - nooldindex
 - nooldvsc2
 - nooptional-file
 - osex = cbl
 - noosvs
 - perform-type = mf
 - noprofile
 - qual
 - noquery
 - recmode = F
 - noref
 - noreseq
 - noretrylock
 - rewrite-ls
 - norm
 - rtncode-size = 4
 - rw
 - noseq
 - noseqchk
 - sequential = record
 - nosetting
 - sign = ASCII
 - nostruct
 - nosupff
 - time
 - notrace
 - trunc = ansi

-
- noverbose
 - novsc2
 - warning = 3
 - nowritelock
 - noxref
 - nozeroseq

You can override the above default compiler options by any of the following:

- An entry in the file **\$COBDIR/cobopt**.
- An entry in the environment variable **\$COBOPT**, or in the file to which this environment variable points.
- Specifying **-C** on the **cob** command line.
- Embedding parameters in the COBOL source code.

For example, in the **\$COBDIR/cobopt** file, the entry:

```
compiler: nolist nobell ␣
```

passes the options **nolist** and **nobell** to the compiler. The same effect would be achieved if you entered either of the following **cob** command lines:

```
cob -C "nolist nobell" filelist ␣
```

or

```
cob -C nolist -C nobell filelist ␣
```

Entries in the **\$COBOPT** environment variable (or the file it points to) override the system-wide default compiler options found in **\$COBDIR/cobopt**, while options specified on the command line override entries in both **\$COBDIR/cobopt** and **\$COBOPT**. Parameters in the source code override all of the above.

Mainframe Options

If you want the AIX VS COBOL system to emulate the mainframe environment, set the following compiler options. These options depend on the mainframe facilities used.

- sequential = line
- assign = external
- ibmcomp
- defaultbyte = 48
- native = EBCDIC
- OSVS
- notrunc

SAA Options

If you want the AIX VS COBOL system to provide support for the SAA definition of COBOL, you should set the following options:

- vsc2
- nomf
- flag(saa)

Options Permitted in \$SET Statements

You can imbed certain compiler options within your source code in \$SET statements. In the following list the entry "any" specifies that a particular compiler option can be specified in any \$SET statement regardless of its position within your source code; "initial" specifies that a particular compiler option can be specified only in a \$SET statement which appears at the start of your source code; while "not available" specifies that a particular compiler option cannot be specified in any \$SET statement.

align	Initial
alter	Initial
analyze	Not available
anim	Not available
ans85	Initial
assign	Initial
autolock	Initial
bell	Initial
bound	Initial
brief	Any
charset	Not available
comp	Initial
coms85	Initial
confirm	Not available
copylib	Not available
copylist	Any
currency-sign	Initial
date	Not available
dbcs	Initial
dbcssosi	Any
defaultbyte	Initial
dg	Initial
directives	Any
echo	Any
echoall	Any
errlist	Not available
errq	Any
filecase	Not available
fileshare	Initial
flag	Any
flagq	Any
flagstd	Any
form	Any
ibmcomp	Initial
ibm-ms	Initial
int	Not available
linkcount	Initial
list	Any
listwidth	Any
mfcomment	Any
mf[level]	Initial
ms(2)	Initial
native	Initial
nestcall	Initial
nls	Not available
odoslide	Initial

oldcopy	Any
oldfileio	Initial
oldindex	Initial
oldvsc2	Initial
optional-file	Initial
osext	Any
osvs	Initial
override	Initial
perform-type	Initial
profile	Not available
qual	Any
query	Any
recmode	Initial
ref	Any
remove	Initial
reseq	Initial
retrylock	Any
rewrite-ls	Not available
rm	Initial
rtncode-size	Initial
rw	Initial
seg	Initial
seqchk	Any
sequential	Initial
setting	Not available
sign	Initial
struct	Not available
supff	Any
time	Not available
trace	Any
trunc	Initial
verbose	Not available
vsc2	Initial
warning	Any
writelock	Initial
xref	Not available
zeroseq	Any

Compiler Messages

If you specify the **verbose** flag (**-v**) on the **cob** command line, each compiler option is acknowledged by the compiler on a separate line and is either accepted, rejected, or ignored. Options that are ignored are those which are not applicable to your environment. After all the options have been acknowledged, the compiler opens its files and starts to compile. At this point it displays the message:

```
* Compiling file-name
```

If any file fails to open correctly, the compiler displays:

```
Open fail : file-name
```

The compilation is aborted, returning control to AIX. Open fail results, for example, if the source file is located in another directory, or if the file name was typed incorrectly.

When the compilation completes successfully, the compiler displays a message which gives the total number of compiler error messages reported by the compiler, and the sizes of the code and data areas and the compiler dictionary. See "Listing Format" for details.

The dictionary size information does not include the overheads of the virtual memory mechanism, and so the dictionary file is likely to be larger than the statistic given here.

Listing Format

The general layout of the list file is as follows:

```
* IBM AIX VS COBOL Compiler/6000 LP <date><time> Page n
```

```
* <file-name>
```

```
<list of options>
```

```
1 Statement 1
```

```
:
```

```
:
```

```
n Statement n
```

```
* IBM AIX VS COBOL Compiler/6000 LP
```

```
* 5601-258 (c) COPYRIGHT IBM CORP. 1987, 1990
```

```
* Copyright (c) 1984, 1987 Micro Focus, Ltd
```

```
* All Rights Reserved
```

```
* Licensed Material - Property of IBM
```

```
* Last Error on page : nn
```

```
*
```

```
* Total messages: n
```

```
* Unrecoverable: n Severe: n
```

```
* Errors: n Warnings: n
```

```
* Informational: n Flags: n
```

```
* Data = nnnnn Code = nnnnn Dictionary = nnnnn
```

If no options were specified, the list of options is replaced by the message:

```
No Options Selected
```

Note that if you specify the **ref** option during compilation, a hexadecimal value denoting the address of each data-name or PROCEDURE statement appears to the right of the page. Addresses of data-names are relative to the start of the data area, while addresses of procedures (that is, sections and paragraphs) are relative to the start of the code area. There is some overhead at the start of the data area and a few bytes of initialization code at the start of the procedure area for each SELECT statement.

The sequence following **ref** is the compiler reference number.

A syntax error is marked in the listing by an error line with the following format:

```
nnnnnn illegal statement
```

```
** nnn-N*****... ..*****
```

```
(nnnn)**
```

```
** id# message-text
```

where:

nnnnnn is the sequence number of the erroneous line.

nnn is the compiler error number.

N is a single alphabetic character representing the category of the severity of the error.

The asterisks following the error number indicate the character position of the error in the preceding erroneous source line. The asterisks at the end of the line simply highlight the error line.

The compiler may not echo an erroneous line to the terminal. Check the previous line if there is any doubt.

The number *nnnn* in parentheses at the end of the line indicates the page of the listing on which the previous error occurred. This enables you to trace back from one error to the previous error. However, this feature assumes that you used the **copylist** option; if this is not the case, the page numbers reported are incorrect.

The line following the compiler error number has the text of the error message. At the beginning of each message is the AIX VS COBOL component identifier. This component number is 1103 for compiler error messages.

Compiler error messages are split into the following five categories:

Unrecoverable	Indicates a fatal error.
Severe	Indicates an error that the compiler was unable to correct. Compilation continues, but the statement at fault is not compiled.
Error	Indicates an error which the compiler has attempted to correct.
Warning	Flags a statement that although syntactically correct may contain a possible error.
Informational	Draws your attention to something in your source code you should be aware of.

An unrecoverable error always causes the compiler to stop running, outputting the relevant error message once it does. However, by default, any other level of error causes processing of the **cob** command to terminate once an intermediate code file has been produced. If the compiler reports any Severe-level compiler errors, you can override these by using the **-W** option with the **cob** command, or by setting the **warning** compiler option. See Chapter 4, "The COBOL Interface" for details. The message given at the close of the compiler's run, provided it is not terminated by an unrecoverable error, indicates both the total number and the category of the errors which occurred.

You will not be able to run or code-generate intermediate code programs which contain any Unrecoverable-level errors. You will have to correct these errors and resubmit your source code to the compiler using the **cob** command.

You can run programs which contain Severe-level errors only if you set the **E** run-time switch to on. See Chapter 7, "Running an AIX VS COBOL Program" for full details of how you may do this. If the **E** run-time switch is set off (**-E**), attempting to run intermediate code programs which contain Severe-level errors will give a run-time error, and the program run will terminate.

The default setting of the E run-time switch is **-E**; that is, you will not be able to run programs that contain Severe-level compiler errors. If you wish to do so you must explicitly set and export the COBSW environment variable to **+E** (see Chapter 4, "The COBOL Interface" for details).

You will not be able to produce object code from intermediate code programs which contain Severe-level errors. Attempting to do so will result in a Native Code Generator error.

You can animate programs with Severe-level errors regardless of the setting of the E run-time switch. If you animate such a program with the **-E** switch setting, a run-time error is reported, but animation does not terminate.

You can animate, run, and produce object files from intermediate code files which contain Error-, Warning-, and Informational-level errors, regardless of the setting of the E run-time switch. However, you may wish to correct these errors first.

A full list of compiler error messages together with recovery hints can be found in Chapter 15, "Error Messages."

Flagging

Flagging can be used to ensure portability of the COBOL syntax you have used in your program. If you use syntax that is outside the dialect of COBOL that you have selected in the **flag** option, it will produce a flagging message on the listing.

A flag is marked in the listing by a flagging line with the following format:

```
nnnnn          flagged feature
** nnn-level-----...  .... (nnn)--
** id# flag-text
```

where:

nnnnn is the sequence number of the flagged line.

nnn is the flag number.

level represents the level at which the feature is flagged, using the following acronyms:

MF	Micro Focus COBOL extensions
OSVS	IBM OS/VS COBOL extensions
VSC2	IBM VS COBOL II extensions
ANS74	ANSI COBOL Standard X3.23, 1974
ANS85	ANSI COBOL Standard X3.23, 1985
LOW	GSA ANSI Low level
L-I	GSA ANSI Low-intermediate level
H-I	GSA ANSI High-intermediate level
HIGH	GSA ANSI High level

The flagged feature is pinpointed at the position of the end of the line of characters beneath the flagged line. The dashes at the end of the line simply highlight the flagging line.

The number in parentheses at the end of the line indicates the page of the listing on which the previous flag occurred. This enables you to trace back from one flag to the previous one. However, this feature assumes that the **copylist** option has been used; if this is not the case, the page numbers reported are incorrect.

The line following the flag number and level has the text of the flag message. At the beginning of each message is the AIX VS COBOL component identifier. This component number is 1103 for flagging messages.

A program in which flags are indicated can still be run. Further details can be found in Chapter 15, "Error Messages."

If the compiler detects an error in a data declaration, it may skip some subsequent data declarations, with the result that error messages are produced when references are made to data items whose declarations have been skipped.

Chapter 6. Native Code Generator Options

Contents

About This Chapter	6-3
Permitted Options	6-4
Default Options	6-5
Native Code Generator Messages	6-6

About This Chapter

This section describes the system-wide default Native Code Generator options. You can supplement or override these default options by modifying the `$COBDIR/cobopt` supplied with your compiler, setting up your variable, `$COBOPT`, or by using the `-N` option on the `cob` command line. Chapter 4, “The COBOL Interface” contains details on how to do this.

Throughout this chapter all references to `$COBOPT` refer to the contents of the `COBOPT` environment variable, or the file to which it points.

Permitted Options

The available Native Code Generator (NCG) options are:

[no] asm [=filename]

Suppresses or requests an assembler listing of the intermediate file being generated. The assembly file is placed in:

prognam.e.s

where:

prognam.e is the name of the program being code-generated. This name may be overridden by the inclusion of a *filename* in the command line. The *filename* may include a path. If the *filename* is specified with this option, the **tmpdir** option is ignored.

Default: **no asm**

[no] boundopt

Specifies whether array access optimization is enabled. **Noboundopt** turns off array access optimizations and so allows you to access elements that are outside the array (this will increase execution time). If the **vsc2** or **osvs** compiler option is set at compile time, the default is **noboundopt**.

If the picture clause for a subscript has more digits than required for the table being indexed, **boundopt** may cause the compiler to use only the least significant bytes of the subscript. This reduces the amount of data used, and could adversely affect programs which access table elements outside of the declared size.

Default: **boundopt**

[no] check

Specifies checking of run-time limit violations (for example, **PERFORM** stack, table bounds). **nocheck** suppresses limit checking and so allows overwriting of data areas (reducing execution time). If the **vsc2** or **osvs** compiler options are set at compile time, the default is **nocheck**.

Default: **check**

[no] list [=filename]

Suppresses or requests the NCG error listing file. The listing may be sent to **stderr**, or may be overridden by including a *filename* in the command line.

Default: **list**

[no] lnkalign

Specifies that linkage records in a **USING** statement are 01 or 77 level items (they are aligned according to the compiler **align** option, as described in Chapter 5, "Compiler Options").

Note: This option may reduce the time needed to access a linkage item, but no checks are made to ensure that the items are aligned (your program could access data incorrectly).

Default: **noInkalign**

[no] obj [=filename]

Suppresses or requests production of a native object file of the intermediate file being generated. The object file is placed in:

progrname.o

where:

progrname is the name of the program being code-generated. This name may be overridden by the inclusion of a *filename* in the command line. The *filename* may include a path. If the *filename* is specified with this option, the **tmpdir** option is ignored.

Default: **obj**

[no] spzero

Causes spaces to be treated as zeros in numeric display fields. This option requires overhead at run time and so may reduce performance. Use the +F COBSW for equivalent function in the .int code.

Default: **nospzero**

[no] sysprof

Causes AIX system profiling code to be added to the COBOL generated native code. See the AIX system documentation for more information on the AIX system profiler.

Default: **nosysprof**

tmpdir = *path*

Sets the temporary directory path to be *path*. This must specify only a path; it cannot include the file name.

Default: Undefined

[no] verbose

Sends messages to the screen output by the Native Code Generator concerning accepted options and the size of your program's code and data areas.

Default: **noverbose**

Default Options

The default options can be overridden either by an entry in the file \$COBOPT or by using the -N flag on the **cob** command line. The following entry in the \$COBOPT file:

```
ncg:nocheck
```

passes the option **nocheck** to the Native Code Generator. An alternative way of achieving the same effect is the **cob** command line:

```
cob -N nocheck file list ◀
```

Entries in the \$COBOPT file override the system-wide default Native Code Generator options. Options specified to the **cob** command line override entries in both \$COBDIR/cobopt and \$COBOPT.

Native Code Generator Messages

If you specify the **verbose** option (-v) on the **cob** command line, each option is acknowledged by the Native Code Generator on a separate line and is either accepted, rejected, or ignored. After all the options have been acknowledged, the Native Code Generator opens its files and starts processing the file.

Chapter 7. Running an AIX VS COBOL Program

Contents

About This Chapter	7-3
Command Line Syntax	7-4
Command Line Examples	7-5
Examples	7-5
Switch Parameters	7-6
Run-Time Switches	7-7
ANIMATOR Switch (A)	7-7
Skip Locked Record Switch (B)	7-7
ANSI COBOL Debug Switch (D)	7-8
COBOL Symbol Switch (e)	7-8
Error Switch (E)	7-8
Compatibility Check Switch (F)	7-8
Keyboard Interrupt Switch (i)	7-9
ISAM Files Sequence Check Switch (K)	7-10
Memory Switch (l)	7-10
Null Switch (N)	7-10
Dynamic Linkage Setup Switch (p)	7-10
File Status Error Switch (Q)	7-11
Reread Locked Record Switch (R)	7-11
Sort Memory Switch (s)	7-12
Sort Switch (S)	7-12
Tab Switch (T)	7-12
Examples	7-13
Run Time Environment Error Messages	7-13
COBOL Profiler	7-14
Profiler Directives	7-14
Profiler Output	7-15

About This Chapter

This chapter describes how to run a program compiled with the AIX VS COBOL compiler. The chapter includes a description of the COBSW switches, which allow you to alter the way your program is run. Also included is a description of the format of run-time error messages. This section finishes with a description of the COBOL profiler facility, which can be used to produce statistics on the run-time performance of your program.

Command Line Syntax

You can run a statically-linked module output by the **cob** command by entering a command line of the form:

```
file-name [parameter-list] ↵
```

where *file-name* is the name of the **a.out** module output by the **cob** command and *parameter-list* is an optional list of parameters to be passed to AIX VS COBOL. Each parameter is a string, separated from adjoining parameters by one or more spaces. These parameters can be read by the module in either of the following ways:

- If a program linked into the module opens file **stdin** (console input) for input, with **ORGANIZATION LINE SEQUENTIAL**, the first **READ** from this file accesses the program parameters in the command line.
- **ACCEPT FROM CONSOLE** also reads from **stdin**, so the first **ACCEPT FROM CONSOLE** will also access the program parameters.

Note: **ACCEPT** without a **FROM** clause is, by default, **ACCEPT FROM CONSOLE**, unless **CONSOLE IS CRT** is specified in the **SPECIAL-NAMES** paragraph. **ACCEPT FROM CRT** does not access program parameters.

You can run dynamically loadable programs created by using the **cob** command by entering a command line of the form:

```
cobrun [option] [switch] filename [parameter-list]
```

where:

option is one of the two following options that can be passed to the **cobrun** command:

- h** is a help option that displays a usage banner on the screen; no other action is taken by the **cobrun** command if the **-h** option is given.
- v** is a verbose option that shows the built command line that is being executed by the **cobrun** command in order to process your requested run.

switch is an option list of switches. See "Switch Parameters" on page 7-6 for details on what these switches can be.

filename is the name of the **.int** or **.gnt** file output by the **cob** command. If both **.int** and **.gnt** versions of the file exist and you do not specify an explicit file extension in the **cobrun** command, the **.gnt** version is the one that is run.

parameter-list is an optional list of parameters to be passed to your COBOL program. Parameter lists are described in more detail earlier in this section.

Command Line Examples

This section shows examples of command lines.

Examples

For a full description of the effect of the **cob** command in the following examples, see Chapter 4, "The COBOL Interface."

1. To execute the statically linked module **pi**, in which the generated version of the program **pi** is linked to the COBOL libraries, enter the following:

```
cob -x pi.cbl ␣  
pi ␣
```

2. To execute the intermediate code file **pi.int**, which is the default output of the **cob** command, enter the following:

```
cob pi.cbl ␣  
cobrun pi.int ␣
```

3. To animate the intermediate code contained in the file **pi.int** output by the **cob** command, enter the following:

```
cob -a pi.cbl ␣  
anim pi.int ␣
```

If you set the A switch in the COBSW environment variable, the same effect can be achieved by entering the following commands:

```
cob -a pi.cbl ␣  
cobrun pi.int ␣
```

4. To execute the generated code in the file **pi.gnt** output, using the **cob** command, enter the following:

```
cob -u pi.cbl ␣  
cobrun pi.gnt ␣
```

5. To execute the statically linked executable file named **pi**, which is output by the **cob** command, enter the following:

```
cob -x -e"" pi.cbl ␣  
pi pi ␣
```

Specifying the **-e** option with a null argument on the **cob** command line ensures that the entry point is read from the command line at run time. Since the entry point is supplied at run time, you can use the static module **pi** to execute any intermediate or native code file. For example, to run the file **myfile.int**, enter the following command:

```
pi myfile ␣
```

where *myfile* could be a free-standing intermediate or native code program, or could call the statically linked **pi** module.

Switch Parameters

You can set certain switches when you execute files output, using the **cob** command. These switches can be any of the following:

- Run-time switches
- ANIMATOR switch (A)
- Skip locked record switch (B)
- ANSI COBOL debug switch (D)
- COBOL symbol switch (e)
- Error switch (E)
- Compatibility check switch (F)
- Keyboard interrupt switch (i)
- ISAM files sequence check switch (K)
- Memory switch (l)
- Null switch (N)
- Dynamic linkage setup switch (p)
- RM file status error switch (Q)
- Reread locked record switch (R)
- Sort memory switch (s)
- Sort switch (S)
- Tab switch (T)

To specify any of these switches, set the environment variable **COBSW** to those which you require. Each switch you want to set to on must be preceded by a '+' sign. Each switch that you wish to set to off must be preceded by a '-' sign.

If your program does not require any of the above switches, you do not need to set **COBSW**. In this case, the default values of the switches will apply.

Note: The use of '+' or '-' is significant for these switches, but the values are used only to delimit the run-time options. If you are running the "sh" shell, you will need to export **COBSW** after you set it, as follows:

```
export COBSW
```

You can also include run-time switches in the **cobrun** command line as follows:

```
cobrun [switch] filename [parameter-list] ␣
```

where:

- switch* is an optional list of the run-time switches listed above.
- filename* is the name of the **.int** or **.gnt** file output by the **cob** command.
- parameter-list* is the optional list of parameters to be passed to AIX VS COBOL.

Run-Time Switches

By default, all run-time switches are set to off. AIX VS COBOL provides a facility that allows you to control events in a program at run time by setting or unsetting up to nine run-time switches in the SPECIAL-NAMES paragraph of your program (see the *Language Reference* for details).

To set a run-time switch to on, set the COBSW environment variable as follows:

```
COBSW=+n
```

where *n* is in the range 0 to 8.

You can specify switches in any order, but each individual switch must be preceded by a sign. For example:

```
COBSW=+1+4
```

sets the run-time switches as follows:

0	Off
1	On
2	Off
3	Off
4	On
5	Off
6	Off
7	Off
8	Off

ANIMATOR Switch (A)

By default, this switch is set to off. If you wish to animate intermediate code output by the **cob** command, you can do so using the **anim** command (see Chapter 11, “Debugging Your Program Using ANIMATOR”), which automatically sets the ANIMATOR switch to on. To use the **cobrun** command to execute intermediate code, but still have ANIMATOR invoked, set the COBSW environment variable to the ANIMATOR switch as follows:

```
COBSW=+A
```

If you animate your program using COBSW = +A when COBPATH is set, the **.cbl** files must be in the same directory as the **.int** files.

Skip Locked Record Switch (B)

By default, this switch is set to off. If you attempt a READ operation on an ISAM file opened for INPUT or I-O and the record is found to be locked, the file position indicator remains pointing to that record. Subsequent READs then attempt the read operation again until the record is found to be unlocked. However, if the Skip Locked Record Switch is set to ON, if a READ operation tries to access a record that is locked, then the file position indicator is moved to the next record in the file. Since the record was locked, an I-O status “record locked” is returned for that READ.

To set this switch to on, set the COBSW environment variable to the Skip Locked Record Switch as follows:

```
COBSW=+B
```

This switch only has effect for files with sequential access.

ANSI COBOL Debug Switch (D)

By default, this switch is set to off. If your program is to use the ANSI COBOL debug facility (see *Language Reference*), set the COBSW environment variable to the ANSI COBOL debug switch, as follows:

```
COBSW=+D
```

COBOL Symbol Switch (e)

Setting this switch off causes the RTE to search only for COBOL programs or symbols to satisfy CALL operations or EXTERNAL data items. The RTE will not search for any C programs or symbols if you set this switch off.

By default, this switch is set on, that is, the RTE does not search for C programs and symbols. If you wish to turn it off, you must set the COBSW environment variable as follows:

```
COBSW=-e
```

The method used to avoid finding C programs and symbols is that the RTE will not look up that name in the file "ldtab.s". This is a file that is created during a compilation where such names are collected for resolution. The RTE does the lookup for names in this file. Therefore, if the code that handles the CALL does not go through the RTE, finding C programs and symbols cannot be avoided by using this switch. The -e switch will only have effect if the calling program is a dynamically loaded COBOL program, that is, a .gnt or .int code file. If the COBOL program is statically bound, then a CALL to C code (which must also be statically bound) will have no need to go through the RTE, and so finding the C program or symbol will not be avoided.

Error Switch (E)

By default, this switch is set to off. If you try to run intermediate code programs which contain S-level compiler errors, you will receive a run-time error. See Chapter 15, "Error Messages" for more information about these errors. To execute intermediate code output by the cob command, which contains S-level compiler errors, set the COBSW environment variable to the Error switch as follows:

```
COBSW=+E
```

Compatibility Check Switch (F)

By default, this switch is set to on. The switch enables or disables various checks at run time to allow programs that would usually fail at run time with run-time error 163 to run successfully. Run-time error 163 is output for a number of error conditions, all of which are described in Chapter 15, "Error Messages." Most of these conditions are cases where intermediate code versions and generated code versions of the same source program produce different effects at run time. To suppress the checking for these error conditions, set this switch to off, as follows:

```
COBSW=-F
```

For example, one of the checks enabled or disabled by the compatibility check switch is the numeric field check. If this switch is set to on when the RTE loads a numeric item into one of its numeric registers, the RTE will check the loaded value to see if the value contains a nonnumeric character. If this is the case, run-time error 163 is output.

Consider the following short program:

```
WORKING-STORAGE SECTION.  
01 VAR PIC 9.  
  
PROCEDURE DIVISION.  
  IF VAR = 0 DISPLAY "ZERO"  
  ELSE DISPLAY "NON-ZERO".
```

The variable VAR is not initialized with a VALUE clause and so contains space (hex 20). If you try to run this program with the compatibility switch set to on (the default setting), the RTE checks for a nonnumeric character in the numeric field and the program fails with run-time error 163.

If you run the intermediate code version of this program with the compatibility switch suppressed, the RTE masks out the top four bits of the value in VAR. This makes the comparison true, and the program displays "ZERO". However, if you then run the generated code version of this program with the compatibility switch suppressed, the RTE performs a byte by byte comparison between the value of VAR (hex 20) and zero (hex 00). In this case, the comparison yields the result false, and the program displays "NON-ZERO".

The following table summarizes the results of running the intermediate and native code versions of the above program with the two settings of the compatibility check switch:

	-F	+F
.int	Displays "ZERO"	Fails with RTE error 163
.gnt	Displays "NON-ZERO"	Displays "NON-ZERO"

If an intermediate code file needs the F switch suppressed (that is, -F set) to run successfully, you must set the SPZERO Native Code Generator option if you want to generate your code and obtain the same results. See Chapter 6, "Native Code Generator Options" for details.

Keyboard Interrupt Switch (i)

By default, this switch is set to on. The AIX VS COBOL system allows you to enable or disable keyboard interrupts by setting or unsetting the interrupt switch at run time. If you want keyboard interrupts to be disabled, set this switch to off as shown below:

```
COBSW=-i
```

ISAM Files Sequence Check Switch (K)

By default, this switch is set to off. The AIX VS COBOL system allows you to enable or disable sequence-checking of indexed keys in ISAM files. This switch allows you to specify if records can be written in any order to ISAM files opened in sequential mode, or if these records must be written in key sequence. You can set this switch to on by entering the following:

```
COBSW=+K
```

Memory Switch (I)

By default, the Run Time Environment performs logical CANCELs unless all the available memory has been used up. As far as your program is concerned, the behavior of logical and real CANCELs is identical, but logical CANCELs are faster. A logical CANCEL flushes all file buffers but does not free any memory after using it. If you do not set the size of the available memory, the RTE requests the maximum amount possible from the operating system. Set the size of the available memory by setting the memory switch at run time.

```
COBSW=-1 integer
```

where *integer* is the size of the available memory in bytes.

If you want all CANCELs to be "real," set this switch to:

```
COBSW=-10
```

By default, when the RTE requires memory space it checks that the new request does not exceed the available memory. If the new request exceeds available memory, the memory that should have been freed by any CANCEL is freed, and the RTE repeats its request for memory. The RTE loads programs that have been logically canceled in preference to reloading from fixed-disk.

Null Switch (N)

By default, the null switch is set to on, as follows:

```
COBSW=+N
```

When a program writes records to a line-sequential file, the default action in cases where a record contains control characters (all characters with ASCII code less than or equal to hex 1B) is to add a null character (hex 00) before each control character. Similarly, on reading a record from a line-sequential file, the default action is to strip these null characters from control characters.

If you wish control characters to be written and read in the same way as other characters, set this switch to off, as shown below:

```
COBSW=-N
```

Dynamic Linkage Setup Switch (p)

If you set the **p** run-time switch off as follows:

```
COBSW=-p
```

the RTE will set up COBOL parameters (that is, Linkage Section items) on demand. By default, this switch is set on which causes the RTE to set up the addressability for all the Linkage Section items in a program when it is called.

You may find that setting the `-p` run-time switch for subprograms with large linkage areas gives improved run-time performance, since the linkage setup is done the first time an item is accessed. Thus, if an item is not accessed, no setup is performed for it.

File Status Error Switch (Q)

By default, this switch is set to off, as follows:

```
COBSW=-Q
```

By setting the file status error switch to on, all status 9 file errors reported in your code are mapped, by means of an internal RTE table, to a status which conforms with the statuses returned in the COBOL dialect of your choice. Any undefined or unrecognized status values are mapped onto status 30, permanent I-O error. See the *Language Reference* for full details on file status errors.

If you want file status errors to be mapped to the values in this alternate table, set this switch to on, as follows:

```
COBSW=+Q
```

The defaults in this alternate table are the file status values used in RM COBOL.

The contents of the internal RTE table that allows this mapping to be performed are as follows:

```
unsigned char alt_Stat[256];
```

The table is indexed by the second byte of any status 9 file errors. Any entry is interpreted as a two-digit binary coded decimal (BCD) number, which is converted to two ASCII digits. This number is stored in place of the original file status. If you want to alter this table, you can do so by patching or by supplying a C routine to alter certain table entries at run time. See "Alternate File Status Table" on page 3-18 for more details on using an alternate file status table.

Reread Locked Record Switch (R)

By default, this switch is set to off. You can set it to on as follows:

```
COBSW=+R
```

If a read is attempted on a file of any organization that is OPEN for INPUT or I-O and has no file status item declared, and the record is found to be locked, the READ is attempted again until the record becomes available (provided the switch is set to on).

Note: This feature makes possible a situation in which two applications cannot proceed because each is trying to access a record locked by the other. Should this situation occur, you will probably have to kill the process.

This behavior is also available on files with file status items declared (but no declaratives) providing you set the `retrylock` compiler option when you compile your program. In this case you must also set the `+Q` run-time switch when you run such programs.

If this switch is set to off, if a READ is attempted on a record which is locked, the READ is not attempted again.

Sort Memory Switch (s)

The Sort Memory switch sets the size to allocate for internal workspace to be used for sorting files.

The default size is equal to the size of 1000 records.

To override the default, set this switch as follows:

```
COBSW=+snnnn
```

where *n* is the number of bytes to be allocated for the sorting workspace. The sort workspace is not allocated until the actual sorting operations begin. When the SORT is complete, the workspace is returned to the system.

Setting this switch will affect the performance of your application.

- If the size allocated is too small for your application, then many work files will be created. The management of these multiple files will reduce performance.
- If the size allocated is too large for the amount of real memory installed on your system, then the system may thrash as it tries to manage that sort file workspace.

When it is necessary to set this switch, the user must consider the actual needs of the application to estimate a reasonable value for the size of this sort workspace. A possible estimation method would be to use a reasonable multiple of the logical record size for the sort file. Setting the value very large will not always be optimal.

Sort Switch (S)

By default, this switch is set to on. You can set it to off as follows:

```
COBSW=-S
```

The S switch forces all SORT statements within your source code to list duplicates in the order in which they appear in the input stream. If this switch is set to off, duplicates are not guaranteed to be in any particular order, even if the SORT...WITH DUPLICATES IN ORDER statement is used. The benefit of setting the sort switch to off is that this allows the Run Time Environment to use a faster sorting algorithm.

Tab Switch (T)

By default, this switch is set to off. When a program writes records to a line-sequential file, the default action is to expand tab characters and output multiple spaces in the record as they occur. You can cause multiple spaces to be replaced by tab characters by setting the tab switch to on, as follows:

```
COBSW=+T
```

On input, tab characters are always expanded to spaces.

When the tab switch is set on, the REWRITE clause does not work correctly on line-sequential files containing tab characters. The tab characters expand when you READ the record. This makes the record to be rewritten longer than the record to be read. The record you rewrite must be the same length as the record you read.

Examples

Switches can be separated by spaces, although spaces are not required. However, do not include a space between the sign ('+' or '-') and its associated switch. If you do include spaces between the switches, then you must quote the entire string so it will be accepted by the AIX shell as a single environment variable.

1. COBSW=+D+T

This enables the ANSI COBOL debug switch and replaces multiple spaces in line-sequential files with tab characters.

2. COBSW=+0-i

This sets run-time switch 0 to on and disables keyboard interrupts.

Run Time Environment Error Messages

Run Time Environment errors are reported by the Run Time Environment (RTE) and may occur when you are running the compiler, ANIMATOR, the Native Code Generator, or one of your own COBOL programs. An RTE error is returned on a program that is syntactically correct but has problems during the actual running of the intermediate code.

RTE error messages are output by the operating system in the following form:

```
action error: file 'filename'  
error code: 999, pc=nnnnnnnn, call=m, seg=x  
999 id# message-text
```

where:

action is what the RTE was doing at the time the message was caused (for example, execution, load, or write).

filename is the name of the file on which the RTE was operating.

999 is the RTE message number. Possible message numbers are listed in Chapter 15, "Error Messages."

nnnnnnnn is a hexadecimal number giving the address of the program counter.

m is a number that is used internally to identify the program that is in error. This is 1 for a main program or greater than 1 for a subprogram.

x is a number that identifies the segment containing the error when .int code is executed and the *seg* option was used. *x* will be 0 if the error is in the root, or from 51 to 99 if the error is in an overlay. If *noseg* was used for .int or .gnt code, *x* will always be 0. *x* has no meaning for statically bound code.

id# will appear at the beginning of each RTE message. For RTE messages, this AIX VS COBOL component identifier is 1203.

message-text is text associated with the message number. The possible message texts are listed in Chapter 15, "Error Messages."

COBOL Profiler

The COBOL profiler is a facility that lets you obtain detailed statistics on the run-time performance of a COBOL program.

When you submit a COBOL source program to the **cob** command, you can specify the **profile** compiler option (see Chapter 5, "Compiler Options"). This option causes the compiler to include code in your program to produce performance statistics each time you run the compiled program. Each time you run a program that was submitted to the **cob** command with the **profile** option set, a file is produced called:

name.ipf

where *name* is the program name if intermediate code was executed, or the first entry-point name if native code was executed. This file contains the performance statistics for that run of the program in a compact form. To convert this compact form of the performance statistics to a readable format, use the **cobprof** program.

The **cobprof** command creates a file called:

name.prf

where *name* is the name of the program. This file contains the performance report for that run of the program.

To run **cobprof**, use a command line of the form:

```
cobprof file-list [+directive-list] ←
```

where:

file-list is a list of files containing compact profiler output. Specify only the root (that is, program) names; do not use the **.ipf** file extension. File names should be separated by one or more spaces.

directive-list is an optional list of directives that control the operation of **cobprof**. The following section describes each of these directives.

Profiler Directives

Profiler directives should be separated by one or more spaces.

The profiler directives are:

alpha Performance statistics are output in alphabetic order by paragraph name. If you do not specify **alpha**, statistics are output in descending order of the total percentage time spent in each paragraph.

all A full performance profile is output. If you do not specify **all**, no statistics are produced for sections or paragraphs that are not entered or PERFORMed during execution of the program.

form "integer"

Specifies the assumed page size for the listing file. The default is 60 lines. The value of integer must be within the range of 3 to 9999.

When you enter this directive, you must escape the double quotes by typing a \ in front of them. Parenthesis also may be used in place of the quotes, but they also must be escaped.

list ["destination"]

Specifies where the listing is to be produced. If you do not specify **list** at all, the output is produced in file *name*.prf, where *name* is the first name in *file-list* in the command line.

If you specify **list** on its own, the output is sent to the console. In this case, page heading and line feeds are omitted.

If you specify **list** with a filename, the output is written to that file.

Specifying **list** on its own automatically sets the **verbose** option.

When you enter this directive, you must escape the the double quotes by typing a \ in front of them. Parenthesis also may be used in place of the quotes, but they also must be escaped.

[no]verbose

Displays messages output by PROFILER on the screen. The default is **noverbose**.

wide

This allows lines in the profiler output to be up to 131 characters wide. If you do not specify **wide**, lines are truncated to 79 characters.

Profiler Output

When you run **cobprof** with the **+verbose** directive you will see the following:

```
IBM AIX VS COBOL Compiler/6000 LP
5601-258 (C) Copyright IBM Corp. 1987, 1990
Profiler V2.0
Copyright (c) 1984, 1987 Micro Focus Ltd.
All Rights Reserved
Licensed Materials - Property of IBM
* name-1
.
.
.
* name-n
```

where each *name* is one of the names in *file-list* in the command line.

For example, if you submit the demonstration program **stock1** to the **cob** command with the **profile** directive, run **stock1**, using **cobrun**, and then enter:

```
cobprof stock1 +LIST ␣
```

The output will be in the following format:

* IBM AIX VS COBOL Compiler/6000 LP
* 5601-258 (C) Copyright IBM Corp. 1987, 1990
* Profiler V2.0
* Copyright (c) 1985 Micro Focus Ltd.
* All Rights Reserved
* Licensed Material - Property of IBM
* stock1

%time	time	entries	ms/entry	Module called once. paragraph
51.43	576	11	52	CORRECT-ERROR
25.71	288	1	288	SR1
17.14	192	1	192	END-IT
5.71	64	3	21	NORMAL-INPUT
0.00	0	1	0	INITIAL (UNNAMED) PARAGRAPH

For each section and paragraph in the program, the following information is given:

- % time

The total percentage of execution time spent in that section or paragraph

- time

The total time (in milliseconds) spent in that section or paragraph

- entries

The number of times the section or paragraph was entered

- ms/entries

The average time (in milliseconds) per entry to the section or paragraph.

Note: The product of the average time and the number of entries should equal the total time spent in the section or paragraph. However, because all three values are truncated, there may be a slight discrepancy. The output also includes the total execution time (in milliseconds).

The INITIAL (UNNAMED) PARAGRAPH in the program is the initialization code executed before the user part of the program is entered.

Chapter 8. File Sharing in the Multi-User Environment

Contents

About This Chapter	8-3
A Typical Multi-User Environment	8-4
Including Multi-User Syntax in Your Program	8-4
Facilities for Multi-User AIX VS COBOL	8-4
Data Locking	8-5
Organization of Shared Files	8-6
The Procedure Division	8-11
File Status	8-11
Demonstration Programs	8-13
Running the Demonstration Programs	8-13

About This Chapter

AIX VS COBOL provides the ability for independent COBOL programs to share data files in its multi-user environment. Each program that uses shared files can have access to those files with file security maintained in a predictable and controllable manner.

You specify file sharing with COBOL syntax in your program. See the *Language Reference* for details of this syntax. See “Including Multi-User Syntax in Your Program” on page 8-4 and “Organization of Shared Files” on page 8-6 for further information on COBOL syntax for the various types of file organization.

If you run AIX VS COBOL within a single-user environment, this multi-user syntax is accepted by the compiler, although the multi-user syntax has no effect when you run your programs. This allows you to develop applications designed for use within multi-user environments in a single-user environment.

A Typical Multi-User Environment

Using AIX VS COBOL, you can lock either a single record, a group of records, or a whole file. When data is locked by one program, no other program can delete or change that data.

If the program locks the whole file (called an *exclusive lock*), no other program can access that file. If your program locks records (either single record locks or multiple record locks), the file can be shared with other programs. Other programs may also lock records in the same file. Each program can lock a single record or multiple records. Any program can access any data that is not locked by another program.

Your program can open several data files at the same time and can specify locking for each data file opened. However, your program is allowed only one type of locking for each file, as follows:

- Locking the whole file
- Locking a single record
- Locking multiple records.

When you do not explicitly specify locking in your program, files opened as I-O, OUTPUT, or EXTEND acquire an exclusive lock by default. That is, the whole file is locked by your program. If your program opens the file for INPUT, the file becomes shareable, and your program cannot hold record locks on the file. Other programs can open the file for INPUT or I-O in a shareable mode. This default locking is used for each file that your program opens when no LOCK MODE syntax is included in your program.

Including Multi-User Syntax in Your Program

All of the syntax that is used to make data files shareable is part of the ANSI 1985 Standard X3.23.

Existing programs written without multi-user syntax can be modified to make them suitable for use in a shared file environment. If you require default locking, submit your original source programs to the **cob** command. For any other kind of locking, add the required syntax to your source program before submitting it to the **cob** command. If your program does not already contain a status item, you must declare one in your program and add any code you require to handle specific status information. When a file status data item is included in your program and the program tries to read a file, a status is returned in this data item. The value in this data item tells you whether the operation was successful (see Chapter 3, "Device- and File-Handling" and the *Language Reference*).

Facilities for Multi-User AIX VS COBOL

By using the compiler options **autolock** or **fileshare**, you can change the default locking to automatic locking of single records without any extra syntax in your program. It is recommended that new programs explicitly specify the required lock mode in the SELECT statement.

You can obtain more sophisticated locking on a single record or a group of records if you include extra syntax in your program. Depending on the organization of the shared files, there are different syntax and programming considerations. The sections that follow describe how to program for shared sequential or line-sequential files, relative files, and indexed sequential files.

Data Locking

There are three types of data locking:

- Automatic locking
- Manual locking
- Exclusive locking.

These data locking mechanisms can be used in the following manner:

Automatic Locks a single record or multiple records. Automatic single record locks mean that when the program reads a record from a file opened for I-O, that record is automatically locked until the program next accesses that file. For files that have been opened for INPUT, records are never locked. Files opened for OUTPUT cannot support automatic locking and always hold an exclusive lock on the whole file.

Your program cannot access a file in automatic (shareable) mode if another program has already opened the file in the exclusive mode.

You can lock records using automatic multiple record locks. The records are locked automatically as they are read and are not released until a CLOSE, UNLOCK or COMMIT statement is executed. When the **writelock** or **fileshare** compiler options are specified, WRITE and REWRITE statements also acquire a record lock when you are locking multiple records.

Manual Locks a single record or multiple records. Manual record locking is similar to automatic, except that you must explicitly lock the record when it is read. That is, you must specify READ WITH LOCK (single records) or READ WITH KEPT LOCK (multiple records) to acquire a lock. As with automatic, only files opened for I-O can acquire record locks. In addition, with multiple records, WRITE and REWRITE statements also acquire a lock if you have specified the **writelock** or **fileshare** compiler option.

Exclusive Locks the entire file as soon as your program executes an OPEN statement on the file. Your program cannot open a file in exclusive mode if another program is already accessing the file. To obtain an exclusive lock on a file, you must have READ and WRITE permissions for that file. With exclusive data locking, the file remains locked until it is closed. If your program opens a data file for OUTPUT, this implies an exclusive lock on the file.

In a multi-user environment, each program can open more than one data file, and each program may have access to the same data files. A file that is shareable can be accessed by one or more programs, each locking one or more records in the file. Figure 8-1 on page 8-6 shows a hypothetical multi-user environment.

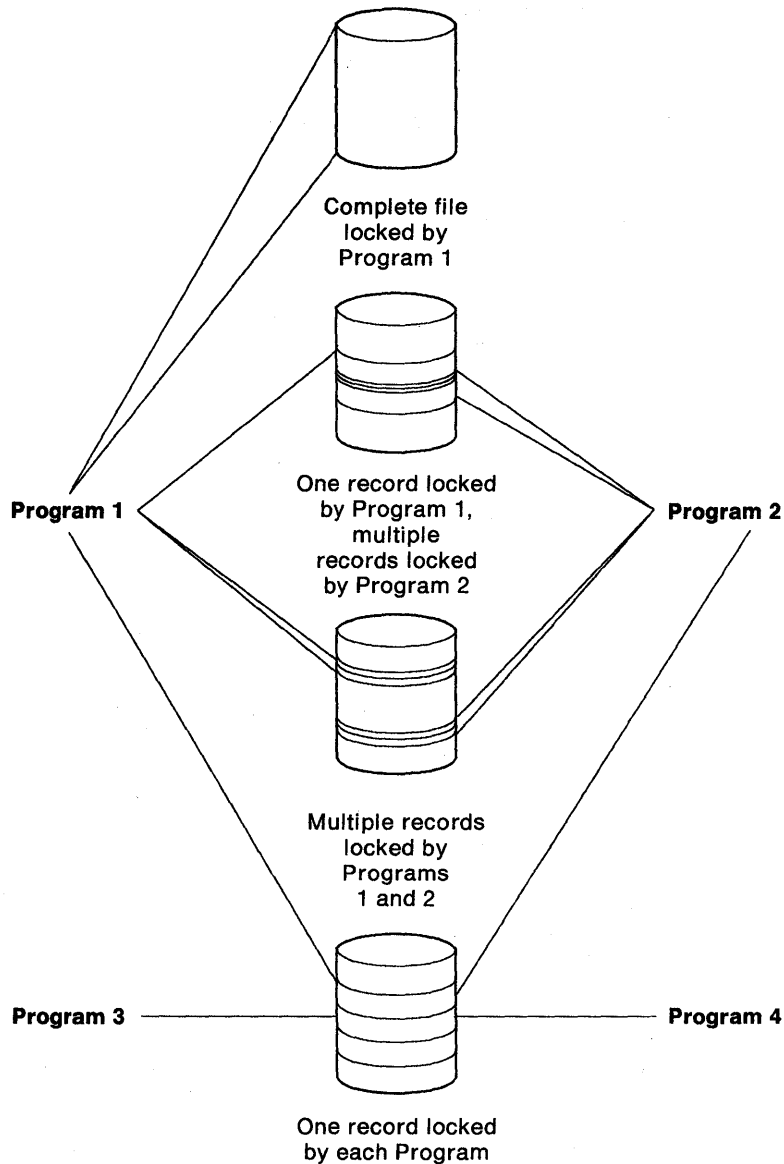


Figure 8-1. A Hypothetical Multi-User Environment

Organization of Shared Files

Depending on the organization of the shared files, there are different syntax and programming considerations. The sections that follow describe how to program for sequential, relative, and indexed sequential files.

Record-Sequential and Line-Sequential Files

When your program uses record-sequential files, you can lock individual records or whole files. You cannot lock groups of records. When your program uses line-sequential files, there is no record locking at all, only file locking. This is because line-sequential files contain variable-length records, and therefore cannot be opened as I-O.

The FILE-CONTROL paragraph for record-sequential and line-sequential files is as follows:

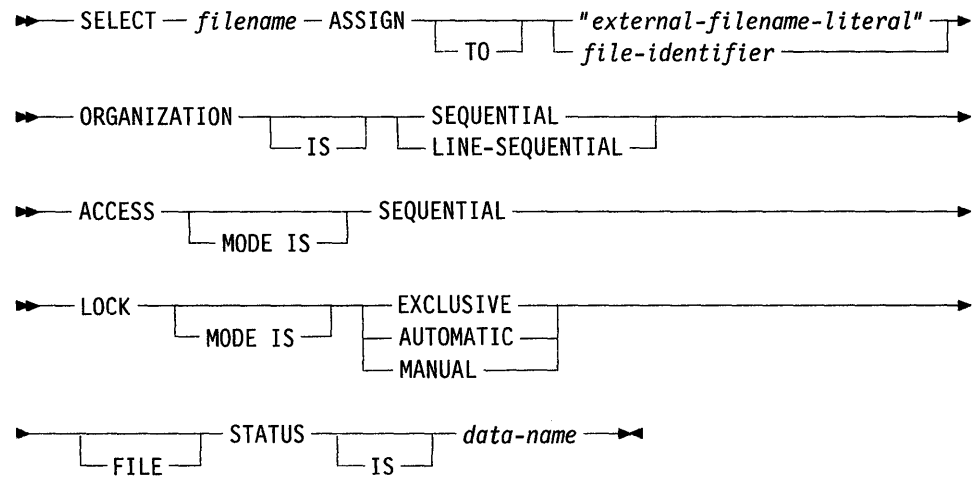


Figure 8-2. FILE-CONTROL Paragraph Syntax for Record and Line-Sequential Files

Each time your program accesses a file with an OPEN, READ, or READ WITH LOCK statement, the file locking you have specified is taken into account. This means that:

- When you specify LOCK MODE IS EXCLUSIVE, the whole file is locked from the time your program opens the file unless the file has been opened for INPUT. If the whole file is locked, other programs will still be able to access the file and to read it, provided they have both READ and WRITE access to the file, but the exclusive lock will prevent other programs from modifying the file.
- When you specify LOCK MODE IS AUTOMATIC, a single record is locked as the program reads it.
- When you specify LOCK MODE IS MANUAL, single records are locked as the program executes a READ WITH LOCK on them.
- When you do not specify a LOCK MODE IS clause, the default locking is used. Files opened for INPUT are shareable. Files opened for OUTPUT, I-O, or EXTEND are exclusive.

Relative Files

When your program uses relative files, you can lock whole files, single records, or groups of records. The FILE-CONTROL paragraph for relative files is shown in Figure 8-3 on page 8-8.

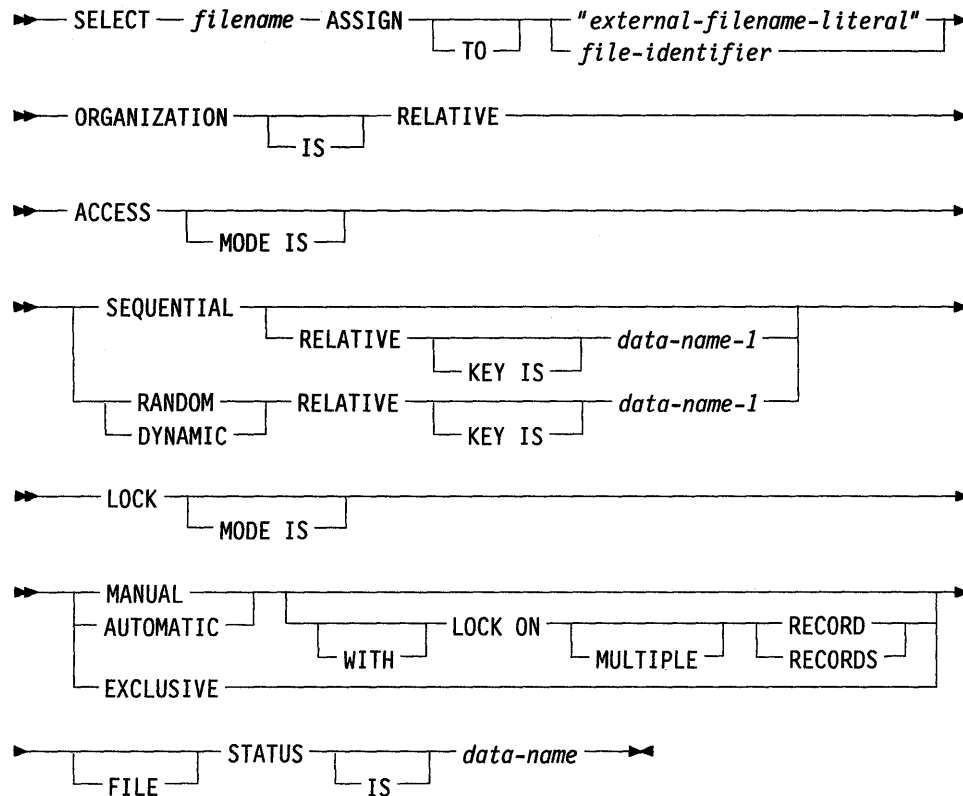


Figure 8-3. FILE-CONTROL Paragraph Syntax for Relative Files

You can lock several records simultaneously using the **WITH LOCK ON MULTIPLE RECORDS** clause. If you receive the following RTE message,

213 Too many locks.

you must close the file or execute a **COMMIT** statement or an **UNLOCK** statement to release these records.

Both manual and automatic locking can be performed on files with both single record locking and multiple record locking.

Each time a file is accessed with an **OPEN**, **READ**, **READ WITH LOCK**, **READ WITH KEPT LOCK**, **WRITE**, or **REWRITE** statement, the file locking you have specified is taken into account. This means that:

- When you specify **LOCK MODE IS EXCLUSIVE**, the whole file is locked from the time your program opens the file unless the file was opened for **INPUT**. If the whole file is locked, other programs will still be able to access the file and to read it, provided they have both **READ** and **WRITE** access to the file. The exclusive lock will prevent programs from modifying the file in any way.
- When you specify **LOCK MODE IS AUTOMATIC**, a single record is locked as the program reads it.

-
- When you specify **LOCK MODE IS AUTOMATIC WITH LOCK ON MULTIPLE RECORDS**, multiple records are locked as your program executes **READ** statements. Records remain locked until one of the following occurs:
 - The file is closed.
 - A **COMMIT** statement is executed.
 - An **UNLOCK** statement is executed.
 - When you specify **LOCK MODE IS MANUAL**, single records are locked as your program executes a **READ WITH LOCK** on the record.
 - When you specify **LOCK MODE IS MANUAL WITH LOCK ON MULTIPLE RECORDS**, multiple records are locked as your program executes **READ WITH KEPT LOCK** statements. Records remain locked until one of the following occurs:
 - The file is closed.
 - A **COMMIT** statement is executed.
 - An **UNLOCK** statement is executed.
 - When you do not specify a **LOCK MODE IS** clause, the default locking is used. Files opened for **INPUT** are shareable. Files opened for **OUTPUT** or for **I-O** are exclusive.

When your program is locking multiple records, you can also acquire a record lock on a **WRITE** or **REWRITE** statement. To do this, you must specify the **writelock** or **fileshare** compiler option when you submit your program. See Chapter 5, “Compiler Options” for details on these options.

Unless you explicitly include the **WITH LOCK ON MULTIPLE RECORDS** clause, single record locks are assumed when the lock is automatic or manual.

Indexed Sequential Files

With indexed sequential files, you can lock whole files, single records, or groups of records. The **FILE-CONTROL** paragraph for indexed sequential files is as follows:

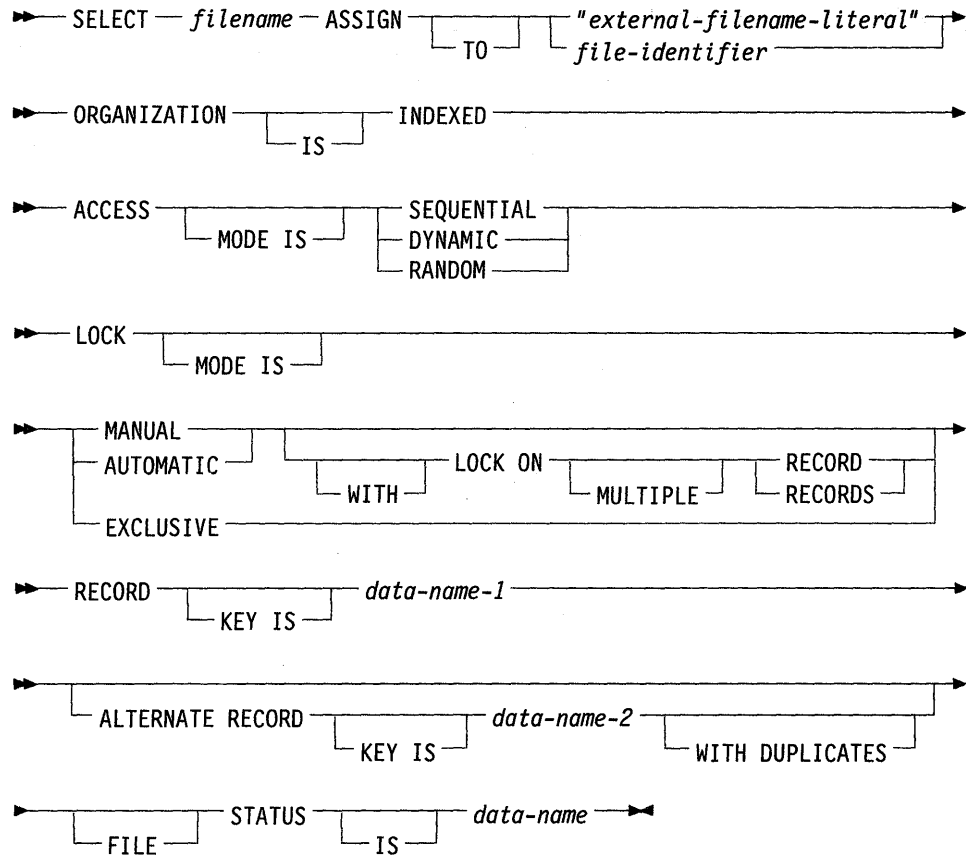


Figure 8-4. FILE-CONTROL Paragraph Syntax for Indexed Sequential Files

As with relative files, use the **WITH LOCK ON RECORD** clause to specify single record locking and the **WITH LOCK ON MULTIPLE RECORDS** clause to specify multiple record locking. You can lock single or multiple records in either manual or automatic lock mode.

You can **OPEN** an indexed sequential file only if you have **WRITE** permission to the index portion of the file.

The Procedure Division

Generally, COBOL programs designed to be run in a multi-user environment do not require any extra consideration in the Procedure Division. One exception to this is when you are checking the file status for a lock condition (see “File Status”). Other instances where you need to use a different syntax in your Procedure Division are as follows:

- The COMMIT statement

This statement releases record locks on all records in all the files the program has opened. This includes SEQUENTIAL, RELATIVE, and INDEXED files, with automatic or manual locking on single and multiple records. The COMMIT statement has no effect on exclusive files.

- The UNLOCK statement

The statement UNLOCK *filename* releases all record locks your program has acquired on the specified file. You may use this statement only for files that are shareable.

- The READ statement

With shareable files opened for I-O whose LOCK MODE is manual, you must include the WITH LOCK phrase (for single record locks) or the WITH KEPT LOCK phrase (for multiple record locks).

Additionally, with shareable files with multiple record locking, the REWRITE and WRITE statements may also lock the record that is acquired. REWRITE and WRITE statements lock records if the **writelock** or **fileshare** compiler option is specified. For more information, see Chapter 5, “Compiler Options.”

File Status

In a multi-user environment, the file status item set up with the FILE STATUS IS *dataname* clause in the FILE-CONTROL paragraph is used to check the status of a file operation. This section explains how to interpret the status codes returned in *dataname*.

The *dataname* you specify must be a two character alphanumeric data item. The first character of *dataname* is called status key 1. This character reports on the success or failure of an input-output operation on a file. The second character of *dataname* is status key 2. If any further information is available, it is returned in status key 2. Redefine the status key 2 as a PIC 9(4) COMP item so that this data item can hold the error message numbers.

See the *Language Reference* for the values that may be returned in these status keys. See Chapter 15, “Error Messages” for a list of the run-time errors.

Additionally, where status key 1 contains the value 9 (operating system error message), status key 2 can contain any of the following values that are specific to a multi-user environment:

- 65 Locked file. Another program has already locked the file to the exclusion of other programs.
- 68 Locked record. Another program has already locked the record.
- 213 Too many locks. The program has already acquired the maximum number of locks on the file. You must execute a COMMIT, UNLOCK, or CLOSE statement to release record locks before continuing.

Test specifically for these conditions in your program. Also, check for status codes, and decide what action you want your program to take upon finding the various status codes.

If another program has already locked the record your program wants to acquire, an attempted WRITE, REWRITE, or DELETE operation will fail.

If your program attempts to read a record already locked by another program, a lock status (error 68) is returned in the file status data-item. However, valid data is also returned. When a sequential read finds a record lock, the current record pointer is not updated. The START... KEY IS GREATER THAN statement can be used to skip over locked records in relative or indexed files.

If your program finds a lock on a record that you are attempting to START, the record lock is ignored, and the current record pointer is updated.

Handling a File or Record Lock

Whenever your program tries to access a file that has been exclusively opened by another program, you must wait until the other program has closed the file before you can access the data in the file.

When your program finds a record lock, you must wait until that record has been released before your program can access it. In the case of a program that has locked multiple records, you must wait until the other program executes a COMMIT or UNLOCK statement, or closes the file.

If a READ is attempted on a file which is OPEN for INPUT or I-O and has no file status item declared, and the record is found to be locked, the READ operation is attempted again at one second intervals until the record becomes available, if you have set the R (Reread Locked Record) run-time switch on. See Chapter 7, "Running an AIX VS COBOL Program" for details.

Note: This feature makes possible a situation in which two applications cannot proceed because each is trying to access a record locked by the other. Should this situation occur, you will probably have to kill the process.

Sharing Files on Multi-User Systems

See your *Language Reference* for more information about how files are shared between users in a multi-user environment.

Demonstration Programs

Your AIX VS COBOL software includes several programs that show how to write programs to run in a shared file environment. The demonstration programs are as follows:

- | | |
|---------------------|---|
| mudemo.cbl | The COBOL source for the controlling program. This program displays information and presents the program with a choice of access and input modes. Depending on the program's choice, one of the four subprograms may be called. |
| stockin.cbl | The COBOL source for the subprogram that demonstrates opening a shared data file for input only. |
| stockioa.cbl | The COBOL source for the subprogram that demonstrates opening a shared data file for I-O, and with automatic record locking. |
| stockiom.cbl | The COBOL source for the subprogram that demonstrates opening a shared data file for I-O, and with manual record locking. |
| stockout.cbl | The COBOL source for the subprogram that demonstrates opening a shared data file for output only. |

Running the Demonstration Programs

After you install the AIX VS COBOL software as described in Chapter 1, "Introduction," submit the demonstration programs to the COBOL process using the following command:

```
cob -x mudemo.cbl stockin.cbl stockout.cbl stockioa.cbl stockiom.cbl
```

Move the resulting executable module **mudemo**, the base name of the first file input to the **cob** command, to an area where users on both terminals can access it. Make sure that both users are accessing the same file.

To run the multi-user demonstration programs, enter the following on both terminals:

```
mudemo
```

Both screens display the initial screen shown in Figure 8-5.

```
=====                               Date dd/mm/yy
IBM AIX VS COBOL                       Time hh:mm
=====

This is a demonstration program for use with VS COBOL. The program
demonstrates how multi-user VS COBOL can lock both records and files.
The program allows an indexed file to be opened in a number of
modes, which demonstrate the locking facility. For more information
on locking refer to the Language Reference Manual.

-----
1. Input 2. I-O Lock Mode Automatic 3. I-O Lock Mode Manual 4. Output 5. Exit

INPUT CHOICE [0]
```

Figure 8-5. Initial Display Screen of the Demonstration Program

When the initial screen is displayed, each operator chooses the access and lock modes by pressing the number associated with the required mode, then pressing **↵**.

Set up the multi-user environment by creating a data file that the two operators can share. One operator must create this file by selecting choice 4 to open the file for output. This display screen shows a “Stock Control System” with stock code, stock description, stock held, and cost per unit. The bottom of the display screen shows the open and lock modes, what the last operation was, whether the last operation was successful, and the file status. Again, there is a choice of operations for the program to perform.

Enter data in the following fields:

- Stock code
- Stock description
- Stock held
- Cost per unit fields.

Use the tab key to move from one field to the next.

Write the data into the data file **mustock.DAT** by writing the record. To do this, operator 1 selects option 1 (write record) and press **↵**. Write five or six more records, and then close the file by selecting option 2 (Exit) and pressing **↵**.

While program 1 is creating a file to share, program 2 can try to access the data file **mustock.DAT**. Program 2 will fail to gain access to this file, because opening a file for output locks the file exclusively. Program 2 will receive a file locked status.

After user 1 creates the **mustock.DAT** file, both users can access the data file at the same time.

If user 1 selects option 2, I-O Lock Mode Automatic on the initial display screen, and accesses the first record, then the first record in the data file **mustock.DAT** is locked by program 1, and remains locked until program 1 accesses the file again. Program 2 can access any other record in the file by choosing to open the file for I-O Lock Mode Automatic or for Input.

Program 2 may try to access the first record, but receives a file status of Record locked and the access is unsuccessful. However, the data will be returned.

Try the various combinations of locking and access for yourself so you become familiar with the way AIX VS COBOL locks data.

Chapter 9. Advanced Programming Features

Contents

About This Chapter	9-3
Library Subroutines	9-4
cobsetjmp and coblongjmp	9-4
cobtidy	9-5
RTE Subprograms	9-5
Put a Character to the Screen	9-6
Read a Character from the Keyboard	9-7
Split/Join a File Name	9-7
File-Related Operations	9-8
Modifying the Behavior of User Attributes	9-9
Modifying the Behavior of ACCEPT/DISPLAY	9-9
Display Screen Input and Output	9-11
Test Keyboard Status	9-13
Sound the Audible Alarm	9-13
Move the Cursor to a Defined Position	9-13
Pack Byte	9-14
Unpack Byte	9-14
CRT Screen Handling	9-14
The ACCEPT and DISPLAY Statements	9-14
Display Attributes	9-15
Screen Handling From C	9-16
Using Escape Sequences to Send Attribute Information to the Screen	9-19
File Handler	9-20
Interface to the COBOL File Handler	9-21
Operation Codes Passed in the Second Byte of the First Parameter	9-21
Information Passed in the FCD at Open Time	9-22
Information Passed for Other Operations	9-22
FCD Information Format	9-23
Key Definitions for Indexed Files	9-25
Global Information	9-25
Key Definitions	9-25
Component Definitions	9-26
CISAM Features	9-26

About This Chapter

IBM AIX VS COBOL provides the following advanced programming features:

- Library subroutines
- Run Time Environment (RTE) subprograms (for performing special functions)
- CRT screen handling
- File handler.

This chapter describes these features.

Library Subroutines

This section describes the library routines provided with your AIX VS COBOL system.

cobsetjmp and **coblongjmp**

The library routines **cobsetjmp** and **coblongjmp** are provided with the AIX VS COBOL system. These routines provide functions similar to the C routines **setjmp** and **longjmp** (see the AIX Operating System documentation) for details of **setjmp** and **longjmp**). These library routines provide a non-local GO TO to use in error-handling and exception-handling.

cobsetjmp saves the environment of the current COBOL program in the buffer provided by the USING parameter, and returns immediately with the status flag set to 0. A subsequent call to **coblongjmp** from somewhere else in the program that called **cobsetjmp**, or from one of the program's subprograms, causes execution to be resumed at the point immediately after the call to **cobsetjmp**.

Note: Before calling **coblongjmp**, the status flag in the buffer may be set to a non-zero value. This allows the value to be tested after the **cobsetjmp** call.

The compiler option **nonestcall** must be set to use these library routines. It is recommended that you set this option in your program source with the \$SET statement.

cobsetjmp and **coblongjmp** can only be used in native code.

Example

A typical use of the **cobsetjmp** and the **coblongjmp** routines is as follows:

```
01 err-buf.
   02 err-stat pic 9(8) comp.
   02 err-env  pic X(N).

procedure division.
p-00.
   call "cobsetjmp" using err-buf.
   if err-stat not equal 0
      perform error-handling.
   go to main-loop.

main-loop.
   perform get-operator-input.
   perform process-input.
   perform create-report.
   go to main-loop.

process-input.

   if 'something went wrong'
      move 5 to err-stat
      call "coblongjmp" using err-buf.
      .
      .
      .
```

where the value of N is based on the `sizeof (jmpbuf)` field found in the `setjmp.h` file in the `/usr/include` directory. The value of N should be the value of the `sizeof (jmpbuf)` field plus 20.

Restrictions

When using these routines, the following restrictions apply:

- **coblongjmp** must be called from the same or lower level CALL/PERFORM hierarchy as was **cobsetjmp**. In the intervening time, control must not be returned to a higher level.
- If **coblongjmp** returns control to a CALL level that differs from the one used by **cobsetjmp**, programs exited by this mechanism appear to have been exited normally and may be CALLED again later in the program run.
- **coblongjmp** cannot be used if a CHAIN has been made since **cobsetjmp** was last CALLED.

cobtidy

The **cobtidy** library routine closes all files opened by the AIX VS COBOL system, and frees all the memory it has used.

Format

The format of **cobtidy** is as follows:

```
void cobtidy();
```

Usage

The **cobtidy** routine can only be called from non-COBOL modules. It can be used to ensure that all COBOL file buffers are flushed and closed when the COBOL system is not to be re-entered. Applications normally use the COBOL verb CHAIN if the current COBOL environment will be closed, and another created. To exit from a non-COBOL module in the AIX VS COBOL system to the operating system, you must use the **cobexit** function.

RTE Subprograms

Included in the RTE are a number of subprograms that you can call from an AIX VS COBOL program. These provide functions that are not available in the COBOL language itself.

Note: Use these routines sparingly and with caution. They are not compatible with every language extension listed in the *Language Reference*.

RTE subprograms are called by a CALL statement in the following form:

```
CALL X "hh" USING parameter-list
```

where *hh* is a two-digit hexadecimal code that identifies the RTE subprogram, and *parameter-list* is a list of the data items in your program to be passed as parameters (the number and type of parameters depend on the particular RTE subprogram you are calling).

In the following descriptions of the calls to these subprograms, the arguments are described to give the correct size of the data object when the **ibmcomp** option is not used. In that case, those objects will be 1 byte in size. If the **ibmcomp** option is used, however, that object would be 2 bytes in size and the call to the RTE subprogram would fail. To adapt these descriptions for use with the **ibmcomp** option, change

```
PIC 99 COMP    to    PIC X
```

You can give a value to these items with a hex specification, e.g.,

```
VALUE x"02"
```

to set the value to 2. See the *Language Reference* for a discussion of the **ibmcomp** option and its effect on data object sizes.

Currently supported RTE subprograms are as follows:

Code	Description
82	Put a character to the screen
83	Read a character from the screen
8C	Split a filename
8D	Join a filename
91	A number of miscellaneous routines mainly connected with file-related operations
A7	A number of routines that affect the behavior of user attributes
AF	A number of routines that affect the behavior of ACCEPT/DISPLAY statements in a program
B7, B8	A number of routines that provide several functions for handling memory-mapped display screens
D9	Test the keyboard status
E5	Sound the audible alarm
E6	Move the cursor to a specified position
F4	Pack byte
F5	Unpack byte

Put a Character to the Screen

The subprogram with call code X"82" allows you to display a character on the screen. A call to this subprogram has the form:

```
CALL X"82" USING character
```

where *character* is a PIC X field containing the character to be displayed at the current cursor position. The cursor moves one position to the right. If the initial position of the cursor is in the last column of a line, the subsequent position of the cursor is dependent on the terminal you are using.

Read a Character from the Keyboard

The subprogram with call code X"83" accepts a character from the keyboard. A call to this subprogram has the form:

```
CALL X"83" USING character
```

where *character* is a PIC X field that contains the character that is returned.

Split/Join a File Name

The subprograms with the call codes X"8C" and X"8D" allow you to separate the standard file names into their component parts, or to join the parts to make a standard file name. The component parts of a file name are the directory (for example, /usr/demo), the name itself (for example, prog), and the file extension (for example, cbl) which together form the file specification (for example, /usr/demo/prog.cbl). A call to this subprogram has the form:

```
CALL X"8C" USING filespec,directory,filename,extension
```

to split a file name or:

```
CALL X"8D" USING filespec,directory,filename,extension
```

to join a file, where:

- *filespec* is a PIC X field of variable length, but not less than 101 bytes, containing the full file specification
- *directory* is a PIC X field of variable length, but not less than 101 bytes, containing the directory pathname
- *filename* is a PIC X field of variable length, but not less than 15 bytes, containing the name of the file
- *extension* is a PIC X field of variable length, but not less than 15 bytes, containing the file name extension.

The split subprogram takes the string found in *filespec* and stores its component parts in *directory*, *filename*, and *extension*.

The join subprogram takes the strings found in *directory*, *filename*, and *extension* and combines them to form a complete file specification which it stores in *filespec*.

The AIX VS COBOL system uses these two subprograms to:

- Produce default listing and intermediate code file names from the source file name
- Produce overlay names
- Produce file names for segments and their inter-segment reference files.

The values allowed in *filespec*, *directory*, *filename*, and *extension* must conform to the standards described in the documentation supplied with the AIX operating system, and each data item must end with a space character. You must ensure that the area allocated within the WORKING-STORAGE section for each data item is not less than the minimum length given above.

File-Related Operations

The RTE subprogram with call code X"91" provides access to seven RTE subprograms. A call to this subprogram has the following form:

```
CALL X"91" USING result, function, file
```

where *result* is the name of a PIC 99 COMP field in which a status code is returned. A zero status code indicates that the call was successful; a nonzero code indicates that the call failed for some reason. For function 18, *file* is the name of a group item that identifies the file on which the subprogram is to operate. This group item is declared as follows:

```
01 FILE.  
  02 NAME-SIZE PIC 99 COMP.  
  *  
  * THE NUMBER OF CHARACTERS IN NAME  
  *  
  02 FILE-NAME PIC X(n).  
  *  
  * THE NAME ITSELF  
  *
```

For functions 46 through 53, *file* is the name of the file descriptor (FD) of the file the subprogram is using. *function* is the name of a PIC 99 COMP field whose value indicates which of the seven RTE subprograms controlled by this call code is to be called. The possible values are as follows:

Value	Description
18	Delete the file.
46	Set the null switch on for the file.
47	Set the null switch off for the file.
48	Set the tab switch on for the file.
49	Set the tab switch off for the file.
52	Use 2-byte record terminators for line-sequential and relative files.
53	Use 1-byte record terminators for line-sequential and relative files.

Functions 46 through 49 apply only to line-sequential files. When you run an AIX VS COBOL program, you can set switches in the COBSW environment variable that determine how control characters and tab characters are treated in line-sequential file records read or written by the program. These functions allow you to override the COBSW switch settings for particular files within the program. See Chapter 7, "Running an AIX VS COBOL Program" for a description of the null and tab switches.

When your program writes records to a line-sequential or relative file, the default is to include a 1-byte record terminator, value hexadecimal 0A (line feed). You can alter this default for particular files by using function 52, which causes a 2-byte record terminator to be used; this has the value hexadecimal 0D0A (carriage return - line feed). Use function 53 to restore the default of 1-byte terminators.

Modifying the Behavior of User Attributes

The RTE subprogram with call code X“A7” invokes the cursor’s display screen-handling system and gives access to a number of RTE subprograms that affect the behavior of the user attribute. When enabled, the user attribute causes all of the characters shown on the display screen to have the same (specified) attribute. See Chapter 10, “Configuring Your AIX VS COBOL System” for further details on the cursor’s display screen-handling system and the user attribute.

A call to this subprogram has the following form:

```
CALL X"A7" USING function, parameter
```

where *function* is the name of a PIC 99 COMP field whose value indicates which of the RTE subprograms controlled by this call code is to be called. *parameter* is the name of a PIC 99 COMP field whose value depends on the value of *function*.

The *function* field may take any of the following values:

Value	Description
6	Read the current user attribute contained in parameter.
7	Set the current user attribute, which is held in parameter.
16	Turn the user attribute on or off. <i>parameter</i> can contain one of two values: <ul style="list-style-type: none">• 0 to turn the user attribute on• 1 to turn the user attribute off.

The user attribute is initially disabled. Once enabled, some of the methods you can use to show text on the display screen (such as DISPLAY...UPON CONSOLE, display screen input-output subprograms, and the ADIS subprogram) use this attribute. The ANSI form of the DISPLAY statement (DISPLAY...UPON CONSOLE) uses the user attribute only if one of the other DISPLAY methods has been used previously. DISPLAY SPACE UPON CONSOLE clears the display screen to the user attribute for each display screen position.

Modifying the Behavior of ACCEPT/DISPLAY

The RTE subprogram with call code X“AF” gives access to a number of RTE subprograms that affect the behavior of ACCEPT and DISPLAY statements in a program. ACCEPT and DISPLAY statements are handled by a part of the RTE called ADIS.

A call to this subprogram has the following form:

```
CALL X"AF" USING function,parameter
```

where *function* is the name of a PIC 99 COMP field whose value indicates which of the RTE subprograms controlled by this call code is to be called. *parameter* is the name of a data item whose size and type depends on the value of *function*.

The *function* field may take any of the following values:

Value	Description
1	Allow individual user function keys, or a series of consecutive user function keys, to be enabled or disabled at run time. You must have already set up the actual key codes for the user functions with the keybcf utility. See Chapter 10, "Configuring Your AIX VS COBOL System" for details. <i>parameter</i> is a group item consisting of the following four data items: <ul style="list-style-type: none">• A PIC 99 COMP field that contains 0 to disable user function keys, or 1 to enable them.• A PIC X field whose value must be 1.• A PIC 99 COMP field that contains the number of the first function key to be enabled or disabled. This number is defined with the keybcf utility. See Chapter 10, "Configuring Your AIX VS COBOL System" for details.• A PIC 99 COMP field that specifies the number of consecutive function keys that are to be enabled or disabled. These numbers are defined with the keybcf utility.

18 Display a character to the display screen at the current cursor position. *parameter* is the name of a PIC X item containing the character to be displayed.

22 Sound the terminal alarm. *parameter* is the name of a PIC X item that can contain any value.

27 Get a character from the keyboard. *parameter* is the name of a 3-byte group item declared in the form of a CONSOLE status data item (see the *Language Reference*). A keystroke is read from the keyboard, and *parameter* is updated as follows:

Byte 1	Meaning
---------------	----------------

1	The second byte contains the number of a user-defined function key, in binary (in the range 1 to 127). See Chapter 10, "Configuring Your AIX VS COBOL System" for more details on function keys.
2	The second byte contains the number of an ADIS function key, in binary (in the range 1 to 127). See Chapter 10, "Configuring Your AIX VS COBOL System" for more details on function keys.
3	The second byte contains the ASCII code of the keyed character.
9	The second byte contains one of the following error codes: <ul style="list-style-type: none">8 A disabled character has been keyed and byte 3 contains the character.9 An invalid keystroke (more than one byte) has occurred.

This subprogram also causes the cursor's display screen-handling system to be invoked. See Chapter 10, "Configuring Your AIX VS COBOL System" for more information.

The RTE subprogram with the call code “_raw_display” gives access to an additional RTE subprogram that affects the behavior of DISPLAY statements in a program.

A call to this subprogram has the following form:

```
CALL "_raw_display" USING function
```

where *function* is the name of a PIC 99 COMP field whose value indicates which DISPLAY mode you wish to be in.

The *function* field may take any of the following values:

Value	Description
0	Normal mode. All non-printable characters are converted to spaces prior to writing them to the screen. This is the default mode.
1	Raw mode. Non-printable characters are allowed. The raw mode does a direct write to stdout , bypassing the AIX VS COBOL screen interface package completely.

Use of raw mode is not recommended. Use of this mode causes all optimization features of the screen handling module to be bypassed, thereby degrading the performance of screen output. Hardcoding of escape sequences is never recommended for portable, maintainable programs in which a variety of terminals are to be used. The AIX VS COBOL system cannot guarantee the future support of hardcoded escape sequence programming, nor guarantee a consistent result for escape sequences or combinations of escape sequences that are hardcoded. Please refer to “Using Escape Sequences to Send Attribute Information to the Screen” on page 9-19 for more information.

Raw mode allows the ability to write escape sequences directly to the screen (**stdout**) bypassing the AIX VS COBOL screen interface package completely. This mode might be useful when you want to send an escape sequence to a terminal to enable an auxiliary port but do not want to modify the terminal screen. In this case, you should call **_raw_display** passing a 0 in the *function* parameter to go back into the normal mode immediately after the DISPLAY statement.

Display Screen Input and Output

The RTE subprograms with call codes X“B7” and X“B8” give access to a number of RTE subprograms that control display screen input and output. These subprograms cause the cursor’s display screen-handling system to be invoked.

You should not use the X“B7” run time environment call in conjunction with AIX VS COBOL ACCEPT and DISPLAY statements that also specify attributes. This is because you can cause semantic inconsistencies as to whether text ACCEPTed or DISPLAYed by these statements should complement, override, or be overridden by attributes placed on the screen map by the X“B7” call.

Thus, where you use X“B7” calls to specify attributes, any ACCEPT and DISPLAY statements used to place text on the area of the screen affected by these calls only appears in the attribute specified by these calls if no other ACCEPT or DISPLAY statements have been executed that also specify any attributes. Otherwise, the effect is undefined.

A call to the X“B7” subprogram has the following form:

```
CALL X"B7" USING function, parameter, buffer
```

where *function* is the name of a PIC 99 COMP data item whose value indicates which of the RTE subprograms controlled by this call code is to be called.

parameter is a group item consisting of the following three data items:

- A PIC 9(4) COMP field showing the length of the data to be read or written.
- A PIC 9(4) COMP field giving the start position on the display screen. Top left is position 1 and 81 is the start of the next line, assuming an 80-column display.
- A PIC 9 (4) COMP field showing the start position in the buffer, starting from position 1.

buffer is the COBOL data area. It is a PIC X (*n*) field and may be as large or as small as you require in order to write your data. *function* may take any of the following values:

Value	Description
0	Read a string of characters from the display screen.
1	Write a string of characters to the display screen.
2	Read a string of attributes from the display screen.
3	Write a string of attributes to the display screen.
4	Clear a specified string of consecutive character positions to spaces.
5	Clear a specified string of consecutive character positions to normal attributes.
6	Write a specified character to a string of consecutive character positions.
7	Write a specified attribute to a string of consecutive character positions.

The RTE subprogram with call code X“B8” gives access to a number of other subprograms that affect display screen input and output.

A call to the X “B8” subprogram has the following form:

```
CALL X"B8" USING function, parameter, text-buffer, attribute-buffer
```

where *function* is the name of a PIC 99 COMP data item whose value indicates which of the RTE subprograms controlled by this call code is to be called.

parameter is a group item consisting of three data items:

- A PIC 9(4) COMP field showing the length of the data to be read or written.
- A PIC 9(4) COMP field giving the start position on the display screen.
- A PIC 9(4) COMP field showing the start position in the buffer, starting from position 1.

function may take any of the following values:

Value	Description
0	Read strings of text and attributes from the display screen.
1	Write strings of text and attributes to the display screen.
2	Swap the text and attributes on the display screen with those in the text and attribute buffers, respectively.

Test Keyboard Status

You can use the RTE subprogram with call code X“D9” to determine whether there is a character waiting to be read from the keyboard. The call of this subprogram has the following form:

```
CALL X"D9" USING parameter
```

where *parameter* is the name of a PIC 99 COMP item. The subprogram returns a zero value in *parameter* if there is no character waiting to be read. The subprogram returns a non-zero value if there is a character waiting to be read.

You must use the syntax CONSOLE IS CRT in the SPECIAL-NAMES paragraph for this call to have effect.

Although this subprogram does not invoke the **curses** display screen-handling system, its effect is undefined if it is invoked.

Sound the Audible Alarm

The subprogram with call code X“E5” causes the audible alarm (the CRT bell) to sound. A call to this subprogram has the form:

```
CALL X"E5"
```

Move the Cursor to a Defined Position

The subprogram with call code X“E6” positions the cursor at the specified screen position. A call to this subprogram has the form:

```
CALL X"E6" USING result,parameter
```

where *result* is not used and *parameter* is a 01 level item containing:

```
02 ROW-NUMBER PIC 99 COMP.  
02 COLUMN-NUMBER PIC 99 COMP.
```

The value of ROW-NUMBER must be in the range 1 to 25, and the value of COLUMN-NUMBER must be in the range 1 to 80.

Pack Byte

The subprogram with call code X"4F" takes eight 1-byte fields from an array, and uses the least significant bit of each byte to form a 1-byte field. The first occurrence of the array becomes the most significant bit of the new byte (bit 7). A call to this subprogram has the form:

```
CALL X"4F" USING byte,array
```

where *byte* is a PIC 99 COMP field that contains the new byte and *array* is a PIC 99 COMP OCCURS 8 field that contains the eight bytes to be packed.

Unpack Byte

The subprogram with the call code X"5F" is similar to the pack byte subprogram, except that a 1-byte field is unpacked to form eight 1-byte fields. Each bit of the byte is moved to the corresponding occurrence in the array, so that bit 6 of the original byte is moved to the 6th occurrence within the array. A call to this subprogram has the form:

```
CALL X"5F" USING byte,array
```

where *byte* is a PIC 99 COMP field containing the byte to be unpacked, and *array* is a PIC 99 COMP OCCURS 8 field that contains the unpacked bits.

CRT Screen Handling

The AIX VS COBOL system supports three different formats of the the ACCEPT and DISPLAY statements:

- ANSI COBOL ACCEPT and DISPLAY
- Screen item ACCEPT and DISPLAY
- Data item ACCEPT and DISPLAY

Both the screen and the data item ACCEPT and DISPLAY statements use the COBOL screen handling routine for screen input-output. The ANSI ACCEPT and DISPLAY statements use this routine only if a screen or data item ACCEPT occurred previously. Once this routine has been invoked, all terminal input-output is controlled with it. It changes your terminal mode and automatically clears the screen on the first output operation.

Your AIX VS COBOL system comes with a demonstration program that uses screen handling features. This program is *scdemo1.cbl* in the \$COBDIR/demo directory.

The AIX VS COBOL system also supports calls, designed to be used from a C program, which allow you to mix the output from a C program with ACCEPT and DISPLAY statements.

These features are described in the following sections.

The ACCEPT and DISPLAY Statements

The ANSI COBOL ACCEPT and DISPLAY statements supported by AIX VS COBOL are the standard ANSI ACCEPT and DISPLAY statements, with minor extensions. These statements allow for up to one line of data to be read into memory from the console, and for one line to be displayed, at a time.

The other two formats of the ACCEPT and DISPLAY statements supported by AIX VS COBOL are Micro Focus extensions to the COBOL language, to make it fully interactive. They allow full screens of data to be displayed or accepted into memory using single statements.

The screen item ACCEPT and DISPLAY statements allow you to display non-scrolling forms, which consist of areas of the screen defined in detail in a part of the Data Division named the Screen Section. Data is moved automatically between screen areas and data items.

The data item ACCEPT and DISPLAY statements allow you to display data items, which consist of non-scrolling forms. At run time, data can be entered into these forms. The areas of screen to be used in these statements are defined in the statements themselves.

The *Language Reference* contains detailed specifications of the above formats of the ACCEPT and DISPLAY statements.

Note: It is illegal to try to display control characters.

Display Attributes

Whenever a text character is displayed on the screen, it has an *attribute* (that is, a character or byte of information) associated with it. The way the character is displayed depends upon its *attribute byte*. AIX VS COBOL RTE allows characters to be displayed on the screen with a number of display attributes.

The attributes available to you are dependent on the terminal you are using and the **terminfo** entry for that terminal. They could include high or low intensity, underline, reverse video, or blinking options. If you wish, you can alter the value of the attribute byte and so alter the way characters are displayed on the screen. You can do so by amending either the *screen attribute* or the *user attribute*. You can do this by using the screen control RTE subprograms described in this chapter.

The screen attribute allows you to specify an attribute that is associated with each character position on the screen. You can define areas of the screen as having different attributes. Whenever a character is displayed on the screen, it has the attributes associated with that position.

The user attribute is associated with a whole screen. All of the characters displayed on the screen take that attribute. Once a program has set the user attribute it is enabled through the whole of that run, although another program within the same suite may change the attribute. The user attribute overrides any screen attributes you may have defined.

The Structure of the Attribute Byte

The following list shows the structure of the screen and user attribute byte:

Bit 0	Highlight
Bit 1	Underline
Bit 2	Reverse video
Bit 3	Blink
Bit 4 to 7	Must be set to 0

Each bit indicates one type of attribute. You can set these bits by using the RTE subprograms with call codes X“A7”, X“B7”, and X“B8”. If you wish you can set several bits to give a combination of attributes. The attributes available to you and how they may be combined is dependent on the type of terminal you are using.

Highlighting

By default, the behavior of ACCEPT and DISPLAY operations that use high intensity attributes is as follows:

- Highlighted text appears in high intensity mode for terminals which support a high intensity attribute but no low intensity attribute. This is as specified in **terminfo**.
- Highlighted text appears in normal mode for terminals which support low intensity mode (as specified in **terminfo**) and which use this low intensity mode as the default mode for unhighlighted text.

For more details on **terminfo**, see Chapter 10, “Configuring Your AIX VS COBOL System.” The AIX VS COBOL system assumes that low intensity and high intensity space characters cannot be distinguished from normal spaces, and so the RTE will attempt some optimization because of this. This is particularly effective on terminals which support low intensity mode.

You can change the behavior of the high and low intensity attributes by setting the COBATTR environment variable:

```
COBATTR=n
```

where *n* can be any of the following values:

- 0 Default action, as specified above.
- 1 Always use the **terminfo** high intensity mode for highlighting; never attempt to use low intensity mode.
- 2 High and low intensity space characters are not assumed to be the same as normal mode space characters.
- 3 As for 1 and 2 above.

Screen Handling From C

The RTE subprograms with the call codes X“B7” and X“B8” allow you to perform screen handling operations from COBOL. However, the AIX VS COBOL system also supports some additional screen-handling routines which can be used from C programs. These allow the RTE and run-time support libraries to handle output from both C programs called from COBOL programs, and from ACCEPT/DISPLAY operations performed by the calling COBOL programs. Normally, if any screen-handling is carried out outside of the control of COBOL (for example, under the control of C), COBOL is not aware of the output of that screen-handling operation when control returns to COBOL. The effect of subsequent screen-handling operations could thus be undefined. The routines described below enable you to avoid this problem.

For screen output that uses called C routines and ACCEPT/DISPLAY operations, you must ensure that you explicitly position the cursor before the cobprintf() or DISPLAY statement. In C you use the cobmove() routine and in COBOL you use the DISPLAY...AT statement.

The routines currently supported are:

- `cobmove`
- `cobaddch`
- `cobaddstr`
- `cobaddstrc`
- `cobprintf`
- `cobscroll`
- `cobclear`
- `cobgetch`
- `cobl原因`
- `cobcols`

These routines are described in the following sections.

When using any of these routines, you must include the header file `cobscreen.h`, which is provided with the AIX VS COBOL system software. This file defines the attributes you can use, the type (`cobchtype`), and declares any external functions which the routine needs.

For those routines where attributes are allowed, you can use the bit-wise OR operator to combine any of the attributes defined in the `cobscreen.h` file. These attributes are listed below:

Attribute	Description
<code>A_NORMAL</code>	Normal, no attribute
<code>A_BOLD</code>	Bold or highlight
<code>A_UNDER</code>	Underline
<code>A_REVERSE</code>	Reverse video
<code>A_BLINK</code>	Blink

cobmove Routine

This routine moves the virtual cursor to the specified line and column on the screen. It has the form:

```
void cobmove(y,x)
int y,x;
```

where *y* is the number of the line to which the virtual cursor is to be moved, and *x* is the number of the column to which the virtual cursor is to be moved.

cobaddch Routine

This routine displays the specified character on the screen at the current virtual cursor position. It has the form:

```
void cobaddch(ch)
cobchtype ch;
```

where *ch* is the required character. This may include attributes.

cobaddstr Routine

This routine displays the specified string on the screen at the current virtual cursor position. It has the form:

```
int cobaddstr(s)
cobchtype *s;
```

where *s* is the required string which can be up to 255 characters long. This may include attributes, but cannot include any control characters other than “\n” (newline).

cobaddstrc Routine

This routine displays the specified string on the screen at the current virtual cursor position. It has the form:

```
int cobaddstrc(c)
char *c;
```

where *c* is the required string which can be up to 255 characters long. This can only contain ordinary characters; it cannot include attributes. The routine uses the normal attribute. It cannot include any control characters other than “\n” (newline).

cobprintf Routine

This routine displays the specified formatted string on the screen at the current virtual cursor position. It has the form:

```
int cobprintf(fmt)
char *fmt;
```

where *fmt* is the required string in printf() style, which can be up to 255 characters long in its extended form. This can only contain ordinary characters; it cannot include attributes. It cannot include any control characters other than “\n” (newline).

This routine returns the number of arguments output, or if an error condition arises, it returns the value “-1”.

cobscroll Routine

This routine scrolls the screen display up one line, starting and finishing at the specified lines. It has the form:

```
void cobscroll (top,bot)
int top,bot;
```

where *top* is the first line to be scrolled up one line and *bot* is the last line to be scrolled up one line.

cobclear Routine

This routine clears the screen display and positions the virtual cursor at line 0, column 0. It has the form:

```
void cobclear( )
```

cobgetch Routine

This routine gets a character from the keyboard. It has the form:

```
int cobgetch( )
```

coblins Routine

This routine returns the number of lines on the screen. It has the form:

```
int coblins( )
```

cobcols Routine

This routine returns the number of columns on the screen. It has the form:

```
int cobcols( )
```

Using Escape Sequences to Send Attribute Information to the Screen

AIX VS COBOL will allow the use of raw escape sequences to send attribute information to the screen. This is to support some older programs that may have had no other means of indicating attributes. This method of coding is not recommended for newer code.

In order to be able to use escape sequences to send information to the screen, you must set the COBCTRLCHAR environment variable:

```
COBCTRLCHAR=y  
export COBCTRLCHAR
```

When this environment variable is set, all optimization features of the screen handling module will be bypassed. Without these optimization techniques, screen output will be noticeably slower. Also, if escape sequences are used to handle attributes, some screen management may need to be done by the user programs. Since these raw escape sequences are outside of the AIX VS COBOL screen handling module, the effects created by them are not known to the COBOL screen handling module.

The hardcoding of escape sequences is never recommended for portable, maintainable programs in which a variety of terminals are to be used. The terminfo mechanism for terminal access is fully supported by AIX VS COBOL to allow terminal selection flexibility.

AIX VS COBOL cannot guarantee the future support of hardcoded escape sequence programming, nor guarantee a consistent result for escape sequences or combinations of escape sequences that are hardcoded.

It is recommended that the following syntax be used for attribute handling:

```
SCREEN SECTION.  
01 screen-name.  
   LINE xxx  
   COL  yyy  
   HIGHLIGHT . . .
```

or

```
DISPLAY data-item LINE xxx POSITION yyy REVERSE HIGH . . .
```

To display special graphic characters to the screen, it is recommended that the Screen Input and Output internal RTE subprograms named X“B7” and X“B8” be used. These are documented in “Display Screen Input and Output” on page 9-11.

File Handler

You can create a run time environment (RTE) that is linked to your file handler(s) rather than the default file handler(s) supplied with the AIX VS COBOL system. The default file handlers for the various types of file organization and record format are shown in Table 9-1.

File Type	Default File Handler Fixed-Length Records	Default File Handler Variable-Length Records
line-sequential	lsfile	lsfilev
sequential	sqfile	sqfilev
indexed	ixfile	ixfilev
relative	rlfile	rlfilev
sort	csort	csortv

The file handler(s) you link to the RTE in preference to any of the default file handlers in Table 9-1 must conform to the format of the file handler interface. See “Interface to the COBOL File Handler” on page 9-21 for format information.

To link your own file handler(s) to the RTE, use the **-m** option on the **cob** command line. See Chapter 5, “Compiler Options” for information on this command.

After compiling your new file handler so that it exists as a **.o** file, consider the following examples. This example,

```
ln newix.o newix ◀  
cob -xo rts32 newix -m ixfile=newix ◀
```

creates an RTE that uses the user-defined file handler “newix” for all fixed-length records indexed file operations.

The following example is similar to the previous example, except that “newix” can handle both fixed- and variable-length records.

```
ln newix.o newix ◀  
cob -xo rts32 newix -m ixfile=newix -m ixfilev=newix ◀
```

Note: If you do not specify a mapping for the file handler(s) as illustrated in the above examples, the default file handlers shown in Table 9-1 are those used by the RTE.

Interface to the COBOL File Handler

Any file handler(s) that you link to the RTE in preference to the default file handler(s) must conform to the interface rules given in the rest of this chapter.

The file handler is invoked through a simple call with two parameters.

The first of these parameters describes the action required and consists of two bytes, the first of which is currently always X'FA'. The second byte describes the operation to be performed.

The second parameter is a parameter block, known as a File Control Description (FCD), through which all other relevant information is passed.

The offsets given in the following sections are from the base of this FCD with the first byte having an offset of 0.

For a detailed description of these fields, see "FCD Information Format" on page 9-23. Unless otherwise noted, a reference to a sequential file includes files with the organization LINE-SEQUENTIAL.

Operation Codes Passed in the Second Byte of the First Parameter

Table 9-2 shows a list of operation codes. The value corresponding to the required operation is passed in the second byte of the first parameter to the file handler. The effect of the instruction on the file handler is also listed.

Table 9-2 (Page 1 of 2). Operation Codes Passed in the Second Byte of the First Parameter

Value	Effect of Instruction	File Type
00	OPEN in INPUT mode	Any
01	OPEN in OUTPUT mode	Any
02	OPEN in I-O mode	Any
03	OPEN in EXTEND mode	Any
04	OPEN in INPUT mode with NO REWIND	Sequential
05	OPEN in OUTPUT mode with NO REWIND	Sequential
08	OPEN in INPUT mode REVERSED	Sequential
80	CLOSE	Any
81	CLOSE WITH LOCK	Any
82	CLOSE WITH NO REWIND	Sequential
84	CLOSE REEL/UNIT	Sequential
85	CLOSE REEL/UNIT FOR REMOVAL	Sequential
86	CLOSE REEL/UNIT WITH NO REWIND	Sequential
8C	READ PREVIOUS WITH NO LOCK	Indexed/relative
8D	Sequential READ WITH NO LOCK	Any
8E	Random READ WITH NO LOCK	Indexed/relative
D8	Sequential READ WITH LOCK	Any
D9	Sequential READ WITH KEPT LOCK	Any
DA	Random READ WITH LOCK	Indexed/relative
DB	Random READ WITH KEPT LOCK	Indexed/relative
DC	COMMIT	Any
DD	ROLLBACK	Any
DE	READ PREVIOUS WITH LOCK	Indexed/relative
DF	READ PREVIOUS WITH KEPT LOCK	Indexed/relative
E1	WRITE BEFORE	Sequential

Table 9-2 (Page 2 of 2). Operation Codes Passed in the Second Byte of the First Parameter

Value	Effect of Instruction	File Type
E2	WRITE AFTER	Sequential
E3	WRITE BEFORE TAB	Sequential Only
E4	WRITE AFTER TAB	Sequential Only
E5	WRITE BEFORE PAGE	Sequential Only
E6	WRITE AFTER PAGE	Sequential Only
E8	START with no key value	Indexed/relative
E9	START with key value	Indexed/relative
EB	START with key not less than value	Indexed/relative
EC	WRITE BEFORE mnemonic name	Sequential
ED	WRITE AFTER mnemonic name	Sequential
F2	WRITE AFTER POSITIONING	Sequential
F3	WRITE	Any
F4	REWRITE	Any
F5	Sequential READ	Any
F6	Random READ	Indexed/relative
F7	DELETE	Indexed/relative
F8	DELETE file	Any
F9	READ PREVIOUS	Indexed/relative
FE	START with key less than value	Indexed/relative
FF	START with key less than value or equal	Indexed/relative

Information Passed in the FCD at Open Time

Information passed to the file handler:

File organization	Offset 5
File access mode	Offset 6
File name length	Offset 11
Lock mode flags	Offset 24
Other flags	Offset 25
Maximum record length	Offset 38
Minimum record length	Offset 50
Recording mode (fixed/variable)	Offset 47
File name pointer	Offset 60

For indexed files only:

Key definition block pointer	Offset 64
File format (C-ISAM/level II/current)	Offset 34

Information returned by the file handler:

Status	Offset 0
Handle	Offset 28
Open mode	Offset 7

Information Passed for Other Operations

Information passed to the file handler:

Handle	Offset 28	
Current record length	Offset 48	(WRITE and REWRITE on variable-length files)
Record pointer	Offset 56	

For sequential files only:

Line count Offset 52 (WRITE AFTER *n*)

For relative files only:

Relative key Offset 43 (RANDOM or DYNAMIC
without NEXT)

For indexed files only:

Key identifier Offset 52 (START or random READ)
Key length Offset 54 (START only)

Information returned by the file handler:

Status Offset 0
Current record length Offset 48 (READ on variable-length
files)

For relative files only:

Relative key Offset 43 (~~SEQUENTIAL~~ access or
READ NEXT)

FCD Information Format

The following figure shows the offset, size, and description of the FCD information formats.

Offset	Size	Description
0	1	First user status byte -- values as defined by ANSI
1	1	Second user status byte -- values as defined by ANSI unless the first status byte=9
2	2	Reserved
4	1	Reserved
5	1	File organization indicator: 0 = Line-sequential 1 = Sequential 2 = Indexed 3 = Relative
6	1	Passed at OPEN time User status indicator and access mode: 128 = User has declared a status field 0 = Sequential access mode 4 = Random access mode 8 = Dynamic access mode Passed at OPEN time

Offset	Size	Description
7	1	File open mode (set at OPEN and CLOSE time): 128 = Closed (initial state) 0 = Open input 1 = Open output 2 = Open I-O 3 = Open extend
8	3	Reserved
11	2	File name length Passed at OPEN time
13	11	Reserved
24	1	Lock mode flags for shareable files: Bit 7 Set if lock on multiple records Bit 6 Set if WRITELOCK enabled Bit 5 Reserved Bit 4 Reserved Bit 3 Reserved Bit 2 Set if lock mode MANUAL Bit 1 Set if lock mode AUTOMATIC Bit 0 Set if lock mode EXCLUSIVE Passed at OPEN time
25	1	Other flags: Bit 7 Set if OPTIONAL file (open input) Bit 6 Reserved Bit 5 Set if NOT OPTIONAL (open I-O and extend) Bit 4 Set if file name is EXTERNAL Bit 3 Reserved Bit 2 Reserved Bit 1 Set if MULTIPLE REEL file (sequential only) Bit 0 Set if LINE ADVANCING file (sequential only) Passed at OPEN time
26	2	Reserved
28	4	Handle
32	1	Reserved
33	1	Flags High order bit 'x0000000' indicates ANSI behavior 0 = ANS74 1 = ANS85 Passed at OPEN time
34	1	File format type 0 = current 1 = c-isam 2 = level II V2 Passed at OPEN time
35	3	Reserved
38	2	Maximum record length Passed at OPEN time
40	3	Reserved
43	4	Relative record number

Offset	Size	Description
47	1	Recording mode: 0 = Fixed 1 = Variable Passed at OPEN time
48	2	Current record length
50	2	Minimum record length Passed at OPEN time
52	2	Key identifier (indexed files) Line count (line sequential files)
54	2	Effective key length (used only with START on indexed files)
56	4	Pointer to record area
60	4	Pointer to file name Passed at OPEN time
64	4	Pointer to key definition area Passed at OPEN time
68	32	Reserved

Key Definitions for Indexed Files

All key definitions come immediately after the global information and before any component definitions. They contain pointers to the relevant component definitions in the form of offsets from the structure base.

Global Information

2 bytes	Length of key definition block
1 byte	Reserved for version number
1 byte	Reserved for index format
1 byte	Reserved for integrity level
1 byte	Reserved for tuning flags
2 bytes	Number of keys
2 bytes	Reserved for reserved index areas
4 bytes	Reserved for index record length

Key Definitions

2 bytes	Component count
2 bytes	Offset to first component definition for this key
1 byte	Key flags
bit 7	Reserved
bit 6	Reserved
bit 5	Reserved
bit 4	Set if prime key
bit 3	Reserved
bit 2	Reserved
bit 1	Set if sparse key
bit 0	Set if password supplied

1 byte	Compression flags
	bit 7 Reserved
	bit 6 Reserved
	bit 5 Reserved
	bit 4 Reserved
	bit 3 Reserved
	bit 2 Set if compression of trailing spaces
	bit 1 Set if compression of identical leading characters
	bit 0 Set if compression of following duplicates
1 byte	Sparse character (key suppressed if whole key contains this value)
1 byte	Reserved
8 bytes	Password

Component Definitions

1 byte	Component flags (reserved - currently 0)
1 byte	Component type (reserved - currently 0)
4 bytes	Component offset
4 bytes	Component length

CISAM Features


The RTE uses a version of CISAM that has been modified to meet the needs of AIX VS COBOL record locking requirements. This version of CISAM is embedded in the `mfisam.o` module, found in the COBOL library `libcobol.a`. You cannot access or use this modified version of CISAM except from a COBOL program that uses the standard COBOL file-handling syntax.

If you have access to a standard version of CISAM, you may wish to use that in place of the modified version supplied with your AIX VS COBOL system. An object module, `cixfile.o`, is supplied with AIX VS COBOL in the archive `libcobol.a`. This provides an interface between the AIX VS COBOL system and the standard CISAM. In order to link the standard CISAM with a COBOL program, use the `+I` flag on the `cob` command line to specify that `cob` is to use the standard CISAM library (which is probably named `libisam.a`). You must also specify the `-m` flag on the `cob` command line to map the symbol "ixfile" onto the CISAM interface "cixfile". `cixfile.o` contains the necessary external references to the CISAM libraries to ensure that they are included in the resulting executable file, in preference to the AIX VS COBOL modified libraries.

Note that you must specify the `+I` flag on the `cob` command line. If you specify the normal library inclusion flag `-I`, the standard CISAM libraries are not included in the executable file.

You can also build a version of animator that has your own version of CISAM linked to it. In this case, you cannot animate your programs using the `anim` command; you must run them with the `+A` run-time switch set. See "Run-Time Switches" on page 7-7 for details on the run-time switches.

Consider the following examples:

- `cob -x prog1.cbl prog2.cbl prog3.cbl` 

compiles and links `prog1.cbl`, `prog2.cbl`, and `prog3.cbl` with the modified version of CISAM supplied with your AIX VS COBOL system in `libcobol.a`.

-
- `cob -x prog1.cbl prog2.cbl prog3.cbl -m ixfile=cixfile +lisam` \leftarrow
compiles and links `prog1.cbl`, `prog2.cbl`, and `prog3.cbl` with the standard version of CISAM found in `libisam.a`.
 - `cob -xo rts32 -e "" -m ixfile=cixfile +lisam` \leftarrow
outputs an RTE that you can use to run intermediate and unlinked native code files with the standard version of CISAM. For example, to run the intermediate code file `prog1.int` using the standard CISAM libraries, enter:
`rts32 prog1.int` \leftarrow

You need to be aware of the following differences between the standard CISAM libraries and the modified version supplied with the AIX VS COBOL system.

- The standard CISAM does not support `WRITE` and `REWRITE` operations acquiring locks, but the modified CISAM does.
- The standard CISAM does not support the `READ WITH NO LOCK` statement; it treats this as a normal `READ` operation. However, the modified CISAM does support this operation.
- The standard CISAM does not support the creation of data and index files in separate directories, but the modified CISAM does. Therefore, the `I` option of `COBCAP` and the `&` option of logical filename mapping using environment variables will not work with the standard CISAM.
- The modified CISAM has a maximum record length of 8 Kbytes. The standard CISAM imposes a lower maximum record length. Check your CISAM documentation for specific details.

Chapter 10. Configuring Your AIX VS COBOL System

Contents

About This Chapter	10-3
Introduction	10-4
terminfo	10-5
cobkeymp	10-5
ADISCTRL	10-5
Keyboard Conversion Process	10-5
keybcf Utility	10-6
Specifying and Accessing Multiple or Alternate cobkeymp Files	10-7
Invoking the keybcf Utility	10-8
Using the keybcf Utility	10-9
Maximum Size of keybcf Buffers	10-14
adiscf Utility	10-14
Invoking the adiscf Utility	10-14
Using the adiscf Utility	10-14

About This Chapter

Your IBM AIX VS COBOL software is supplied with the default configuration tailored for your system. This chapter describes the information you must have to alter the default behavior of the AIX VS COBOL extensions to the ACCEPT and DISPLAY statements as described in the *Language Reference*. Do not attempt to alter the default behavior of the AIX VS COBOL extensions to the ACCEPT and DISPLAY statements unless you have system administrator authority.

Introduction

The Run Time Environment (RTE) module that provides extended ACCEPT and DISPLAY facilities is called ADIS. The configuration information needed before these facilities can function correctly is provided in the following three databases:

- terminfo
- cobkeymp
- ADISCTRL.

The information provided in these three databases allows ADIS to translate the machine-specific character codes it receives from your keyboard into a terminal independent code. ADIS then maps this terminal independent code to AIX VS COBOL system program code.

terminfo

The **terminfo** database is the AIX terminal description database. It provides the RTE with information concerning the terminal you are using. You must ensure that an entry exists within **terminfo** for your particular terminal, and that the **TERM** environment variable is set to the name of that terminal. See the AIX Operating System documentation for a full description of **terminfo**.

cobkeymp

Entries in the **cobkeymp** database map control characters and terminfo codes onto a standard set of function keys that the ADIS ACCEPT/DISPLAY module can recognize. You can set up your own **cobkeymp** database using the **keybcf** utility. See “keybcf Utility” on page 10-6 for further details. If you do not set up your own **cobkeymp** database, the AIX VS COBOL system uses a set of internal defaults for its functions.

ADISCTRL

The ADISCTRL database is the configuration file for the AIX VS COBOL ADIS ACCEPT/DISPLAY module. It specifies the editing functions to be used by the ADIS function keys. You can alter the entries in this database using the **adiscf** utility. See “adiscf Utility” on page 10-14 for further details.

Information held in both the **terminfo** and the **cobkeymp** databases is terminal specific. Both define, to the ADIS ACCEPT/DISPLAY module, the hexadecimal sequences a terminal sends when any particular key is pressed.

The information held in the ADISCTRL database is terminal independent. It defines, to the ADIS ACCEPT/DISPLAY module, the action to be taken when a keystroke is recognized as a function key.

Keyboard Conversion Process

Figure 10-1 on page 10-6 shows how ADIS translates the machine-specific character codes it receives from your keyboard into editing commands.

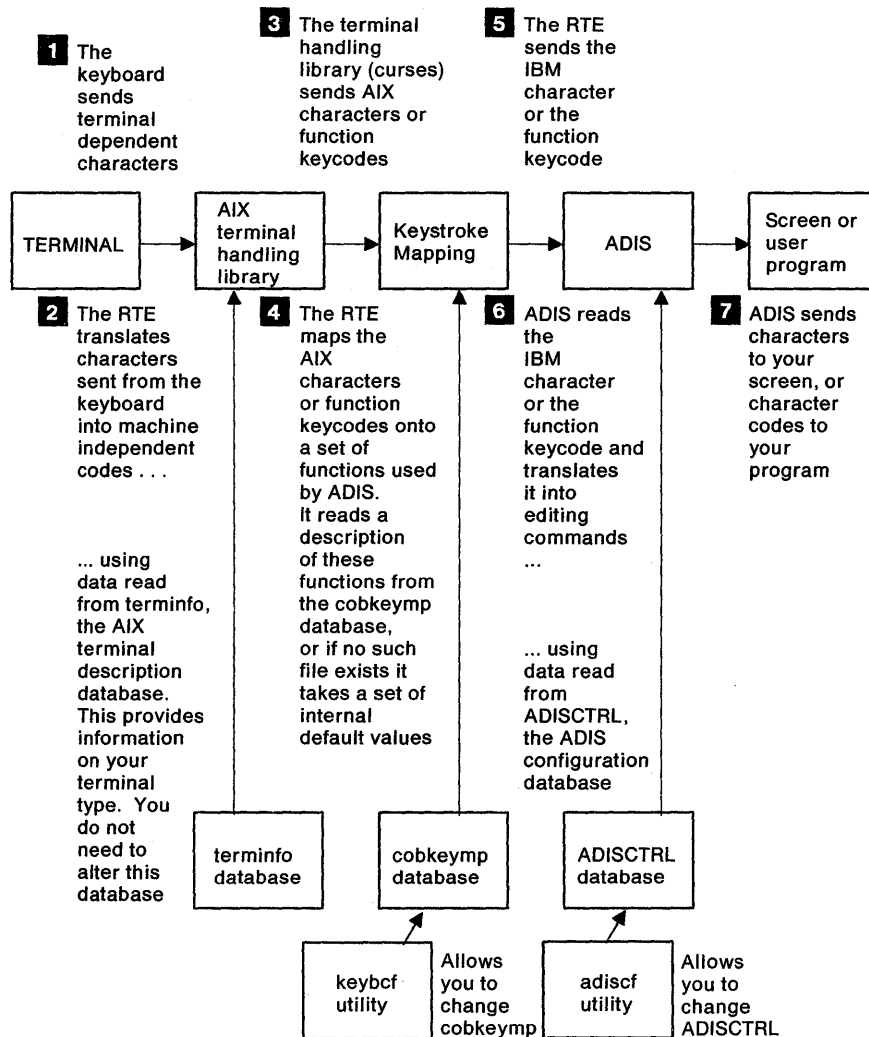


Figure 10-1. Character Conversion Process

keybcf Utility

A set of internal default values for how the RTE interprets control characters and keystroke mnemonics is provided with your AIX VS COBOL system software. These default values are contained in the **cobkeymp** file. See Table 10-1 on page 10-7 for a list of these defaults.

To change any of the default values, use the **keybcf** utility to set up your own **cobkeymp** file. This contains your own set of keystrokes that you wish to perform for each function.

You do not need to create a different **cobkeymp** file for each terminal type on your system. Instead, try to ensure that your **cobkeymp** file is suitable for as many different types of terminals as possible. All terminal-specific decoding is done by the RTE using the **terminfo** database. However, you may have to supply alternative keystrokes for terminals not having special function keys.

If you wish to edit the **cobkeymp** file located in the \$COBDIR directory, you must have superuser authority to update this file. It is recommended that you make a copy of the original **cobkeymp** file for safe keeping prior to editing this file.

In addition to the COBOL **keybcf** mapping of keys, the AIX VS COBOL system by default does not map a carriage return (hex 0D) to a newline (hex 0A) on input. Earlier versions of AIX VS COBOL by default did map a carriage return to a newline. If you do not use the AIX system default **terminfo** files, then you need to make sure your **terminfo** files are correct based on the new default.

Specifying and Accessing Multiple or Alternate **cobkeymp** Files

You can create multiple **cobkeymp** files. By default, a user-specified **cobkeymp** file must exist in the directory where it will be used for the running of the user program. To use a **cobkeymp** file from another directory to run a program, you must use the **dd_style** file name mapping as described in “File Name Mapping” on page 3-7. This method can also be used to select from several different **cobkeymp** files that you have defined.

For example, you could create two **cobkeymp** files:

```
/u/test/cobkeymp
```

and

```
/u/other/test/cobkeymp
```

Then, if you want to run a program “mycode.cbl”, you could do the following:

```
cob mycode.cbl
dd_cobkeymp=/u/test/cobkeymp
export dd_cobkeymp
cobrun mycode.int
```

This would find and use the /u/test/cobkeymp keyboard configuration.

Table 10-1 (Page 1 of 2). Default ADIS Control Keys		
ADIS Function Key Number	Meaning	RT Default Key
00	Terminate Accept	Enter Keyboard Enter Keypad Ctrl+M Ctrl+J
01	Terminate Program	Ctrl+D
02	Carriage Return	
03	Cursor Left	Left arrow
04	Cursor Right	Right Arrow
05	Cursor Up	Up arrow
06	Cursor Down	Down Arrow
07	Move to start of screen	Home
08	Move to next tab stop	undefined
09	Move to previous tab stop	undefined

Table 10-1 (Page 2 of 2). Default ADIS Control Keys		
ADIS Function Key Number	Meaning	RT Default Key
10	Move to end of screen	End Ctrl + O
11	Move to next field	Ctrl + N Tab Ctrl + I
12	Move to previous field	Ctrl + P Shift Tab Ctrl + L
13	Change case of character	Ctrl + F
14	Rubout character	Backspace shift + Backspace Ctrl + H
15	Retype rubout character	Ctrl + R
16	Insert single character	Insert shift + Insert
17	Delete character	Delete
18	Restore deleted character	Ctrl + U
19	Clear to end of field	Ctrl + Delete
20	Clear Field	Ctrl + X
21	Clear to end of screen	Ctrl + End
22	Clear screen	Ctrl + Home
23	Set Insert mode	Ctrl + Insert
24	Set Replace mode	undefined
25	Reset field to its original value	Ctrl + A
26	Move to start of field	Ctrl + W

Invoking the keybcf Utility

To set up your own **cobkeymp** file using the **keybcf** utility, type the command line:

```
keybcf ◀
```

The RTE searches for a **cobkeymp** file, first in the current directory and then in the COBOL system directory, \$COBDIR. If RTE finds a **cobkeymp** file, it uses the values given there. However, if RTE does not find a **cobkeymp** file, it uses the default values. If a **cobkeymp** file already exists, you are asked if you want to edit it.

In order to change the system **cobkeymp** file (that is, the one located in \$COBDIR), you must have superuser authority.

Using the keybcf Utility

keybcf is menu-driven. Once **keybcf** is invoked, it displays the initial menu, as shown in Figure 10-2.

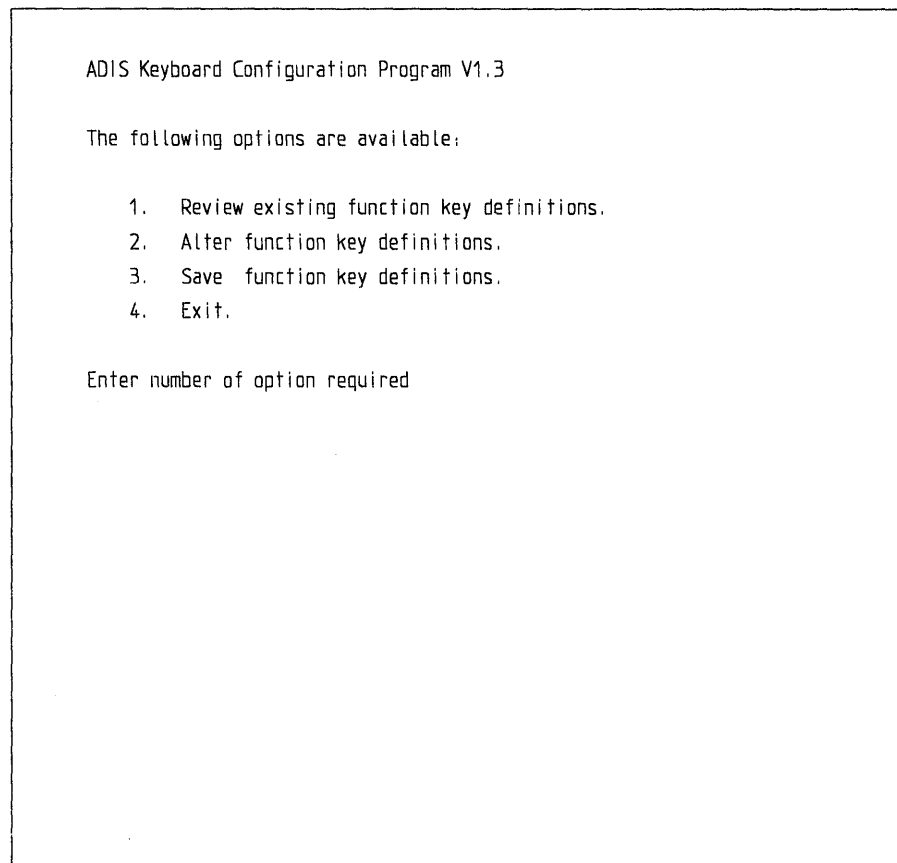


Figure 10-2. Main keybcf Display Screen

To select the option of your choice, press its associated number. The submenu for that function is then displayed.

Review Existing Function Key Definitions

Enter a 1 on the main **keybcf** display screen to see a submenu in which you are prompted to choose the set of function keys you want to review. You can review the following currently defined lists:

- ADIS function key list
- ANIMATOR function key list
- User function key list
- Compatibility function key list.

The ADIS function key list defines the keys that carry out specified functions when you are executing a COBOL program in ACCEPT mode. ANIMATOR and COBOL system programs use the ANIMATOR function key list. It describes the mappings of control and terminfo codes onto various special operation keys. The user function key list defines the mapping of control and terminfo codes onto user function keys. During an ACCEPT operation, the user function key table is searched before the ADIS function key table is searched. The user function keys are initially enabled. To disable these function keys during the execution of a program, you must first disable them by CALLing the X"AF" subprogram. See Chapter 9, "Advanced Programming Features" for more information.

The compatibility function key list defines the mapping of control and terminfo codes onto user keys defined in a dialect of COBOL other than AIX VS COBOL. If you want function keys to return values that are compatible with other dialects of COBOL, alter the compatibility function key list rather than the user function key list. By default, the compatibility function key list is configured for compatibility with Ryan McFarland COBOL Version 2.0, under UNIX.

The CRT STATUS clause allows you to ascertain which function key was used to end an ACCEPT operation. See the *Language Reference* for information on how to use this clause.

To select an option, press its associated number. Once you have entered the number of the list you want to review, the hexadecimal values of all the currently defined function keys in that list are displayed. Press any key to move from one display screen to the next. At the end of the list, press any key to return to the review submenu.

Alter Function Key Definitions

Enter a **2** on the main **keybcf** menu to see a submenu from which you can choose a set of function keys to alter. As with the review submenu, you can select the ADIS, ANIMATOR, user, or compatibility key lists. Figure 10-3 on page 10-11 shows the format of the display screen as it appears after you have made your selection.

```
ADIS/ANIMATOR/User/Compatibility Function Key List

Function          nn

Enter required key sequence:

I=Insert, D=Delete, X=Hexadecimal Input, Space=Skip, Q=Quit
```

Figure 10-3. Alter Function Key Options

Note: *nn* is the hexadecimal value of the key assigned to that function.

Each function in the function key list you selected is displayed individually, as shown in Figure 10-3. To cycle from one function to the next without altering the key defined for each function, press the Space bar. To replace any keys currently defined in one of the lists, press the new key(s) that will perform that function. Any keys used (pressed) must be defined in **terminfo**. Once you have entered the new keys, the program automatically cycles to the next function. There is a pause before the program cycles to the next function. To retain a currently defined function key, and add another key to perform the same function, press **I** before you enter the required key. If you want to delete a defined function key from a list, press **D**; **keybcf** automatically cycles to the next function.

To enter new function keys, do either of the following:

- Press the actual key(s) you want to perform a certain function.
- Enter the hexadecimal sequence for the key(s).

Normally, you enter new function keys by pressing the actual key to which the function is to be assigned. However, you may need to define keys that are not on the keyboard you are using but are available on the one on which your program will run. You can do this by entering the hexadecimal sequences for the keys you want to define.

To enter a hexadecimal sequence, press **X**. The word **Hex** appears at the bottom right of your display screen, indicating that the program is expecting hexadecimal input. If you enter an invalid hexadecimal sequence, you receive an error when you try to cycle to the next function and are prompted to enter a valid sequence.

To determine valid hexadecimal sequences, you can refer to the AIX Operating System documentation, to the **terminfo** file on your system, and to Table 10-2. The AIX Operating System documentation regarding keyboards gives the strings that are returned by each of the possible key states. If the returned string for a particular key state is a hexadecimal value, you can directly enter that value as the hexadecimal sequence representing the key you wish to define.

However, if the returned string begins with an **ESC**, refer to your **terminfo** source file, normally found in the **/usr/lib/terminfo** directory. The standard IBM **terminfo** source file is named **ibm.ti**. For a key state that has a returned string starting with an **ESC** to be valid, the key state needs to be related to a capability in your **terminfo** file, and the referenced capability needs to be one that is defined in Table 10-2.

The **terminfo** source file relates capabilities to particular key states. To determine if a particular key state represents a valid hexadecimal sequence for **keybcf**, locate your terminal type and then the particular **ESC** sequence in your **terminfo** source file. Note that the short capability name appears to the left of the equals sign. If this short capability name appears in Table 10-2, the **FF XX YY** number associated with this name is a valid hexadecimal sequence and can be entered as the required key sequence.

For example, suppose you want to use the left arrow key (key number 79) to carry out the cursor left function. First, refer to the information on keyboards in the AIX Operating System documentation to find that key 79 returns the string **type** and note that **kcub1** is the short capability name equated to this escape sequence. Next, look in Table 10-2 and find that **kcub1**, **key_left**, gives the valid key sequence **FF 01 04**. Then enter this hexadecimal sequence in **keybcf**.

Alternatively, if you press the left arrow key on your keyboard, the system enters these three bytes for you.

You can return to the Alter submenu at any time by pressing **Q**. Entering **5** on this submenu returns you to the main **keybcf** menu.

Table 10-2 (Page 1 of 2). Hexadecimal Sequences for Key Functions Not on Your Keyboard		
Capability Name	Variable	Key Sequence
kcud1	KEY_DOWN	FF 01 02
kcuul	KEY_UP	FF 01 03
kcub1	KEY_LEFT	FF 01 04
kcuf1	KEY_RIGHT	FF 01 05
khome	KEY_HOME	FF 01 06
kbs	KEY_BACKSPACE	FF 01 07
kf0	KEY_F0	1B

Table 10-2 (Page 2 of 2). Hexadecimal Sequences for Key Functions Not on Your Keyboard		
Capability Name	Variable	Key Sequence
kf1	KEY_F1	FF 01 81
kf10	KEY_F10	FF 01 8A
kf2	KEY_F2	FF 01 82
kf3	KEY_F3	FF 01 83
kf4	KEY_F4	FF 01 84
kf5	KEY_F5	FF 01 85
kf6	KEY_F6	FF 01 86
kf7	KEY_F7	FF 01 87
kf8	KEY_F8	FF 01 88
kf9	KEY_F9	FF 01 89
kd11	KEY_DL	FF 01 08
ki11	KEY_IL	FF 01 09
kdch1	KEY_DC	FF 01 0A
kich1	KEY_IC	FF 01 0B
krmir	KEY_EIC	FF 01 0C
kclr	KEY_CLEAR	FF 01 0D
ked	KEY_EOS	FF 01 0E
kel	KEY_EOL	FF 01 0F
kind	KEY_SF	FF 01 10
kri	KEY_SR	FF 01 11
knp	KEY_NPAGE	FF 01 12
kpp	KEY_PPAGE	FF 01 13
khst	KEY_STAB	FF 01 14
kctab	KEY_CTAB	FF 01 15
ktbc	KEY_CATAB	FF 01 16
k11	KEY_LL	FF 01 1B

Save Function Key Definitions

To save any alterations you have made to any of the function lists, press **3** on the main **keybcf** menu. This saves the amended function lists in a **cobkeymp** file.

Exit

Press **4** on the main **keybcf** menu to return to the main AIX VS COBOL system.

Maximum Size of keybcf Buffers

The **keybcf** buffers hold the key definitions for all four key lists: the ADIS key list, the Animator key list, the user key list, and the compatibility key list. The total size of all the keys defined must not exceed 768 bytes.

The compatibility key list is an alternative user key list, which can be selected using **adiscf**. The intended use of this alternative user key list is for compatibility with other COBOL dialects, such as RM.

If you are unlikely to use this alternative key list, you could delete the key definitions, freeing space to define the keys in the user key list. This would give you more space to define keys if you need it.

adiscf Utility

ADISCTRL is the configuration database for the ADIS module. It can hold up to a maximum of 16 configurations, any of which are available to you. An entry at the start of the ADISCTRL file determines which configuration ADIS uses.

You can alter any of the configurations held in the ADISCTRL database using the configuration utility **adiscf**. This program is designed around a hierarchy of menus. These menus appear at the bottom of your display screen and list the options available to you at any time. You select the option you require by pressing a single key (often a function key) on your keyboard.

Invoking the adiscf Utility

To invoke **adiscf**, enter the command line:

```
adiscf ↵
```

The RTE searches for an ADISCTRL database, first in the current directory and then in the COBOL system directory, \$COBDIR. If one exists, **adiscf** reads in the configuration currently selected for use by ADIS.

If you want to alter the file, first copy it to your own directory. Otherwise, any alterations you make will affect the environments of all of the users on your system. If an ADISCTRL file does not exist, the following message is shown on your display screen:

```
ADISCTRL does not exist - Defaults used
```

adiscf has a set of default values built into it. These are used if ADISCTRL does not exist.

Using the adiscf Utility

Once you have invoked **adiscf**, the initial menu is displayed, as shown in Figure 10-4 on page 10-15.

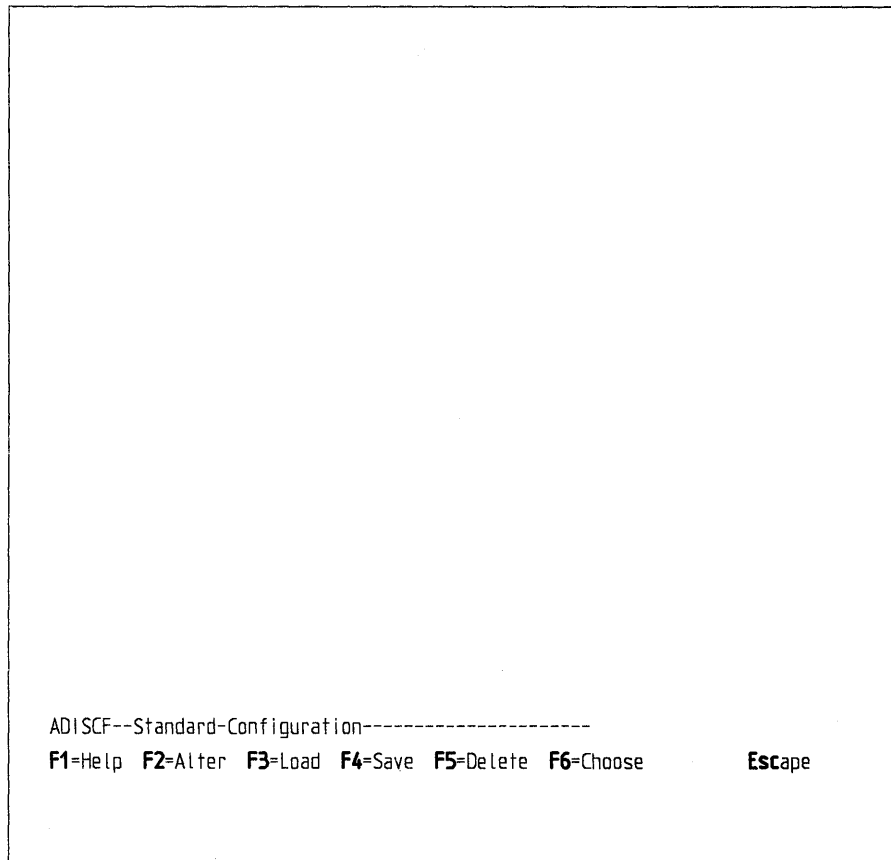


Figure 10-4. Main adiscf Command Menu

The line above the line showing available menu options lists information that identifies the menu you are on and the configuration currently loaded (the standard configuration in this case). Each menu contains a similar *information line*.

To select the option you require from the main menu, press the relevant function key or letter:

- F1 or H** Help facility (available on each menu). It shows a display screen with information on the facilities of the current menu.
- F2 or A** Alters the currently loaded configuration file.
- F3 or L** Loads a particular configuration file from the ADISCTRL database into memory. You must load a configuration before you can alter it.
- F4 or S** Saves the new configuration file you have written in the ADISCTRL database.
- F5 or D** Deletes a configuration from the ADISCTRL database.
- F6 or C** Chooses the configuration to be used by ADIS.
- Escape** Escapes from the **adiscf** utility and returns to the AIX VS COBOL system. If you have changed a configuration file since it was last saved, you are asked to confirm that you wish to leave the **adiscf** program without saving your changes in the ADISCTRL database.

Once you select an option, **adiscf** displays a submenu for that particular option. Most of these menus have the following form:

ADISCF--*Name*-----*Value*-----

where *Name* is the name of the particular submenu (for example, Alter-CRT-Under-Highlighting), and *Value* is the value currently selected for that feature (for example, Underline).

The following sections describe the submenus available to you when you select one of the options from the main menu.

Alter Option

Select the **Alter** option by pressing **F2** or **A** on the Main Configuration menu to alter all, or part, of a configuration. By pressing the appropriate function key on the Alter Configuration menu, you can change the following:

- The way your programs display text when you use the CRT-UNDER phrase
- One or all of the ACCEPT/DISPLAY options
- TAB stops
- Indicator texts
- Message texts
- Positions at which indicator and message texts are displayed
- ADIS key mappings.

The following sections give details on how you can change these features. See the *Language Reference* for a description of free-format and fixed-format fields.

Altering CRT-UNDER-HIGHLIGHTING Options

Pressing **F2** or **C** on the Alter Configuration menu displays a submenu that allows you to alter the type of highlighting used with the DISPLAY...UPON CRT-UNDER statement, the DISPLAY...WITH UNDERLINE statement, or when the UNDERLINE clause is used in the display screen section.

The following list gives the key you press to select each option:

F2 or I	Intensity; text appears bold
F3 or U	Underscore (the default); text is underlined
F4 or R	Reverse video; text appears in reverse video
F5 or B	Blink; text appears blinking.

Press **Escape** to return to the Alter Configuration menu. Some terminals do not support bold and blink. Your particular terminal may not function with these options.

Altering ACCEPT-DISPLAY Options

Pressing **F3** or **A** on the Alter Configuration menu displays a submenu that allows you to specify how you want the cursor to behave, and what you want the fields to look like during an ACCEPT operation. You can choose to alter individual options or all the available options.

Press **F2** or **A** to alter all of the ACCEPT/DISPLAY options. A description of each option and its current value is displayed, with one option per display. Press **↵** to move from one display screen to the next. To change the value of an option, type the number of your choice at the relevant prompt on each display screen. You can alter as few or as many options as you wish.

Press **F3** or **I** to alter individual ACCEPT/DISPLAY options. This displays a list of the available options with a number next to each. Press **F2** or **N** to toggle to the next page of options. Type the number of the option you want to alter, or move the cursor to the relevant line, either by using the keys configured to move the cursor up or down a line, or by typing **/U** to move the cursor up or **/D** to move the cursor down. Press **↵** to select the option you want. A description of that option and its current value is then displayed. To alter the value of an option, type the number of your choice at the prompt and press **↵**.

The following is a list of the ACCEPT/DISPLAY options you can change:

- **User function keys enable/disable**

Allows you to disable or enable the user function keys. These are usually the function keys on your keyboard. You can choose the following:

1. Disables all user function keys. If you press a user function key during an ACCEPT, it is treated as an invalid key.
2. The default. Enables all user function keys. If you press a user function key during an ACCEPT, the ACCEPT is terminated.

- **Range of data keys accepted**

Allows you to specify which characters are to be allowed during input to an ACCEPT. You are prompted to enter the number of the option you require.

1. Characters with ASCII codes in the range 0 to 127 are allowed.
2. Characters with ASCII codes in the range 0 to 255 are allowed.
3. Characters with ASCII codes in the range 32 to 127 are allowed.
4. The default. Characters with ASCII codes in the range 32 to 255 are allowed.

Note: Even if you enable characters within the range 0 to 31 (that is, you choose either option 1 or 2), you may still not be able to enter some of these characters into a field. This is because some of these characters may form the start of some key sequences generated by function or cursor keys. If these keys are enabled, they have priority over the data keys.

- **Prompt character**

Allows you to specify the character to be displayed in the empty part of the field during an ACCEPT. The system displays the selected character in all portions of the field into which you have not yet entered data. The selected character also indicates the extent of the field. This prompt character is used for all picture types except PIC G.

- **Prompt character used in PIC N fields**

Allows you to specify the character to be displayed in the empty part of a PIC G field during an ACCEPT. (PIC N is the same as PIC G in AIX VS COBOL. PIC N was part of an alternate implementation of DBCS support.)

- **Pre-display of fields before an ACCEPT**

Allows you to specify whether you want the contents of data fields to be displayed automatically before an ACCEPT statement. If you do not specify automatic display of data fields before an ACCEPT statement, the display screen remains as it is. You can choose between the following options:

1. Pre-display of numeric-edited fields with numeric editing enabled (when the cursor moves into them). No other pre-display occurs.
2. Pre-display of numeric fields with numeric editing enabled (when the cursor moves into them). No other pre-display occurs.
3. Pre-display of all fields immediately before data is accepted into the field.
4. The default. Pre-display of all fields before any data entry is allowed.

- **ACCEPT in a SECURE field**

Allows you to specify how you want the cursor to behave, and what you want the field to look like, during an ACCEPT into a SECURE field. Possible options are as follows:

1. The default. No character display is shown on the display screen as each character is entered, but the cursor advances to the next character position.
2. An asterisk (*) is displayed as each character is entered, and the cursor advances to the next character position.
3. A space is displayed as each character is entered, and the cursor advances to the next character position.

- **Auto-skip between fields**

Allows you to specify whether you want the cursor to move to the next field automatically when the current field is full. This applies only to multiple data fields within one ACCEPT statement. The available options are as follows:

1. No auto-skip. You must press an explicit field-tab or cursor key (other than rubout) to move to the next field.
2. The default. Auto-skip enabled. If the current field is full, any cursor movement, or pressing a character key, causes the cursor to move to the next field.

Note: This option has no effect on display screen section ACCEPT operations. Auto-skip is off by default for these operations. You can turn this option on by specifying the AUTO phrase in your source program.

- **Termination of an ACCEPT**

Allows you to specify which actions terminate an ACCEPT, as follows:

1. The default. Pressing the **terminate accept** key.
2. Pressing the **next field** key when the cursor is in the last field of an accept.
3. Typing or retyping a data character in the last available character position of an accept (provided auto-skip between fields is enabled).

Note: This option controls only normal termination. Function keys still terminate an ACCEPT, if they are enabled.

- **Validation control if ACCEPT is terminated by a function key**

Allows you to specify whether validation clauses must be satisfied, when terminating an ACCEPT using a function key. You can choose between the following:

1. The default. No validation takes place.
2. Normal validation criteria must be satisfied for the current field.

- **End of field effects**

Allows you to specify how you want the cursor to behave, if you attempt to type data once a field is full. You can select one of the following options:

1. The cursor moves beyond the end of the field, and overtyping is rejected.
2. The cursor stays at the end of the field, and overtyping is rejected.
3. The default. The cursor stays at the end of the field, and overtyping is allowed.

- **Field overflow buffers enable/disable**

Allows you to specify whether data is to be saved in an overflow buffer when displaced from the end of a field.

1. The default. Displaced data is saved in an overflow buffer.
2. Displaced data is not saved in an overflow buffer.

- **Auto-restore during rubout in replacement editing mode**

Allows you to specify the action of the rubout key in free-format fields, when in replacement editing mode.

1. The default. Auto-restore is enabled. Previously overtyped characters are restored, as characters are deleted.
2. Auto-restore is disabled. Deleted characters are replaced by the filler character.

- **Accepts into numeric-edited fields**

Allows you to specify how you want numeric-edited fields to look during an ACCEPT, as follows:

1. Input is accepted as for alphanumeric fields and is normalized to remove illegal characters on exit from the field.
2. Same as for option 1; however, you can enter only digits, signs, decimal points, or commas.
3. The default. Fields up to 32 characters long are accepted in formatted mode. Characters other than digits, signs, and decimal points are rejected. Fields are reformatted to show the editing symbols as data is entered. Fields longer than 32 characters are accepted as for option 1.
4. Same as for option 3, with the exception that fields longer than 32 characters are accepted as for option 2.

- **Accepts into nonedited numeric fields**

Allows you to specify how you want nonedited numeric fields to look during an ACCEPT, as follows:

1. The default. Unsigned and embedded signed nonedited numeric fields with a V in their PIC clauses are treated as though they were a PIC 9(m) field followed by a PIC 9(n) field. Fields with separate signs are treated as though they were PIC S9(m + n).
2. Same as for option 1 except that fields with a V in the PIC clause are treated as PIC S9(m + n).
3. All nonedited numeric fields are treated as alphanumeric fields.

Note: If you specify a nonzero SIZE clause, all nonedited numeric fields are treated as free-format fields, regardless of the setting of this option.

- **Enable/Disable auto-clear or pre-clear**

Allows you to specify how you want a field to appear when the cursor first enters it, as follows:

1. The default. No pre-clear or auto-clear takes place.
2. Pre-clear mode. The field is cleared to spaces or zeroes. Pressing the **Undo** key restores the original contents of the field.
3. Auto-clear mode. If the first keystroke made after the cursor enters a new field is a valid data character, the field is cleared to spaces or zeroes before processing the character. An invalid data character turns auto-clear mode off. Press the **Undo** key to restore the original contents of the field.
4. Same as for option 2, except that pressing the **Undo** key does not restore the original contents of the field.

- **Force a field to be updated if it is not altered**

Allows you to specify how you want the contents of a field to appear if you leave the field without altering it, as follows:

1. The default. The data item is not updated if the field is not altered.
2. The data item is always updated, even if the field is not altered. This option has effect only under either of the following conditions:
 - The field is numeric or numeric-edited and the original data item did not contain any numeric data
 - The field is right justified and the original contents of the field were not.

- **Note end of field**

Allows you to specify if ADIS notes the position of the last character entered into a field:

1. The default. ADIS does not note the position of the last character.
2. ADIS notes the position of the last character entered into a field during an ACCEPT.

This option applies only if the prompt character is disabled.

- **RM/COBOL-style numeric data entry**

Allows you to specify if you want your system to emulate RM/COBOL-style entry of numeric data items, as follows:

1. The default. The standard IBM entry of numeric data items is enabled.
2. The RM/COBOL style of numeric and numeric-edited data entry is enabled.

- **Restrict maximum size of a field**

Allows you to specify if the size of ACCEPT fields is restricted to one line, as follows:

1. The default. Fields are not restricted to one line.
2. Fields are restricted to one line.

- **Control cursor positioning after an ACCEPT**

Allows you to control where the cursor is placed at the end of an ACCEPT operation, as follows:

1. The default. The cursor is moved to the next character position following the end of the current field.
2. The cursor is left at its current position.

- **Control behavior of UPDATE phrases**

Allows you to specify if a CONVERT clause is implied if you specify an UPDATE phrase, as follows:

1. The default. The CONVERT clause is not implied.
2. The CONVERT clause is implied.

This option is provided to enable you to emulate the behavior of RM/COBOL, Versions 2.0 and 2.1.

- **Selection of the function key list to be used**

Allows you to specify which function key list your system will use to map control and terminfo codes onto user function keys, as follows:

1. The default. The standard IBM user function key list.
2. The compatibility function key list. The list supplied with the AIX VS COBOL system is the RM/COBOL function key list, but you may alter this for compatibility with any supported dialect of COBOL.

- **Control action of the COLUMN + n clause**

Controls the location of the field in a SCREEN SECTION accept or display when COLUMN + 1 is used, as follows:

1. The default. The field will be positioned immediately following the previous field if COLUMN + 1 is used.
2. There will be a one-character gap between this field and the preceding field if COLUMN + 1 is used.

- **Control the default color for SCREEN SECTION accepts or displays**

Allows you to specify whether the current default screen color or white-on-black is used when no color is specified, as follows:

1. The default. The current default screen color is used when no color is specified.
2. White-on-black is used when no color is specified.

- **Control of whether cursor left/right keys can exit a field**

Allows you to control whether pressing a right or left arrow key should be able to exit an input field, as follows:

1. The default. At the start or end of a field, the cursor left/right keys move to the previous or next field, if there is one.
2. The cursor left and cursor right keys cannot move the cursor out of the field.

- **Left justification of free format edited numerics**

Allows you to control whether free format edited numeric fields are left-justified as they are entered, as follows:

1. The default. The field is not left-justified.
2. The field switch is left-justified, provided RM numeric handling is switched off.

- **Control action of Kanji modifier characters Daku-On and Han-Daku-On during an accept**

Allows you to control whether Daku-On and Han-Daku-On act as modifier characters or not.

Press **Escape** to return to the Alter Configuration menu.

Altering Tab Stop Options

Press **F4** or **T** on the Alter Configuration menu to display a submenu that allows you to set a maximum of 80 tab stops. This submenu displays a ruler at the top of the display screen on which the current positions of tab stops are shown by the letter **T**. Use the **Cursor Left** and **Cursor Right** keys to move the cursor along the ruler. Press one of the following keys to perform the function of your choice:

F2 or **S** Set tab stop

Sets a tab stop at the current cursor position. **T** is displayed to show the new tab stop.

F3 or **D** Delete tab stop

Deletes any existing tab stop. **T**, which shows the tab stop position, is also deleted.

F4 or **F** Finish editing tab stops

Saves the changes you have made to the positions of the tab stops.

Press **Escape** to return to the Alter Configuration menu.

Altering Indicators

Pressing **F5** or **I** on the Alter Configuration menu displays a submenu which allows you to alter the text displayed by ADIS during an ACCEPT operation to indicate various conditions. The text for each message can be up to 32 characters long. Type your own message, then press **↵** to make the change.

Press one of the following keys to select the message you want to alter:

F2 or **I** Insert/replace

Allows you to create your own messages for the insert/replace indicators and the clear insert/replace indicators.

F3 or **O** Off-end-of-field

Allows you to create your own messages for the off-end-of-field and clear-off-end-of-field indicators.

F4 or **A** Auto-clear

Allows you to create your own messages for the auto-clear and clear-auto-clear indicators.

Note: A limited amount of space in a configuration is available for message texts for indicators. If the new messages you create are too large to fit into the amount of space you have left in your configuration, an audible warning is emitted.

If you receive this warning, you must write a shorter message.

After you have successfully entered your messages, press **Escape** to return to the Alter Configuration menu.

Altering Messages

Press **F6** or **M** on the Alter Configuration menu to display a submenu that allows you to alter the text of messages displayed by ADIS during an ACCEPT operation, to indicate various error conditions. You can choose to alter all the available options or individual ones.

Press **F2** or **A** to alter all of the messages. The current text of each message and the conditions under which the text is output are shown, with one message per display screen. Press **↵** to cycle from one display screen to the next. To alter any message, type the text of the new message at the prompt on the appropriate display screen. You can alter as few or as many messages as you wish.

Press **F3** or **I** to alter individual messages. This displays a numbered list of the currently defined messages. Press **F2** or **/N** to move to the next display screen of available messages. Type the number of the message you want to change, or position the cursor on the appropriate line, either by using the keys configured to move the cursor up or down a line, or by typing **/U** to move the cursor up or **/D** to move the cursor down. Press **↵** to select that message. The message currently configured for that error condition is displayed. Type the new message at the prompt on the display screen. Press **↵** to enter the change and return to the list of messages.

You can alter the messages for the following conditions:

- Abort confirmation when the **Abort** key is pressed.
- Field must be completely filled.
- Field cannot be left empty.
- End of field has been reached.
- The cursor is past the end of field.
- Data is lost from end of field.
- Cannot insert here.
- Cannot delete here.
- Keystroke has no meaning here.
- No field beyond here.
- Cannot change character case here.
- Nothing available to retype.
- Nothing available to restore.
- Leading part of number is outside range.
- Trailing part of number is outside range.
- Sign is used incorrectly.
- Cannot use a negative value.
- No room for a two-byte character.
- Cannot use more than one decimal point.
- Must enter numeric digits.

Note: Only a limited amount of space in a configuration is available for message texts. If the new messages you create are too large to fit into the amount of space you have left in your configuration, an audible warning is emitted.

If you receive this warning, you must write a shorter message.

Once you have successfully entered your messages, press **Escape** to return to the Alter Configuration menu.

Altering Message and Indicator Positions

Pressing **F7** or **P** on the Alter Configuration menu takes you to a submenu that allows you to specify whether messages and indicators normally displayed by ADIS during an **ACCEPT** operation are to be displayed. The submenu also allows you to alter the position on the display screen at which the messages and indicators are displayed.

Use the cursor keys to move around the display screen from field to field. If you want error messages or a specific indicator to be displayed during an **ACCEPT** operation, enter **Y** at the relevant field. Enter **N** if you do not wish them to be displayed.

Note: Even if you choose not to display error messages, the Abort Confirmation message is always displayed when the **Abort** key is pressed during an **ACCEPT** operation, unless you delete any text currently configured for this message. Enter the Altering Messages submenu to do this.

If you do choose to display the messages or any specific indicators, you can alter the default positions at which they are shown on your display screen during an **ACCEPT** operation. Use the cursor keys to position the cursor in the relevant fields and type the new lines and columns at which you wish the messages or indicators to be displayed.

Note: 0101 is at the top left-hand corner of your display screen. If you configure more than one indicator to be displayed at the same position, and more than one needs to be displayed at the same time, the off-end-of-field and auto-clear indicators take priority over the insert/replace indicator.

Press **←** once you have made your entries to return to the Alter Configuration menu and save your changes. Press **Escape** to return to the Alter Configuration menu if you do not wish to save your changes, or if you have made none.

Altering ADIS Key Control

Pressing **F8** or **K** on the Alter Configuration menu displays a submenu which allows you to alter the function performed by any specific key, make a key act as an ADIS function key, or disable it completely. An ADIS function key is a key which is used during an ACCEPT operation to perform an editing function, such as moving the cursor.

Note: You can achieve the same effect by modifying the **cobkeymp** file using **keybcf**. You cannot use **adiscf** to alter the ANIMATOR or user function key codes, although these can be changed using **keybcf**.

Pressing **F2** or **E** on this submenu shows a display screen which allows you to enable or disable ADIS control keys. A list of all the functions performed by the ADIS function keys is displayed, together with the current status of each key. You can alter the status of any key to be one of the following:

- D** Disabled. The key does not perform its associated function during an ACCEPT operation.
- E** Enabled. The key performs its associated function during an ACCEPT operation.
- F** Function key. The key acts as a function key during an ACCEPT operation.

Use the cursor keys to move from field to field on the display screen, and type the required letter in the field(s) you wish to alter.

Pressing **F3** or **F** on this submenu displays a list of all the available function keys and their current editing function. The function normally associated with a key can be changed. You can map any function to any key that is defined in **terminfo**. Each function key is associated with three fields that can be altered. These fields are as follows:

- Align field** By default this field is set to N. If set to Y, a field is cleared from the current cursor position to the end of the field during an ACCEPT operation. Numeric fields are aligned to the decimal point if it is to the right of the current cursor position.
- Mapping field** This field contains the number of the function to which the associated key is currently mapped. Enter the number of the required function if you wish to alter this.
- Validate field** If you set this field to Y, all validation criteria must be satisfied before the cursor can leave a field during an ACCEPT operation. If you enter N, validation criteria is not checked during an ACCEPT.

Table 10-3 on page 10-26 shows the default settings for each of the above fields for all of the ADIS function keys.

Table 10-3. Default Mappings of ADIS Function Keys

No.	Function Name	Valid-date	Align	No.	Mapped to Name
0	Terminate accept	[Y]	[N]	[0]	Terminate ACCEPT
1	Terminate program	[Y]	[N]	[1]	Terminate program
2	Carriage return	[Y]	[N]	[0]	Terminate ACCEPT
3	Cursor left	[Y]	[N]	[3]	Cursor left
4	Cursor right	[Y]	[N]	[4]	Cursor right
5	Cursor up	[Y]	[N]	[5]	Cursor up
6	Cursor down	[Y]	[N]	[6]	Cursor down
7	Move to start of screen	[Y]	[N]	[7]	Move to start of screen
8	Move to next tab stop	[Y]	[N]	[11]	Move to next field
9	Move to previous tab stop	[Y]	[N]	[12]	Move to previous field
10	Move to end of screen	[Y]	[N]	[10]	Move to end of screen
11	Move to next field	[Y]	[N]	[11]	Move to next field
12	Move to previous field	[Y]	[N]	[12]	Move to previous field
13	Change case of character	[Y]	[N]	[13]	Change case of character
14	Rubout character	[Y]	[N]	[14]	Rubout character
15	Retype rubbed out character	[Y]	[N]	[15]	Retype rubbed out character
16	Insert single character	[Y]	[N]	[16]	Insert single character
17	Delete character	[Y]	[N]	[17]	Delete character
18	Restore deleted character	[Y]	[N]	[18]	Restore deleted character
19	Clear to end of field	[Y]	[N]	[19]	Clear to end of field
20	Clear field	[Y]	[N]	[20]	Clear field
21	Clear to end of screen	[Y]	[N]	[21]	Clear to end of screen
22	Clear screen	[Y]	[N]	[22]	Clear screen
23	Set insert mode	[Y]	[N]	[58]	Insert toggle
24	Set replace mode	[-]	[-]	[255]	Undefined
25	Reset field (Undo)	[Y]	[N]	[25]	Reset field (Undo)
26	Move to start of field	[Y]	[N]	[26]	Move to start of field
Other values that can be mapped:					
55 :	RM clear field	56 :	RM back space	57 :	RM tab
58 :	Insert toggle	59 :	Replace toggle	60 :	Forward tab
61 :	Back tab	62 :	Restore	255 :	Undefined

Press **←** in either submenu after you have made your entries to return to the Alter Configuration menu and save your changes.

Press **Escape** in either submenu to return to the Alter Configuration menu if you do not wish to save your changes or if you have not made any.

Load Option

Selecting the **load** option by pressing **F3** or **L** on the main **adisf** menu loads an existing configuration into memory. You must load a configuration into memory before you can make any changes to that particular configuration.

When you select the **load** option, **adisf** shows a display screen on which the available configuration files in **ADISCTRL** are listed, as shown in Figure 10-5.

```
Number      Name
-----
1          Default Configuration
2          RM COBOL 2.0 Compatibility

ADISCF----Load-Configuration-----

F1=/Help Up=/Up List Down=/Down List Enter=Select Configuration  Escape
Select configuration using cursor keys or enter number [1]
```

Figure 10-5. Load Option

In the above example, this is the **Default Configuration** file. To select a file, enter its number at the prompt, or move the cursor to the line specifying the required file (either by using the keys configured to move the cursor up or down a line, or by typing **/U** to move the cursor up, or **/D** to move the cursor down). When you move the cursor to a file name, that file and its number appear at the prompt position at the bottom of your display screen.

Once you have selected a configuration file, press **↵**. **adisf** loads the configuration file you have selected into memory from the **ADISCTRL** file.

You are then returned to the main **adisf** menu. The configuration file you have just selected appears on the information line. You can use the **Alter** option to alter this configuration.

Save Option

Once you have altered a configuration file using the **Alter** option, press **F4** or **S** on the main **adiscf** menu to save your changes. You must do this if you want to use the new version of your configuration. If you attempt to leave **adiscf** without first saving any changes you have made to a configuration, you are asked to confirm that you want to exit without saving any changes.

Press the relevant key to select the option of your choice:

F2 or **U** New configuration

Saves your configuration as a new configuration file within ADISCTRL. You are prompted to type the name of this file.

F3 or **O** Overwrite existing configuration

Saves your configuration in ADISCTRL with the same name as an existing configuration file.

Note: This option overwrites the existing file with that name.

adiscf displays a numbered list of the existing configuration files. Type the number of the file you want to overwrite, or position the cursor on the relevant line (either by using the keys configured to move the cursor up or down a line, or by typing **/U** to move the cursor up, or **/D** to move the cursor down).

After making your entry, press **↵** in either submenu to return to the main **adiscf** menu and save your configuration file.

If you do not wish to save your configuration file, press **Escape** in either submenu to return to the main **adiscf** menu.

Note: You can store up to a maximum of 16 configuration files in ADISCTRL. If you try to save more than this number, you will receive the following message:

The configuration file is full - No new entries allowed

If you receive this message, you must either delete or overwrite an existing configuration before you can save any others.

Delete Option

To delete an existing configuration from the ADISCTRL file, select the **delete** option on the main **adiscf** menu by pressing **F5** or **D**.

adiscf then shows a display screen listing the existing configurations. Choose the configuration you want to delete by entering its number at the prompt (either by using the keys configured to move the cursor up or down a line, or by typing **/U** to move the cursor up, or **/D** to move the cursor down).

Press **↵** to delete the selected configuration. The selected file is deleted from the ADISCTRL file and you are returned to the main menu.

Note: You cannot delete the configuration which is currently chosen. To delete this configuration, you must first use the **choose** option to choose an alternative configuration.

Choose Option

Select the **choose** option from the main **adiscf** menu by pressing **F6** or **C**. This allows you to choose the configuration to be used by ADIS when you next run your programs.

adiscf shows a display screen that lists the available configurations. Choose a configuration either by entering its number to the prompt or by positioning the cursor on the relevant configuration name (either by using the keys configured to move the cursor up or down a line, or by typing **/U** to move the cursor up, or **/D** to move the cursor down).

Press **↵** to choose the selected configuration and return to the main **adiscf** menu. When you next run a program using your AIX VS COBOL system, ADIS will automatically use the configuration you have just chosen.

Press **Escape** to return to the main **adiscf** menu if you do not want to choose a configuration. The current configuration is then used by ADIS the next time you run a program using your AIX VS COBOL system.

Note: Choosing a configuration specifies the configuration ADIS is to use when you next run a program. Loading a configuration specifies the configuration you wish to alter. Loading a configuration does not affect the chosen configuration, and choosing a configuration does not affect the loaded one.

Chapter 11. Debugging Your Program Using ANIMATOR

Contents

About This Chapter	11-3
Introduction	11-4
Facilities Not Supported by ANIMATOR	11-5
Getting Started	11-5
Running ANIMATOR	11-6
Specifying Directives	11-6
ANIMATOR Directives	11-6
ANIMATOR Display Screen	11-7
Using ANIMATOR Commands	11-8
Help Display Screens	11-9
Animating STOCK1	11-9
Using Break Points	11-11
Examining the Contents of Data Items	11-13
Ending Animation	11-14
Animating Your Own Programs	11-15
Using the ANIMATOR Switch	11-15
Command Line Switches	11-16
File Searches	11-16
Animating CALLED Programs	11-17
OS/VS COBOL-Style PERFORMS	11-17
Other Remarks about Animation	11-18
Cursor Control Keys	11-18
ANIMATOR Commands	11-19
Help	11-19
View	11-20
Align	11-20
eXchange	11-20
Where	11-20
looKup	11-21
word-left (<) and word-right (>)	11-21
Escape Key	11-21
Letter Commands	11-21
Step	11-21
Go	11-22
next-If	11-22
Perform	11-23
Reset	11-23
Break	11-24
Env	11-27
Query	11-32
Find	11-37
Locate	11-38
Text	11-39
Do	11-40
ANIMATOR Command Summary	11-41

About This Chapter

This chapter describes how to use the ANIMATOR facility. ANIMATOR can be used to interactively debug your AIX VS COBOL program. This chapter contains a list of all the available ANIMATOR commands and a step-by-step example of how to use ANIMATOR to inspect the demo program **stock1**.

Introduction

ANIMATOR is an interactive program debugging tool for use with AIX VS COBOL. ANIMATOR executes the intermediate code produced by the AIX VS COBOL compiler, and simultaneously displays the source program on your display screen. As execution proceeds through the intermediate code, ANIMATOR displays the corresponding part of the source program on your display screen.

With ANIMATOR, you can see the sequence in which the statements of your program are executed. You can halt the program at any time and display the contents of data items and change them. You can alter the sequence in which statements are executed or cause statements to be skipped.

These facilities allow you to debug your programs. You can also use ANIMATOR to familiarize yourself with the logic of a program written by someone else.

To debug statically bound native code, you can use the symbolic debugger, **dbx**. See the **-g** option in Chapter 4, "The COBOL Interface" for more information.

Facilities Not Supported by ANIMATOR

ANIMATOR will not execute statements that implement the COBOL communications facility (such as SEND, RECEIVE, or ENABLE). You can still animate such programs, provided you use ANIMATOR commands to ignore all such statements.

Programs containing multi-reel file-handling syntax do not display prompts for reel swaps when you animate them.

Getting Started

ANIMATOR was designed to run on an 80 by 25 character display screen. If you run ANIMATOR on a display screen that is not 80 by 25, the screen integrity is not guaranteed.

If you animate a program which contains READs from **stdin**, you can enter input by keying it in and terminating with **↵**. When the READ from **stdin** is executed, you will not be prompted to enter the input.

When you animate programs that **WRITE** to **stdout**, the screen may be corrupted when the **WRITE** is executed. You can restore the display by using the **TEXT** command followed by the **REFRESH** command.

If you animate a program using the **anim** command without specifying an extension and both the intermediate (**.int**) and generated code (**.gnt**) versions exist, the **.gnt** version of the program is run.

If the **.idy** and copy source files that ANIMATOR requires are not in the directory specified on the **anim** command line, you can use **COBIDY** to specify alternative paths to search for the **idy** files and use **COBCPY** to specify alternative paths to search for the copy files. See Appendix A "Environment Variables" for details.

Before you can animate a program, you must compile the program with the AIX VS COBOL compiler using the **-a** flag. To compile the example program **stock1** for animation, enter the following:

```
cob -a stock1.cb1 ↵
```

You can also compile a program by using the **ref** directive (you must use the **-C** flag in the **cob** command to use the **ref** option). If you compile a program with **ref**, the source listing produced by the compiler includes the addresses of data items within the intermediate code produced by the compiler. You can use these addresses to access data items when using ANIMATOR.

Running ANIMATOR

Now that you have compiled **stock1** you can animate it by entering the following:

```
anim stock1.int ↵
```

Specifying Directives

You can also specify directives after the program name on the command line, before pressing ↵ to begin animation. Use the following format for directives:

```
[no] keyword "argument"
```

where:

no turns *keyword* off. **no** can either adjoin the keyword or be separated from it by one or more spaces. **no** applies only to certain directives specified later in this chapter. *keyword* is a directive. "*argument*" is a qualifier to *keyword* and applies to only certain directives where specified in the list in "ANIMATOR Directives." The argument must appear in either of the following forms:

- *argument*\
- *argument*\.

The argument can adjoin *keyword* or be separated from it by one or more spaces. When quotes are used, *argument* may contain spaces. When parentheses are used, no spaces are permitted.

ANIMATOR Directives

Use the following directives to control the animation of your program:

```
[ no break  
  break "procedure-name"  
  break =procedure-name ]
```

This directive causes ANIMATOR to zoom through the program to a break point set at the paragraph or section you have specified in *procedure-name*.

Default: **no break**

end

This directive indicates the end of the ANIMATOR directives. The directive can be followed by text for use in ACCEPT *data-name* FROM *function-name* statements where *function-name* is defined in the SPECIAL-NAMES paragraph with the clause COMMAND-LINE IS *mnemonic-name*.

[no] **flash**

Specifies whether you want the user display screen to flash during displays to the screen.

Default: **noflash**

```

no zoom
zoom
zoom "program-name"
zoom =program-name

```

This directive causes ANIMATOR to execute in zoom mode. If *program-name* is specified, normal step mode is restored when the sub-program with the given name is entered.

Default: **nozoom**

ANIMATOR Display Screen

When you run ANIMATOR, the display screen will clear and you can see the ANIMATOR display screen and main menu, as illustrated in Figure 11-1.

```

38 PROCEDURE DIVISION.
39 SR1 SECTION.
40 DISPLAY SPACE.
41 OPEN I-O STOCK-FILE.
42 DISPLAY SCREEN-HEADINGS.
43 NORMAL INPUT.
44 MOVE SPACE TO ENTER-IT.
45 DISPLAY ENTER-IT.
46 CORRECT-ERROR.
47 ACCEPT ENTER-IT.
48 IF CRT-STOCK-CODE = SPACE GO TO END-IT.
49 IF CRT-UNIT-SIZE NOT NUMERIC GO TO CORRECT-ERROR.
50 MOVE CRT-PROD-DESC TO PRODUCT DESC.
51 MOVE CRT-UNIT-SIZE TO UNIT-SIZE.
52 MOVE CRT-STOCK-CODE TO STOCK-CODE.
53 WRITE STOCK-ITEM; INVALID GO TO CORRECT-ERROR.
54 GO TO NORMAL-INPUT.
55 END-IT.
56 CLOSE STOCK-FILE.
57 DISPLAY SPACE.
58 DISPLAY "END OF PROGRAM".
Animate-stock 1_____Level=01-Speed=3_____
F1=Help F2=View F3=Align F4=eXchange F5=Where F6=lookUp F9/F10=word-/> Escape
Step Go Zoom next-If Perform Reset Break Env Query Find Locate Text Do 0-9=Speed

```

Figure 11-1. ANIMATOR Display Screen

The ANIMATOR display screen shows the PROCEDURE DIVISION of the example program **stock1**. The cursor is positioned on the line containing the first executable statement in the program, DISPLAY SPACE.

At the bottom of your display screen, a menu of ANIMATOR commands is displayed. See "Using ANIMATOR Commands" on page 11-8 for information on how to use these commands.

The source code of **stock1** and the command menu are separated by a dotted line, the information line. The information line contains the following items about the current state of the program:

- The name of the program, **stock1**
- The current **PERFORM** level, which tells you whether or not you are currently **PERFORM**ing a procedure
- The current animation speed, which tells you how quickly your program will run when you start to animate it.

At the moment, **ANIMATOR** is not animating your program, but is waiting for you to select one of the animator commands.

Using **ANIMATOR** Commands

This section describes how to use the **ANIMATOR** commands. The command menu at the bottom of the display screen tells you which **ANIMATOR** commands you can use. To select one of the commands in the command menu, press the key of the letter capitalized in the command in the command menu. For example, to select the **Break** command, press the **B** key. You can use either **b** or **B** (**ANIMATOR** commands are not case-sensitive). In this section, uppercase is used for keys that select **ANIMATOR** commands, but both uppercase and lowercase letters can be used to make your selection.

Note: Most commands are selected according to the initial letter of the command, as in **Break**. There are some exceptions. For example, to select the **eXchange** command, the uppercase letter is the second letter **X**, so you press **X** to select this command.

Press **B** to select **Break**. The second line of the command menu is replaced by a different menu:

```
Set Unset Cancel-all Examine If Do On-count
```

Note: Some commands have an immediate effect on your program when you select them. Most commands cause a submenu to be displayed from which you can make a further selection.

Press **Escape** to return to the main menu.

A function key from **F1** to **F10** is associated with each command (see the first line of the command menu). You can select commands in either of two ways:

- Press the appropriate letter key, as described above.
- Press the appropriate function key, if it is provided on your keyboard.

Most of these function key commands appear on all the **ANIMATOR** command menus.

Help Display Screens

Help display screens that describe the commands are available to you while you are running the program. To look at a Help display screen, type **H** and press **↵** or press the **F1** function key. To return to the main menu, press **F1** or the space bar.

Each command has a Help display screen that describes the commands on its submenu. Access a Help display screen by pressing **H** or **F1** after you select the command.

For example, press **B** to select the **Break** command. Then, press **H** or **F1** to get the Help display screen describing **Break** and its associated command menu. Press **F1** or the space bar to return to the **Break** command menu, then press **Escape** to return to the main command menu.

“Animating STOCK1” describes how to use some of the common commands to animate **stock1**, particularly the following:

- Execute your program under the control of **ANIMATOR**.
- Set break points in your program to halt the execution at certain points.
- Look at and change the value of data items in a program while it is halted.

Animating STOCK1

With **ANIMATOR** running, and the source of **stock1** on your display screen, you can execute your program in any of the following three ways:

- Step through the program one statement at a time, using the **Step** command.
- Run the program continuously, using the **Go** command.
- Run the program as you would normally, without animating it, using the **Zoom** command.

Step Command

Select **Step** by pressing **S**. The cursor moves to the next statement in **stock1**, **OPEN I-O STOCK-FILE**.

You have executed the first statement in **stock1**, **DISPLAY SPACE**. **Step** causes **ANIMATOR** to execute the current statement and stop.

Press **S** four more times to execute the next four statements in **stock1**. Each time you press **S**, the cursor moves to the next statement to be executed. The cursor moves past lines with **NORMAL-INPUT** and **CORRECT-ERROR** because these are paragraph names and cannot be executed.

After you have pressed **S** four times, the next statement to be executed is the following:

ACCEPT ENTER-IT.

Before you execute this statement, take a look at the display screen by pressing **V** to select the **View** command. The **ANIMATOR** display screen is replaced by the following:

```
STOCK CODE < >
DESCRIPTION < >
UNIT SIZE < >
```

This is the current state of the display screen as you would see it if you were running **stock1** normally. The **View** command allows you to switch between the ANIMATOR display screen and the user display screen so that you can keep track of any display screen output that your program is producing. To get back to the ANIMATOR display screen, press any key.

Press **S** again to execute the current statement, **ACCEPT**. The ANIMATOR display screen is replaced by the user display screen that you have just viewed. Whenever ANIMATOR executes an **ACCEPT** statement, you are automatically switched over to the user display screen so you can enter the required data. Enter some appropriate values in the three fields on the user display screen and press **↵**.

When you press **↵**, you complete the **ACCEPT** statement, and the user display screen is replaced by the ANIMATOR display screen. The next statement to be executed is now the statement following the **ACCEPT**; an **IF** statement.

Go Command

You can now execute your program statement by statement, using the **Step** command. You can also use the **Go** command to animate your program so that statements are executed without interruption. Press **G** to select the **Go** command.

G restarts **stock1**. ANIMATOR no longer stops after executing one statement, but continues executing statements in sequence. As the cursor reaches the last statement in paragraph **CORRECT-ERROR, GO TO NORMAL-INPUT**, the cursor moves to the first statement in paragraph **NORMAL-INPUT**, the **MOVE** statement.

When the cursor moves to the second statement in **CORRECT-ERROR**, the **IF** statement, press **Escape**. A message is displayed, as follows:

Keyboard interrupt

The message is shown at the bottom of your display screen, and ANIMATOR stops animating the program.

You can interrupt program animation at any time by pressing **Escape**. This allows you to use other ANIMATOR commands to find out what is happening in your program.

You can change the speed at which your program runs while you are animating it. On the command menu, the following entry is displayed at the bottom of the second line:

0-9 = speed

The animation speed of a program is established by a number from 0 through 9. 0 is the slowest speed and 9 is the fastest. The following entry appears on the information line on your display screen:

Speed=3

This entry indicates that the current animation speed is 3. You can alter the animation speed by pressing the appropriate digit key. To increase animation speed to 6, press **6**, then press **G** to restart the program. This causes the cursor to move from statement to statement more quickly. The information line is updated to reflect the change in animation speed.

Let **stock1** run at speed 6 for a couple of cycles through CORRECT-ERROR, entering appropriate data on the user display screen as it appears. To change the animation speed to 1, press **1** while the ANIMATOR display screen is displayed. The program now runs much more slowly. The execution of the program does not need to be halted to change the ANIMATOR speed.

Press **Escape** to halt the program.

Zoom Command

You can execute a program as though ANIMATOR is not there, using the **Zoom** command. Select **Zoom** by pressing **Z**.

Select **Z** to restart **stock1**. The ANIMATOR display screen is replaced by the user display screen. Enter some data and press **↵**. When you animate **stock1**, this has the effect of returning you to the ANIMATOR display screen. However, since **stock1** is running in zoom mode, it behaves in the same way as when it is run without ANIMATOR. The data that you enter is cleared from the display screen and you are prompted to enter more data.

You can regain control of a program running in zoom mode by pressing **Escape**. When you press **Escape**, you are returned to the ANIMATOR display screen with execution stopped at whichever statement was about to be executed.

Using Break Points

You can use ANIMATOR to set break points in a program. A break point is a point in your program at which execution automatically stops whenever that point is reached. For example, you can set a break point immediately after a particular statement so that ANIMATOR stops your program immediately after executing the statement, allowing you to look at the effect the statement has.

To use **Break**, press **B** from the main command menu. The **Break** command menu is displayed.

Setting Break Points

To set a break point in **stock1** so that ANIMATOR stops immediately before executing the ACCEPT statement at the start of CORRECT-ERROR, use the **Cursor Up** and **Cursor Down** arrow keys to move the cursor from its current position to the line containing the ACCEPT statement. Select **Set** by pressing **S**. **Set** sets a break point on the statement at the current cursor position.

When you set a break point on the ACCEPT statement, ANIMATOR detects the break point when it is about to execute the statement, and halts the program until you restart it using **Step**, **Go**, or **Zoom**.

Start **stock1** by pressing **G**. When the ACCEPT statement is reached, a message is displayed at the foot of the display screen:

Break-point encountered

The program stops without executing ACCEPT.

This break point remains active until you unset it. Refer to "Unsetting Break Points" on page 11-12.

You can set from 1 through 4 break points in a program. To set another break point on the last statement in CORRECT-ERROR, the GO TO NORMAL-INPUT statement, press **B** to get the **Break** command menu. Then, move the cursor to the line containing the GO TO statement. Press **S** to set a break point on the statement.

Press **G** to start **stock1**. The program halts before executing the GO TO statement. Press **G** again to restart the program. The program is again halted by the first break point, on the ACCEPT statement. Press **G** again; the program halts at the GO TO statement.

The ANIMATOR display screen does not indicate whether a break point is set on a particular statement. You can find out where break points are set by using **Examine** in the **Break** command menu.

Press **B** to select the **Break** command menu. Press **E** to select the **Examine** command. The cursor moves to the line containing the ACCEPT statement, indicating that your first break point is set here. Now, press **E** again. The cursor moves to the line containing the GO TO statement, indicating the position of your second break point. Each time you select **Examine**, the cursor moves to the next break point. When you have examined the last break point, select **Examine** again to display the first break point again.

Unsetting Break Points

You can use the **Unset** command in the **Break** command menu to remove break points you have set.

To remove the break point that you set on the ACCEPT statement, press **B** to get the **Break** menu. Move the cursor to the line of the ACCEPT statement and press **U** to **Unset** the break point. **Unset** removes the break point from the statement at the current cursor position.

To check whether the break point has been removed, press **G** to start **stock1**. The execution will not stop at the ACCEPT statement.

You can remove a break point by using the **Cancel-all** command. **Cancel-all** removes all break points that are set. You do not need to position the cursor on a line with a break point when you use **Cancel-all**.

To remove the remaining break point on the GO TO statement, press **B** to select the **Break** command menu. Then, press **C** to select **Cancel-all**. Press **G** to restart **stock1** and check that the break point is actually gone. Allow the program to cycle a few times around the loop, entering appropriate data on the display screen, and stop the program by pressing **Escape**.

Examining the Contents of Data Items

Query allows you to look at the current contents of any data item in your program.

For example, you can use **Query** to look at the contents of the data item called CRT-STOCK-CODE.

Press **Q** to select **Query**. The second line of the command menu is replaced by the **Query** menu:

```
Cursor-name Enter-name Repeat Monitor-off Dump-list
```

To tell ANIMATOR which data item you want to view, do one of the following:

- Position the cursor on the name of the item and use the **Cursor-name** command.
- Use the **Enter-name** command and enter the name of the data item.

To use **Cursor-name** command, use the **Cursor Up** and **Cursor Down** arrow keys to move the cursor to any line referring to the item CRT-STOCK-CODE (for instance, the second line in CORRECT-ERROR). Then, use the **Cursor Left** and **Cursor Right** keys to move the cursor anywhere inside the string CRT-STOCK-CODE. Press **c** to select **Cursor-name**. **Cursor-name** selects and displays the contents of the data item indicated by the current cursor position. The current contents of CRT-STOCK-CODE are displayed at the bottom of the display screen.

To return to the main menu, press **Escape**. Press **S** twice to step through the next two statements. The CRT-STOCK CODE contents display disappears from the display screen.

You can monitor the data item to watch what happens to the contents of CRT-STOCK-CODE as your program runs.

To monitor the data item, go to the **Query** menu by pressing **Q**. Next, select CRT-STOCK-CODE in the same manner as with **Cursor-name** (see “Examining the Contents of Data Items”).

When the contents of CRT-STOCK-CODE are displayed, the **Query** menu is replaced by the following menu:

```
F1or/H=help F2=/C=clear F3=/X=hex F4=/M=monitor  
F7=/P=parent F8=/S=son F9=/B=brother /0=Other menu Escape
```

The **Monitor** command, shown on the **Query** menu, causes the contents of the item you have selected to be displayed on the display screen when you return to the main menu and restart the program. Each time the contents of that item are changed, the changes are reflected in the display on the display screen.

To select the **Monitor** command, press the / key and the **M** key. In the cases where ANIMATOR is capable of accepting input on the bottom line, you must press the following two keys to select a letter command:

- The / key, which acts as an escape key telling ANIMATOR to expect a command rather than input
- The appropriate letter key.

Note: When you use two keys rather than one key to select a command, press the keys sequentially rather than simultaneously. For example, press /, and then press **M**.

As an alternative, you can use the appropriate function key to select a command.

Press /**M** to select **Monitor**. Now, press **G** to restart **stock1**. When ANIMATOR executes the following statement:

```
MOVE SPACE TO ENTER-IT.
```

in **NORMAL-INPUT**, the displayed contents of **CRT-STOCK-CODE** are cleared (replaced by spaces).

Press **Escape** to halt the program.

Ending Animation

You can end animation of a program in either of the following two ways:

- Allow the program to run to completion.
- Halt the program and leave ANIMATOR.

If your program executes a **STOP RUN** statement, the program returns control to ANIMATOR, and ANIMATOR warns you that the program has terminated. You can cause **stock1** to terminate normally by entering all spaces in the **STOCK CODE** field on the user display screen (press **←** when the user display screen is shown). A message will be shown at the bottom of the display screen:

```
STOP RUN encountered with RETURN-CODE=+0000:use Escape to terminate
```

You are prompted to press **Escape** to confirm that you want to leave the program. If you press **Escape**, you are asked if you want to leave ANIMATOR. If you press **Y**, ANIMATOR closes down; if you press **N**, ANIMATOR repositions the cursor on the **STOP RUN** statement.

If you do not press **Escape**, ANIMATOR leaves the cursor positioned on the **STOP RUN** statement and returns you to the main menu.

You can leave ANIMATOR at any time. If the program is animating or running in zoom mode, press **Escape** to halt the program, then press **Escape** again to leave ANIMATOR. You are prompted to confirm that you want to halt the program and leave ANIMATOR.

Animating Your Own Programs

To animate your own programs, do the following:

1. Compile your program with the **-a** flag. This causes the compiler to produce a file with the extension **.idy**.

Ensure that all files are present when you animate the program. If the program contains **COPY** statements, ensure that the copy files are also present.

2. Enter the following:

```
anim file-name ↵
```

where *file-name* is the name of the intermediate code file produced when you compile your program. Refer to “File Searches” on page 11-16 for details on how ANIMATOR searches for the file to animate.

3. Enter ANIMATOR commands to control animation of your program.

Using the ANIMATOR Switch

You can animate a compiled program other than with the **anim** command. An AIX environment variable called **COBSW** has several uses, one of which is to set the ANIMATOR switch. To set the ANIMATOR switch, assign the value “+A” to **COBSW**:

- If you are using **sh** or **ksh**:

```
COBSW="+A" ↵  
export COBSW
```

- If you are using **csh**:

```
set COBSW="+A"
```

Once you have set the switch, run the AIX VS COBOL programs as usual with the **cobrun** command. When the Run Time Environment (RTE) loads and enters an intermediate code program, that program will be executed under ANIMATOR control, provided the following conditions are true:

1. The ANIMATOR switch is set.
2. The **.idy** and **.cbl** files for the program are present. The **.idy** file is produced only if you compile the program with the **-a** flag (the default).

For example:

```
cob -a myprog.cbl ↵
```

compiles **myprog.cbl** so that it can be animated. To animate **myprog**, do one of the following:

- Enter the command:

```
anim myprog.int ↵
```

- Set the ANIMATOR switch, using the appropriate command for your shell, and run **myprog** with **cobrun**:

```
COBSW="+A"  
cobrun myprog ↵
```

You can use the ANIMATOR switch to animate a program called by a nonCOBOL program, or by a COBOL program that is not itself animated. The switch turns on ANIMATOR whenever a program that can be animated is loaded and entered.

Command Line Switches

When you run a program, you can set or unset a number of switches in the command line.

If you animate a program, you cannot specify switch settings in the **anim** command. You can, however, pass switch settings to an animated program with the COBSW environment variable and the ANIMATOR switch. Do this by including the values of switch settings in the value of COBSW, following the setting of the ANIMATOR switch. For example:

```
COBSW="+A -1 +4"  
cobrun myprog ◀
```

animates **myprog**, unsets switch 1, and sets switch 4.

See Chapter 7, "Running an AIX VS COBOL Program" for more information about switch parameters.

File Searches

When you run ANIMATOR, a search is made along a specified path for the COBOL source files (the **.idy**, **.cbl**, and **.cpy** files) required by the program to be animated. The path ANIMATOR searches is either specified by default as the current directory or by the path set up in the command line preceding the name of the file to be animated.

If you animate your program using the **+A** run-time switch and you have COBPATH set, the **.cbl** files must be in the same directory as the **.int** files.

You can specify alternative paths that ANIMATOR is to search if the **.idy** source files that ANIMATOR requires are not in the directory specified in the command line. ANIMATOR uses the AIX environment variable COBIDY. This environment variable must be set up before execution of ANIMATOR is started.

To allow ANIMATOR to search for the **.idy** files it requires, you can set up COBIDY with either a single path or a number of paths. You can set up a single path in COBIDY as follows:

```
COBIDY="/usr/lib/cobol"
```

You can set up more than one path as follows:

```
COBIDY="/usr/lib/cobol:/usr/myfile:" ...
```

In the examples above, the quotation marks are used to delimit the string. The quotation marks are not present in the value of the environment variable.

When ANIMATOR fails to find the **.idy** files for the file to be animated in the path specified in the command line, COBIDY is automatically appended to the file name. ANIMATOR uses any paths that have been set up in COBIDY to search the specified directories for the **.idy** files it requires. If ANIMATOR cannot find the **.idy** files using any of the specified paths, and no other error condition has occurred, a file not found error condition is returned.

When more than one path has been specified in COBIDY, and when the use of a path results in failure to find the files required for animation, the search on the next specified path occurs only if the file not found error condition is returned. If any error condition other than file not found is returned, the system will output the specific error condition and will not attempt to search any subsequent paths.

Similar path searches use the COBCPY environment variable to find the copy files needed. The **.cbl** source must be in the current directory.

Animating CALLED Programs

When you animate a program, any programs that it CALLs are also animated, if these programs have been compiled for animation by setting the **-a** flag with **cob**.

You can specify that a CALLED program is to be executed without animation. See “Step” on page 11-23.

You can animate a CALLED program without animating the program that CALLs it by using the **anim** command with the name of the CALLED program. If the CALLED program does not expect any parameters from the CALLing program (if the PROCEDURE DIVISION header has no USING phrase), this is sufficient. However, if the CALLED program expects parameters from the CALLing program, you must supply these parameter values before you execute the program.

For each parameter expected by the CALLED program, do the following:

1. Find an occurrence of the parameter name (such as the USING phrase of the PROCEDURE DIVISION header) and move the cursor anywhere within the name.
2. Press **Q** to get the **Query** menu
3. Press **C** to use the **cursor-name** command. A message is displayed at the foot of the display screen:
Linkage record not linked; assign data area? Y/N
4. Press **Y** and enter the appropriate value in the displayed field.

OS/VS COBOL-Style PERFORMS

Nested PERFORM statements behave differently in IBM OS/VS COBOL and in AIX VS COBOL. The difference lies in what happens when control reaches the exit point of a PERFORM while inner PERFORMs are still active.

In AIX VS COBOL, only the innermost PERFORM exit point is active. When control reaches an exit point for an outer PERFORM, no PERFORM return occurs.

In OS/VS COBOL, all PERFORM exit points are active simultaneously. When control reaches an exit point for an outer PERFORM, a return from that PERFORM will occur (and from any inner PERFORMs).

You can specify that PERFORMs in an AIX VS COBOL program are to behave as OS/VS COBOL PERFORMs by compiling the program with the directive **perform-type = osvs**. When you animate a program compiled with **perform-type = osvs**, ANIMATOR displays a message warning you that OS/VS COBOL behavior is being implemented (rather than AIX VS COBOL behavior) upon reaching a PERFORM exit point when inner PERFORMs are still active. ANIMATOR prompts you to indicate if you want future occurrences of this kind to be similarly flagged.

Other Remarks about Animation

You may encounter the following behaviors when using animation:

- If an I-O error occurs during animation and no file status bytes were defined, then the RTE error returned for the I-O error will not indicate the correct status code. To determine the correct status code, use the **query** command on the file in which the error occurred to obtain the file status returned.
- If you animate a program that contains READs from **stdin**, you will not be prompted to enter the necessary input when those READs are executed. When a READ from **stdin** is executed, you can enter input simply by keying it in and terminating it by hitting the Enter key.
- When you animate programs that WRITE to **stdout**, the screen may be corrupted when the WRITE is executed. You can restore the display by using the **Text** command followed by the **Refresh** command.
- You can animate an ACCEPT statement which gets information from the command-line. Use the **end** animator directive and give the command-line information after it. The animator will use this data when the ACCEPT statement accessing the command-line is executed. See "ANIMATOR Directives" on page 11-6 for more information on using the **end** animator directive.

Cursor Control Keys

You can use the following cursor control keys on your keyboard to alter the contents of the ANIMATOR display screen to show any part of your program:

Key	Function
←	Moves the cursor one character to the left.
→	Moves the cursor one character to the right.
Tab	Moves the cursor to the next tabulated position.
Backtab	Moves the cursor to the previous tabulated position.
Home	Moves the cursor to column 8 of the current line. If you press Home twice, the cursor moves to column 8 on the first line of the display screen. If you press Home three times, the cursor moves to the first line in the file.
End	Moves the cursor to the end of the current line. If you press End twice, the cursor moves to the end of the bottom line of the display screen. If you press End three times, the cursor moves to the last line in the file.
↵	Moves the cursor to column 8 of the next line.
↑	Moves the cursor up one line.

↓	Moves the cursor down one line.
PgUp	Displays the previous 20 lines of source.
PgDn	Displays the next 20 lines of source.
10 x pg up	Moves the cursor up 200 lines (or to the first line, if there are fewer than 200 lines before the current line).
10 x pg dn	Moves the cursor down 200 lines (or to the last line, if there are fewer than 200 lines after the current line).
Ctrl-e	Moves the cursor to the end of the file.
Ctrl-t	Moves the cursor to the top of the file.
Ctrl-v	Escapes to the shell. You are able to enter AIX commands. Type "exit" to return to the animator.

ANIMATOR Commands

This section describes each of the ANIMATOR commands in detail, with a brief summary of each command.

The current line is the line of source on which the cursor is currently positioned. The current statement is the statement in your program that the ANIMATOR will execute next.

The current line and the current statement are not necessarily the same. When your program is not executing, you can use the cursor control keys to move the cursor to any portion of your source program.

You can invoke ANIMATOR commands in either of two ways:

- Press one of the function keys on your keyboard.
- Press one of the letter keys on your keyboard, using either the uppercase or lowercase version of the letter.

Your keyboard may not have all the same function keys described here. Each function key command has a letter equivalent, so all function key commands are available. Check the contents of the command menu or its associated Help display screen to verify which commands are available.

Help

Each command menu has an associated Help display screen, providing a brief description of the commands that you can select from the command menu. If you select **Help** from any of the menus, a Help display screen will appear, describing the commands in that menu. Select **Help** by pressing either **F1** or **H**.

To leave a Help display screen and return to the command menu the screen describes, press **F1** or the space bar.

You can use **F1** from any menu to display a Help display screen. However, on some command menus, pressing **H** will not display a Help display screen. Instead, you must press **/H**. A command menu will indicate whether you must press **/H** rather than **H**.

View

View switches between the ANIMATOR display screen and the user display screen (the display screen you would see if you ran your program without animation).

When you are animating a program, your display screen shows the ANIMATOR display screen, which shows the source of the program as it is animated. You can view the user display screen from time to time, using **View**, to check that any display screen output performed by your program is correct.

Note: If your program performs an ACCEPT, the system will automatically replace the ANIMATOR display screen with the user display screen for the duration of the ACCEPT.

Select **View** by pressing either **F2** or **V**.

If you select **View** while the ANIMATOR display screen is shown, the ANIMATOR display screen is replaced by the user display screen. To return to the ANIMATOR display screen, press any key.

Align

Align alters the contents of the ANIMATOR display screen. When you select **Align**, the ANIMATOR display screen is adjusted so that the current line becomes the third line of the ANIMATOR display screen.

Use the cursor control keys to position the cursor in the line you want to move. Then, select **Align** by pressing either **F3** or **A**.

eXchange

eXchange moves the cursor from one part of a split display screen to another.

Text, described in "Text" on page 11-39, divides the ANIMATOR display screen into two parts, each of which can display a different portion of your program. Many commands require you to position the cursor within the displayed source, and you cannot use the cursor control keys to move the cursor between the two portions of a split display screen.

Select **eXchange** by pressing either **F4** or **X**.

If you select **eXchange** when the ANIMATOR display screen is not divided into two portions, the command has no effect.

Where

Where moves the cursor to the current statement (the statement in your program that ANIMATOR will execute next). Use this if you have used the cursor control keys to display another part of your program and you want to return to the part of your program that is about to be animated.

Select **Where** by pressing either **F5** or **W**.

lookUp

lookUp adjusts the ANIMATOR display screen, by moving a selected line in your program to the third line on your display screen. (You can use **Align**, described in “Align” on page 11-20, only if the line you want as the third line of the ANIMATOR display screen is already on the ANIMATOR display screen.)

Select **lookUp** by pressing either **F6** or **K**. At the prompt, enter the number of the line you want to move to the third line on the ANIMATOR display screen. Then, press **↵**. The ANIMATOR display screen will be adjusted so that the line you have specified becomes the third line of the ANIMATOR display screen.

You may want to have a listing of the program with line numbers shown. When you compile the program, use the **reseq** option and the **list** option following the **-C** flag to obtain a listing with line numbers.

word-left (<) and word-right (>)

word-left moves the cursor to the previous word in the ANIMATOR display screen. Select **word-left** by pressing either **F9** or **<**.

word-right moves the cursor to the next word in the ANIMATOR display screen. Select **word-right** by pressing either **F10** or **>**.

Escape Key

The **Escape** key has three functions when you are running ANIMATOR:

- If you press **Escape** while the main menu is displayed and your program is not being animated, you will leave ANIMATOR. You are prompted to confirm that you do want to leave ANIMATOR, in case you have pressed **Escape** accidentally. Press **Y** if you want to leave ANIMATOR. Otherwise, press **N** and you will be returned to the main menu.
- If you press **Escape** while your program is being animated, animation stops immediately and you are returned to the main menu. However, if your program is executing a format **ACCEPT** when you press **Escape**, you must press **↵** to complete the **ACCEPT** before the **Escape** takes effect.
- If you press **Escape** while a command menu other than the main menu is displayed, you are returned to the main menu.

Letter Commands

The first level of letter commands are those shown in the main menu. Several of these commands, which display other command menus, are described in this section.

Step

Step causes ANIMATOR to execute the current statement in your program and then stop.

Select **Step** by pressing **S**.

Go

Go causes ANIMATOR to start animating your program; that is, at any given moment, the ANIMATOR display screen shows the portion of the source program being executed, and the cursor appears on the program line that is currently being executed.

Select **Go** by pressing **G**.

Halt animation at any time by pressing **Escape**. You are returned to the main menu.

You can select commands from the **Go** menu while your program is being animated.

Animation Speed (0-9)

Alter the speed at which your program is animated by pressing any of the digit keys 0 to 9. Animation speed is expressed as a number from 0 through 9, where 9 is the fastest speed and 0 is the slowest. The new animation speed takes effect when ANIMATOR finishes animating the current program statement.

The current animation speed is displayed on the information line. When you enter ANIMATOR, the animation speed is set at 3.

You do not have to be in the **Go** command menu to alter animation speed; press the digit keys 0 to 9 from any ANIMATOR command menu. You can also alter the animation speed while your program is not being animated. For example, you can set the animation speed before you select **Go**.

Zoom

Zoom causes your program to be executed without animation; that is, the ANIMATOR display screen is replaced by the user display screen while your program is running.

Select **Zoom** by pressing **Z**.

Note: While your program is executing without animation, it runs at the fastest possible speed, regardless of the current setting of the animation speed.

You can halt your program at any time by pressing **Escape**. The user display screen is replaced by the ANIMATOR display screen and you are returned to the main menu. The cursor is positioned on the line containing the statement that ANIMATOR will execute next.

Zoom in the main menu has the same effect as **Zoom** in the **Go** command menu.

next-If

The **next-If** command executes your program without animation up to, but not including, the next IF statement. Your program then halts and returns to the ANIMATOR display screen in which the IF statement will be the current statement.

You select the **next-If** command by pressing **I**.

Press **Escape** to halt your program before it reaches the next IF statement.

If your program does not contain another IF statement, it will run to completion.

Perform

Perform displays a menu of commands that allow you to animate your program such that:

- When a **PERFORM** statement is animated, the **PERFORMed** procedures are executed without animation.
- When a **CALL** statement is animated, the **CALled** subprogram is executed without animation.

Use this facility to execute portions of your program at the fastest speed when you are satisfied that they are bug-free.

Select **Perform** by pressing **P**.

Step

Step causes **ANIMATOR** to execute the current statement and then stop, as with the **Step** command in the main menu. However, if the next program statement is a **PERFORM** or **CALL** statement, **ANIMATOR** will execute the **PERFORMed** procedure(s) or the **CALled** subprogram without animation.

You are returned to the main menu when the statement, procedure, or subprogram has been executed.

Select **Step** by pressing **S**.

Exit

If the next program statement to be animated is in a **PERFORMed** procedure, **Exit** executes the rest of the statements in the procedure without animation.

Select **Exit** by pressing **E**.

Your program will stop when execution leaves the **PERFORMed** procedure(s). The cursor is positioned on the line that contains the next statement to be animated, after the **PERFORM** statement that executed the procedure(s).

If you select **Exit** while execution is not within a **PERFORMed** procedure, a message is displayed reminding you that the program is at **PERFORM** level 1, and the command has no effect.

Reset

Reset displays a menu of commands allowing you to alter the sequence of execution of your program in various ways.

Select **Reset** by pressing **R**.

Note: If you alter the sequence in which statements are executed in your program, the program may produce unexpected results.

Cursor-position

Cursor-position makes the statement in the current line the current program statement. When you start animation, the statements are executed in sequence from this point.

Use the cursor control keys to position the cursor in the appropriate line before you select the command. Select **Cursor-position** by pressing **C**.

Next

Next skips the current statement without executing it, and makes the statement following the current statement the new current statement.

Select **Next** by pressing **N**.

Start

Start makes the first statement in the Procedure Division the current program statement.

Select **Start** by pressing **S**.

Quit-perform

If the current program statement is in a PERFORMed procedure, **Quit-perform** causes ANIMATOR to ignore the rest of the statements in the PERFORMed procedure(s), making the statement following the most recent PERFORM the current program statement.

Select **Quit-perform** by pressing **Q**.

If you select **Quit-perform** when your program is not in a PERFORMed procedure, a message is displayed reminding you that you are at PERFORM level 1 and the command has no effect.

Break

Break displays a menu of commands allowing you to set and unset break points in your program. When ANIMATOR encounters a break point, it stops animation of your program and returns to the main menu.

Select **Break** by pressing **B**.

Set

Set sets a break point at the statement in the current line. The effect of this break point is that when ANIMATOR reaches this statement, it stops before executing it. The statement with the break point then becomes the current statement.

Use the cursor control keys to position the cursor in the appropriate line. Then, select **Set** by pressing **S**.

You can set up to four break points in a program with **Break**. If you select **Set** when you have already set four break points, or if you select **Set** for a statement that already has a break point, the command has no effect.

Program break points are set only for the duration of the current ANIMATOR session. If you set a break point and leave ANIMATOR without unsetting the break point, the break point will not be set the next time you animate the same program.

Unset

Unset removes the break point from the statement in the current line.

Use the cursor control keys to position the cursor in the appropriate line. Then, select **Unset** by pressing **U**.

If you select **Unset** for a statement that does not have a break point set on it, the command has no effect.

Cancel-all

Cancel-all removes all of the break points you have set in your program.

Select **Cancel-all** by pressing **C**.

If there are no break points set in your program, **Cancel-all** has no effect.

Examine

Examine allows you to see where you have set break points in your program. When you select **Examine** for the first time, ANIMATOR moves the cursor to the line containing the first break point in your program. Each time you select **Examine**, ANIMATOR moves the cursor to the next break point in your program. If you select **Examine** after displaying the last break point in your program, ANIMATOR moves the cursor back to the first break point.

Select **Examine** by pressing **E**.

Examine has no effect if there are no break points set in your program.

If

If sets a conditional break point at the statement in the current line. You can set a conditional break point in addition to ordinary break points using the **Set** command.

Note: A conditional break point counts as one of the four available break points in a program.

ANIMATOR will always stop animating your program when it encounters an ordinary break point. When ANIMATOR encounters a conditional break point, it tests the condition that you associate with the break point. If the condition is true, ANIMATOR stops animating your program (the conditional break point acts like an ordinary break point). If the condition is false, ANIMATOR ignores the break point and continues animating your program.

Select **If** by pressing **I**.

Use the cursor control keys to position the cursor in the appropriate line before you select the command. When you select **If**, a message in the message area prompts you to enter the condition to be associated with the break point. The condition that you enter here must conform to the COBOL syntax for conditions (see the *Language Reference* for details of the syntax for conditions). If you have already used **If**, the condition you specified when you last used the command is displayed. Press **↵** to enter the same condition again; otherwise, enter the new condition and press **↵**.

You can set only one conditional break point in a program. If you want to set another conditional break point, you must first unset the current conditional break point using the **Unset** command.

When you select **If**, the following menu of commands is displayed:

- Press **F1** or **/H** to see a Help display screen on conditional break points.
- Press **F2** or **/C** to clear the displayed condition to spaces.
- Press **Escape** to return to the main menu without setting a conditional break point.

Note: You cannot set a conditional break point on a statement that already has an ordinary break point or vice versa.

Do

Do executes a COBOL statement input from the keyboard.

Select **Do** by pressing **D**.

The **Do** command allows you to input a COBOL statement that will be executed when the break point in your program is reached. You can select the **Do** command at any time during animation of your program.

The **Do** facility in the **Break** command menu is particularly useful if you wish to try out a new COBOL statement without permanently changing your program. When you select the **Do** command, a break point is automatically set at the current line.

ANIMATOR prompts you to enter a COBOL statement. Enter a syntactically valid COBOL statement, and then press **↵**. When the break point set by the **Do** command is reached during the animation of your program, the COBOL statement you input is executed. Execution of your program continues once your input statement has been executed.

Note: The statement entered here is not included in the source code.

When you select **Do**, the following menu of commands is displayed:

- Press **F1** or **/H** to see the Help display screen for the **Do** command.
- Press **F2** or **/C** to clear the displayed COBOL statement to spaces.
- Press **Escape** to return to the main menu without entering a COBOL statement.

On-count

On-count is used in conjunction with a break point to allow you to specify the number of times you want the program to run until the break point associated with this value is activated and ANIMATOR stops animating the program.

When ANIMATOR encounters a break point with an **On-count** set, it calculates the number of times that the break point has been reached. If this is equal to the value set up in the **On-count**, ANIMATOR stops animating your program (the break point acts as an ordinary break point). If the break point has been reached fewer times than the value of the **On-count**, ANIMATOR ignores the break point and continues animating your program.

Select **On-count** by pressing **O**.

Use the cursor control keys to position the cursor in the appropriate line before you select the command. When you select **On-count**, a message in the message area prompts you to enter the number of times the break point is to be reached before it is activated. The number of times the break point is to be reached must be greater than one. If you have already used **On-count**, the number of times the break point is to be reached is displayed (it is the number you specified when you last entered the command). Press **Escape** to enter the same number of times the break point is to be reached again, and to return to the **Break** command menu to set a break point. Otherwise, enter the new value for the number of times the break point is to be reached, and press **↵** to return to the **Break** command to set a break point. The on-count numbers must be set before its break point is set.

You can set up to four **On-counts** in a program, one for each break point, up to the maximum number of break points allowed.

When you select **On-count**, the following menu of commands is displayed:

- Press **F1** or **/H** to see the Help display screen for the **On-count** command.
- Press **F2** or **/C** to clear the displayed value of **On-count** to zero.
- Press **Escape** to return to the **Break** command menu if the displayed **On-count** is to be unchanged.

Env

Env displays a command menu that allows you to control various aspects of the environment in which your program is animated.

Select **Env** by pressing **E**.

Program-break

Program-break displays a command menu that allows you to specify when ANIMATOR is to take control of a program that you are executing without animation when that program contains **CALL** statements.

Note: Make the appropriate selection from this command menu before using the **Zoom** command.

Select **Program-break** by pressing **P**.

The following options are provided on the **Program-break** command menu:

This: When you select **This** and you use the **Zoom** command to execute your program, ANIMATOR will execute your program without animation until ANIMATOR returns to the program from a CALLED subprogram. ANIMATOR will then stop executing your program and return you to the main menu. In effect, you are setting a break point on the statement immediately after each CALL statement in your program.

Select **This** by pressing T.

Select: **Select** allows you to specify the name of a CALLED subprogram that you want to animate. When you are executing your program without animation, ANIMATOR will stop your program at the first statement in the named subprogram.

Select the **Select** command by pressing S.

A message in the message area prompts you to enter the name of the subprogram that you want to animate. Enter the name of the subprogram and press **↵**.

Note: **Select** and **This** are mutually exclusive. If you first select **This** and then select the **Select** command before executing your program in zoom mode, **Select** will take effect rather than **This**. If you reverse the order, **This** will take effect rather than **Select**.

Cancel: **Cancel** cancels the effect of the command you have chosen from this menu (**This** or **Select**). After selecting **Cancel**, you can execute your program in zoom mode without ANIMATOR taking control at any point.

Select **Cancel** by pressing C.

If you have not previously selected either **This** or **Select**, **Cancel** has no effect.

Threshold-level

Threshold-level allows you to indicate to ANIMATOR that PERFORMed procedures and CALLED subprograms below a certain level are to be executed without animation.

The current PERFORM level is displayed on the information line. When you enter a program at the beginning, the PERFORM level is 01, indicating that you are not in a PERFORMed procedure.

As execution passes into a PERFORMed procedure, the PERFORM level increases by one. As execution passes out of a PERFORMed procedure, the PERFORM level decreases by one.

Note: In-line PERFORMs do not alter the PERFORM level.

Entering a CALLED subprogram also affects the PERFORM level, as shown in Figure 11-2 on page 11-29.

Main program:	PERFORM levels		
	Main	PROGA	PROGB
:	01	-	-
:			
CALL "PROGA"	01	-	-
Subprogram PROGA:			
:	02	01	-
:			
CALL "PROGB"	02	01	-
Subprogram PROGB:			
:	03	02	01
:			
:			
EXIT PROGRAM.	03	02	01
:	02	01	-
EXIT PROGRAM.	02	01	-
:	01	-	-
:			

Figure 11-2. Example of CALL Statement/PERFORM Level Relationship

As execution passes into a CALLED subprogram, the PERFORM level of the CALLING program increases by one. As execution returns from a CALLED subprogram, the PERFORM level of the CALLING program decreases by one.

Threshold-level displays a menu of commands that allow you to set and unset the threshold-level. Whenever the PERFORM level is greater than the threshold level, your program is executed without animation.

Set **Threshold-level** by pressing T.

The following options are provided on the **Threshold-level** command menu:

Set: Set sets the threshold level to the current value of the PERFORM level, as displayed in the information line.

Set **Set** by pressing S.

Unset: Unset unsets the threshold level, indicating that all PERFORM levels in your program are animated.

Select **Unset** by pressing U.

Until

Until allows you to set a *general conditional break point* in your program. The conditional break point that you can set using **If** in the **Break** command menu is associated with a particular statement; that is, ANIMATOR only tests the condition when it executes that statement. The general conditional break point applies to the whole program. ANIMATOR tests the condition before executing each statement. As long as the condition remains false, ANIMATOR continues to animate your program. As soon as the condition becomes true, ANIMATOR stops animating your program and returns you to the main menu. When animating stops due to a general conditional break point, this break point is unset.

Select **Until** by pressing **U**.

This displays a menu of commands that allow you to set and unset a general conditional break point.

Note: The general conditional break point slows the execution of your program considerably, even at maximum animation speed. Use ordinary break points wherever possible.

The following options are provided on the **Until** command menu:

Set: **Set** sets a general conditional break point for your program. You can set only one general conditional break point in a program. A general conditional break point does not count as one of the four available break points.

Select **Set** by pressing **S**.

ANIMATOR then prompts you to enter the condition for the break point. This condition must conform to the syntax for COBOL conditions (see the *Language Reference* for details). Enter the condition and press **↵** to set the break point and return to the main menu.

If you have already set a general conditional break point, selecting **Set** displays the associated condition.

When you select **Set**, the following menu of commands is displayed:

- Press **F1** or **/H** to see a Help display screen describing the general conditional break point.
- Press **F2** or **/C** to clear the displayed condition to spaces. This is useful if you are altering a break point that you have already set.
- Press **Escape** to return to the main menu without specifying a break point.

Note: The general conditional break point, like all other break points, exists only during the current ANIMATOR session. If you set a general conditional break point and leave ANIMATOR, the break point will not be set when you next animate the same program.

Unset: **Unset** removes the general conditional break point that you have set. If you have not set a break point, the command has no effect.

Select **Unset** by pressing **U**.

Examine: **Examine** displays the condition associated with the general conditional break point. If you want to change this condition, you must use the **Set** command.

Select **Examine** by pressing **E**.

If you have not set a general conditional break point, the command has no effect.

Back track

Back track allows you to record and retrace the statements executed in the execution path prior to the statement where ANIMATOR stopped animating your program. Before you can retrace your program using **Back track**, the execution path must be recorded.

Select **Back track** by pressing **B**.

The following options are provided on the **Back track** command menu:

Set: **Set** is used to start the recording of the execution path.

Select **Set** by pressing **S**.

From 1 through 100 executable statements can be recorded while the main program is being run. If a **CALL** statement is the next statement to be executed, you are able to record from 1 through 100 additional statements in that **CALL**ed procedure.

Unset: **Unset** is used to stop the monitoring of the execution path.

Note: If you want to examine the execution path, do so before using **Unset**, as **Unset** will delete the record of the path.

Select **Unset** by pressing **U**.

Examine: **Examine** is used to retrace the statements executed prior to the statement where ANIMATOR stopped animating. Use the **Cursor Up** and **Cursor Down** keys to move through the backtrack trail. The current statement indicates the executable statement reached in your retracing.

Select **Examine** by pressing **E**.

When you have traced backward 100 statements in a program, have reached the start or the end of a program, have reached the statement at which you started recording the execution path, or have reached the statement at which ANIMATOR stopped animating your program, a message is displayed in the message area to indicate that you have reached the end of the backtrack trail.

Threshold-level: **Threshold-level** in the **Examine** command menu has the same effect as **Threshold-level** in the **Env** command menu, but only for **PERFORM**ed procedures.

Press **Escape** to return to the **Examine** command menu.

Query

Query displays a menu of commands allowing you to display and alter the contents of any data item in your program.

Select **Query** by pressing **Q**.

Cursor-name

Cursor-name displays the contents of the data item whose name is indicated by the current cursor position.

Select **Cursor-name** by pressing **C**.

Use the cursor control keys to position the cursor anywhere within the appropriate data item name before you select the command. Any occurrence of the data item name will serve the purpose.

If you select **Cursor-name** while the cursor is not positioned within the name of a data item, ANIMATOR warns you that this is not a data item.

Enter-name

Enter-name displays the contents of the data item you enter.

Select **Enter-name** by pressing **E**.

ANIMATOR prompts you to enter the name of the data item.

Enter the appropriate name (in uppercase or lowercase) and press **↵** to display the contents of the data item.

You have the option of specifying an offset with the data item name. For example, if your program contains an item called ALPHA-ITEM whose current value is the string "A short string", then when you select **Enter-name** and give the name ALPHA-ITEM, the displayed contents will be:

A short string

If, however, you select **Enter-name** and give the data item name as follows:

ALPHA-ITEM + 4

the displayed contents will be:

ort string

You must leave at least one space on either side of the "+".

You can also specify a data item when you select **Enter-name** by giving the hexadecimal address of the item within the Data Division. To get the correct address, you have to compile the program with the **ref** directive to get a source listing that includes hexadecimal addresses.

When you choose **Enter-name**, a menu of commands is displayed, as follows:

1. Press **F1** or **/H** to show a Help display screen.
2. Press **F2** or **/C** to clear the displayed data item name to spaces.
3. Press **Escape** to return to the main menu without entering a data item name.

Repeat

Repeat displays the contents of the data item that you last displayed. If you have not previously displayed the contents of any data item, the command has no effect.

Select **Repeat** by pressing **R**.

Monitor-off

Monitor-off switches off monitoring of the contents of a data item. See “Monitor” on page 11-34 for details of data item monitoring.

Select **Monitor-off** by pressing **M**.

If you are not currently monitoring the contents of a data item, the command has no effect.

Dump-list

Dump-list allows you to save, on fixed-disk, a list of data values you have created for use as test data. You can create this list using the commands on a menu that you display by selecting the **Other menu** command after selecting and displaying the contents of a data item. See “Other Menu” on page 11-36 for more information about value lists.

Select **Dump-list** by pressing **D**.

The data value list is saved in a file called *name.ILS*, where *name* is the name of your program.

Note: If you have already saved a value list from this program, the file containing the list will be overwritten.

When you animate this program in a later ANIMATOR session, this list is automatically restored from the file.

Commands That Operate on a Selected Data Item

When you have selected a data item (using either the **Cursor-name**, **Enter-name**, or **Repeat** command), the **Query** command menu is replaced by another menu of commands that allow you to manipulate the displayed value in various ways. The **Query** command menu is replaced by one of two menus:

- The text menu, in which the contents of the selected data item are displayed as ASCII characters
- The hex menu, in which the contents of the selected data item are displayed both in ASCII and hexadecimal characters.

When you select a data item, the text menu is displayed by default, but you can switch between the two menus using one of the commands on the menu. Most of the commands described in the following sections are common to both menus.

Once you have selected a data item, the name of the item appears in the information line.

As long as either the hex or text menu is displayed, you can alter the contents of the selected item by typing in a new value. When you press **←** this new value will replace the previous value of the item. If you press **Escape** instead of **←**, the new value you have entered is ignored.

Clear: **Clear** clears the contents of the selected data item to spaces.

Select **Clear** by pressing either **F2** or **/C**.

heX and Text: **heX** appears only in the text menu. **heX** switches you over to the hex menu.

Select **heX** by pressing either **F3** or **/X**.

Text appears only in the hex menu. **Text** switches you over to the text menu.

Select **Text** by pressing either **F3** or **/T**.

Monitor: **Monitor** allows you to display the contents of the selected data item throughout the execution of your program. If you do not monitor the selected data item, the display of its contents will disappear when you return to the main menu. If you do monitor the selected data item, the display of its contents remains on the display screen and will be updated as the value of the data item changes during execution of your program.

Note: You can monitor only one data item in your program.

Select **Monitor** by pressing either **F4** or **/M**.

You can discontinue monitoring of a data item by using **Monitor-off** in the **Query** command menu.

Up-table and Down-table: **Up-table** and **Down-table** appear on the menu only if the data item that you select is a table.

When you select a table item using the **Query** command, the information line will contain the name of the item and the current value of the variable used as a subscript. The displayed value at the foot of the display screen represents the contents of that table entry.

Use **Up-table** and **Down-table** to select other entries in the same table and display their contents.

If you select **Up-table**, the previous table entry is selected and its contents are displayed.

Select **Up-table** by pressing either **F5** or **/U**.

If you select **Down-table**, the next table entry is selected and its contents are displayed.

Select **Down-table** by pressing either **F6** or **/D**.

If you attempt to select an entry that is not in the table (before the first entry or after the last entry), ANIMATOR warns you that you cannot do this.

Parent: **Parent** selects and displays the contents of the group item in which the selected data item occurs.

Select **Parent** by pressing either **F7** or **/P**.

If the selected data item is not contained within a group item, the command has no effect.

Son: **Son** selects the first higher level item within the selected item and displays its contents.

Select **Son** by pressing either **F8** or **/S**.

If the selected data item is an elementary item, the command has no effect.

Brother: **Brother** selects and displays the contents of the next data item with the same level as the selected data item.

Select **Brother** by pressing either **F9** or **/B**.

heX/ASCII: **heX/ASCII** appears only in the hex menu, and moves the cursor between the hexadecimal and ASCII display of the contents of the currently selected data item.

Select **heX/ASCII** by pressing either **F10** or **/X**.

Other Menu: **Other Menu** displays a menu of commands allowing you to create and manipulate a list of data values. This facility allows you to create a set of test data for your program. Once you have selected a data item using **Query**, you can select a value from this list of data values and assign it to the selected data items.

Select **Other Menu** by pressing **/O**. To return from the menu displayed by **Other Menu**, press **/O** again.

Create a list by using the **Add** command to add values to the end of the list. You can also use **Before** and **After** to add values within the list. The size and type of value that you can add to the list is determined by the **PICTURE** clause of the selected data item. Use **Locate** to display the declaration of the selected item.

Delete values from the list using **Delete**. Alter values in the list using **Update**. Scroll up and down through the items in the list using **Previous** and **Next**.

Update: **Update** alters the value of the currently displayed item in the value list.

Use **Next** or **Previous** to display the value list item you want to alter and enter the new value. Now select **Update** by pressing either **F2** or **/U**.

Add: **Add** adds a new value to the end of the value list.

Enter the value to be added to the list and select **Add** by pressing either **F3** or **/A**.

Delete: **Delete** deletes a value from the value list.

Use **Next** and **Previous** to display the value that you want to delete.

Select **Delete** by pressing either **F4** or **/D**.

Alternatively, enter the value that you want to delete from the list and select **Delete**. If that value occurs anywhere in the list, it will be deleted; otherwise the command has no effect.

Next: **Next** displays the next item in the value list.

Select **Next** by pressing either **F6** or **/N**.

Previous: **Previous** displays the previous item in the value list.

Select **Previous** by pressing either **F5** or **/P**.

Before: **Before** inserts a new value into the value list immediately before the currently displayed list item.

Use **Previous** and **Next** to display the list item where you want to make the insertion, then enter the value to be inserted.

Select **Before** by pressing either **F7** or **/B**.

Following: **Following** inserts a new value into the value list immediately after the currently displayed list item.

Use **Previous** and **Next** to display the list item where you want to make the insertion, then enter the value to be inserted.

Select **Following** by pressing either **F8** or **/F**.

Locate: **Locate** displays the declaration of the selected data item.

Select **Locate** by pressing either **F9** or **/L**.

ANIMATOR adjusts the display screen to show the line in which the selected data item is declared.

Selecting a Value from the Value List

Once you have created a list of data values using the commands in this menu, you can assign any of these values to the selected data item. To do this, use **Previous** and **Next** to select a particular list value, then press **↵**. The displayed list value is now assigned to the selected data item and you are returned to the main menu.

Effect of Cursor Control Keys on a Selected Data Item

Once you have selected and displayed the contents of a data item using **Query** command menu, you can use the cursor control keys to move the cursor through the displayed value:

- ← Moves the cursor left one character in the displayed value
- Moves the cursor right one character in the displayed value
- ↑ Displays the previous 80 bytes of a data item in the text menu, or the previous 16 bytes of a data item in the hex menu
- ↓ Displays the next 80 bytes of a data item in the text menu, or the next 16 bytes of a data item in the hex menu.

Note: If you change any characters in the displayed contents when using the cursor control keys to move through the displayed value, the changes take effect as soon as you scroll the changed characters out of the displayed area. These changes are made even if you later return to the main menu by pressing **Escape** instead of **↵**.

Find

Find searches your program for the next occurrence of a specified character string.

Select **Find** by pressing **F**.

You are prompted to enter the character string for which you are searching. If you have used **Find** previously in the same session, the last string that you specified is displayed. Press **↵** if you want to search for the same string again.

Otherwise, enter the character string for which you want to search. You can enter a string up to 32 characters long. Press **↵**, and ANIMATOR positions the cursor immediately after the next occurrence of the specified string in your program. If ANIMATOR cannot find an occurrence of the string, the following message is displayed:

Not found

and the cursor is not moved.

Find has the following characteristics:

- If the string for which you are searching is in uppercase, you must specify the search string in uppercase. If you are searching for a lowercase string, you must specify the search string in lowercase.
- **Find** searches forward only, starting from the current cursor position.
- You can include significant spaces at the end of the search string by terminating the search string with the character #.
- If you terminate the search string with the characters #M, ANIMATOR will not search the contents of any COPY files.

When you select **Find**, the following menu of commands is displayed:

- Press **F1** or **/H** to see a Help display screen describing the **Find** command.
- Press **F2** or **/C** to clear the displayed search string to spaces.
- Press **Escape** to return to the main menu without specifying a search string.

Locate

Locate displays a menu of commands allowing you to locate the definition of any data item, file, or procedure in your program.

Select **Locate** by pressing **L**.

Cursor-name

Cursor-name locates the declaration of the item indicated by the current cursor position.

Select **Cursor-name** by pressing **C**.

Use the cursor control keys to position the cursor anywhere within an occurrence of the data item, file, or procedure name before you select the command.

ANIMATOR moves the cursor to one of the following locations:

- The line in which a data item is declared
- The first line of the SELECT clause for a file
- The paragraph or section name at the head of a procedure.

Enter-name

Enter-name locates the declaration of a data item, file, or procedure whose name you enter.

Select **Enter-name** by pressing **E**.

If you have already used **Enter-name** in this session, the last name you entered is displayed. Press **↵** to use this name again.

Enter the name of the data item, file, or procedure for which you are searching, and press **↵**. ANIMATOR moves the cursor to one of the following locations:

- The line in which a data item is declared
- The first line of the SELECT clause for a file
- The paragraph or section name at the head of a procedure.

ANIMATOR warns you if it cannot locate the item whose name you enter.

When you select **Enter-name** the following menu of commands is displayed:

- Press **F1** or **/H** to see a Help display screen about the **Enter-name** command.
- Press **F2** or **/C** to clear the displayed name to spaces.
- Press **Escape** to return to the main menu without specifying a name.

Text

Text displays a menu of commands which allow you to alter the format of the ANIMATOR display screen.

Select **Text** by pressing **T**.

Split

Split divides the ANIMATOR display screen into two separate portions.

Select **Split** by pressing **S**.

Use the cursor control keys to move the cursor to the line at which you want to split the ANIMATOR display screen before you select the command.

When you select **Split**, a line is drawn across the display screen on the current line and the text in the top portion is duplicated, as far as possible, in the lower portion. The cursor is positioned in the lower portion of the display screen.

You cannot move the cursor from one part of a split display screen to the other using cursor control keys. You must use **eXchange** (press **X** or function key **F4**). The cursor control keys affect only the portion of the split display screen in which the cursor is currently positioned.

When you animate your program, only the portion of the ANIMATOR display screen that contains the current statement is affected.

You can restore a split display screen to a single display screen using **Join**. Under certain circumstances, a split display screen will be restored to a single display screen automatically. If your program CALLs and animates a subprogram, a split display screen will be restored to a single display screen upon entering the subprogram (although you can stop animation in the subprogram and split the display screen there). Moreover, when control returns to the CALLing program, the split display screen is no longer split.

Join

Join restores a split display screen to a single display screen.

Select **Join** by pressing **J**.

Refresh

Refresh repaints the ANIMATOR display screen, which may become corrupted as a result of standard ANSI DISPLAY statements.

Select **Refresh** by pressing **R**.

Do

Do executes a COBOL statement input from the keyboard.

Select **Do** by pressing **D**.

ANIMATOR prompts you to enter a COBOL statement. You can enter a valid COBOL statement here. When you enter the statement, press **↵**. ANIMATOR will execute the statement immediately.

The statement you enter here is not included in the source code. In particular, the statement will not be executed when ANIMATOR reaches the same point in your program again unless you reenter it using **Do**.

When you select **Do**, the following command menu is displayed:

- Press **F1** or **/H** to see a Help display screen about the **Do** command.
- Press **F2** or **/C** to clear the displayed COBOL statement to spaces.
- Press **Escape** to return to the main menu without entering a COBOL statement.

ANIMATOR Command Summary

The ANIMATOR commands are listed with a brief description in Table 11-1. You can select many of these commands by pressing the appropriate function key specified in the relevant menus.

Table 11-1 (Page 1 of 4). ANIMATOR Command Summary		
Main Menu	Submenu	ANIMATOR Action
Help		Display Help display screen. Also available from each submenu. Some submenus may indicate that you must press /H rather than H. Press the space bar to return from the Help display screen.
View		Display user display screen. Press any key to return to ANIMATOR screen.
Align		Make current line the third line of the ANIMATOR display screen.
eXchange		Move cursor to other portion of a split display screen.
Where		Move cursor to current statement.
looK-up		Make specified line number third line of the ANIMATOR display screen.
<		Move cursor to previous word in source.
>		Move cursor to next word in source.
Step		Execute current statement and stop.
Go		Animate program.
	0-9	Set animation speed (0 slowest, 9 fastest).
	Zoom	Execute program without animation.
Zoom		Execute program without animation.
next-If		Execute program without animation up to, but not including, the next IF statement.
Perform		Control animation of PERFORM and CALL.
	Step	Execute current statement and stop. If current statement is PERFORM or CALL, execute procedure or subprogram without animation before stopping.
	Exit	If current statement is in a procedure or subprogram, execute rest of procedure or subprogram without animation and then stop.
Reset		Alter location of current statement.
	Cursor-position	Statement on current line becomes the current statement.
	Next	Skip current statement and make the next statement the current statement.
	Start	First executable statement in Procedure Division becomes the current statement.
	Quit-perform	If current statement is in a procedure, make the current statement the statement that follows the PERFORM.

Table 11-1 (Page 2 of 4). ANIMATOR Command Summary

Main Menu	Submenu	ANIMATOR Action
Break		Set and unset program break points.
	Set	Set break point at statement on the current line.
	Unset	Remove break point at statement on the current line.
	Cancel-all	Remove all break points from the program.
	Examine	Move cursor to the next break point.
	If	Set a conditional break point at statement on the current line. /Clear Clear displayed condition.
	Do	Enter COBOL statement for execution when the break point is reached. /Clear Clear displayed COBOL statement to spaces.
	On-count	Set the number of times break point is passed before action is taken. /Clear Clear displayed number of times break point passed to zero.
Env		Specify details of animation environment.
	Program-break	Specify which program is to be animated if a suite of CALLED programs is executed without animation. This The current program will always be animated. Select Enter name of program to be animated. /Clear Clear displayed name Cancel Cancel effect of This or Select .
	Threshold-level	Set PERFORM level above which PERFORMed procedures and CALLED subprograms are executed without animation. Set Set threshold level to current PERFORM level. Unset Unset the threshold level (all procedures and subprograms to be animated).
	Until	Set or unset a general conditional break point. Set Enter a condition which, when it becomes true, causes execution to halt. /Clear Clear displayed condition. Unset Remove general conditional break point. Examine Display condition for current general conditional break point.
	Back track	Set, unset, or examine the execution path of program. Set Start monitoring of each statement executed. Unset Stop monitoring of each statement executed. Examine Retrace execution path.

Table 11-1 (Page 3 of 4). ANIMATOR Command Summary

Main Menu	Submenu	ANIMATOR Action
		<p>After selecting the retracing of the execution path:</p> <p>Cursor-up Display executed statement preceding statement on current line.</p> <p>Cursor-down Display executed statement following statement on current line.</p> <p>Threshold-level Set PERFORM level above which PERFORMed procedures are executed without animation.</p>
		<p>After selecting the level of PERFORM statements below which PERFORMed procedures are to be animated:</p> <p>Set Set threshold level to current PERFORM level.</p> <p>Unset Set threshold level to 01 (all procedures to be animated).</p>
Query		Display and optionally change contents of a data item.
	Cursor-name	Select and display contents of item indicated by cursor.
	Enter-name	Select and display contents of specified item.
	Repeat	Select and display contents of same data item queried last time.
	Monitor-off	Stop monitoring contents of selected item.
	Dump-lists	<p>Save data value list in disk file.</p> <p>After selecting and displaying the contents of a data item:</p> <p>/Clear Clear item contents to spaces.</p> <p>he/X Switch from text menu to hex menu, or vice versa.</p> <p>/Text</p> <p>/Monitor Start monitoring contents of selected item throughout execution.</p> <p>/Up-table Select and display contents of previous table item (only if selected item is a table).</p> <p>/Down-table Select and display contents of next table item (only if selected item is a table).</p> <p>/Parent Select and display contents of group item containing selected item.</p> <p>/Son Select and display contents of first higher level item within selected item.</p> <p>/Brother Select and display contents of next item at same level as selected item.</p> <p>he/X-ASCII Move cursor from hex contents display to ASCII contents display, or vice versa (only on hex menu).</p>

Table 11-1 (Page 4 of 4). ANIMATOR Command Summary		
Main Menu	Submenu	ANIMATOR Action
		<p>↑ Display previous 80 bytes (text) or 16 bytes (hex) of selected item.</p> <p>↓ Display next 80 bytes (text) or 16 bytes (hex) of selected item.</p> <p>→ Move cursor one character to the right in contents display.</p> <p>← Move cursor one character to the left in contents display.</p> <p>/Other-menu Display menu of commands for manipulating value lists.</p> <p>/Update Replace current value list item by displayed value.</p> <p>/Add Add displayed value to end of value list.</p> <p>/Delete Delete displayed value from value list.</p> <p>/Previous Display previous item in value list.</p> <p>/Next Display next item in value list.</p> <p>/Before Insert displayed value into value list immediately before the current list item.</p> <p>/Following Insert displayed value into value list immediately after the current list item.</p> <p>/Locate Move cursor to declaration of selected item.</p>
Find		Find next occurrence of specified string in the source program.
	/Clear	Clear displayed string to spaces.
Locate		Move cursor to declaration of data item, file, or procedure.
	Cursor-name	Locate declaration of name indicated by cursor.
	Enter-name	Locate declaration of specified name.
	/Clear	Clear displayed name to spaces.
Text		Control appearance of ANIMATOR display screen.
	Split	Split ANIMATOR display screen into two at current line.
	Join	Replace a split display screen by a single display screen.
	Refresh	Repaint the ANIMATOR display screen.
Do		Enter COBOL statement for immediate execution.
	/Clear	Clear displayed COBOL statement to spaces.
0-9		Set animation speed (0 slowest, 9 fastest).

Chapter 12. Designing Display Screens and Programs Using FORMS-2

Contents

About This Chapter	12-3
Introduction	12-4
Outputs	12-5
Phases	12-5
Operator Interface	12-6
FORMS-2 Validation	12-7
Initialization Phase	12-16
Initialization Display Screen I01	12-16
Initialization Display Screen I02	12-17
Work Phase	12-18
Display Screen W01	12-18
Work Display Screen	12-19
Work Phase Completion	12-28
Data Descriptions	12-29
Record Name and Data-Name Generation	12-29
Picture Generation	12-30
Editing the DDS File	12-30
Incorporation of DDS File Contents	12-30
Checkout Program	12-31
Checkout Program Generation	12-31
Checkout Program Compilation	12-31
Checkout Program Running	12-31
Checkout Processing	12-32
Checkout Completion	12-32
Display Screen Image File	12-33
Display Screen Image File Generation	12-33
FORMS-2 Maintenance	12-33
Printed Forms	12-34
Form Images in the Design Process	12-34
FORMS-2 User Display Screen Generation Example	12-35
Index Program	12-39
Index Program Generation	12-40
Index File Generation	12-41
Index Program Compilation	12-41
Index Program Running	12-41
User Index Program Example	12-43

About This Chapter

This chapter describes the FORMS-2 facility that can be used to interactively create and edit display screens for use in your IBM AIX VS COBOL programs. An explanation of how to run FORMS-2 and a description of the files produced are included.

Introduction

The FORMS-2 package is an extension to the AIX VS COBOL software development system that enables you to create and edit data entry display screens for applications programs at a console. The package provides several facilities to aid you in the design and development of interactive applications written in AIX VS COBOL:

- Translation of user display screen layouts into COBOL record descriptions for inclusion in AIX VS COBOL applications programs
- Verification of user display screen layouts in a checkout program before their incorporation in an application program
- Retention of exact display screen images of the user display screens in fixed-disk files for subsequent editing and printing
- Generation of an entire AIX VS COBOL program to allow data capture, update, and interrogation by means of application display screens and an indexed sequential file.
- If the DBCS-variety of AIX VS COBOL is installed, you can enter Double-Byte Character Set (DBCS) data when using the Forms-2 package.

Note: The maximum length for the environment variable COBDIR is 40 characters if using the FORMS-2 package.

Outputs

You can choose any valid combination of the above facilities and, depending on the options you select, FORMS-2 will automatically produce the following four types of fixed-disk output files:

- A source file of AIX VS COBOL data description statements (DDS) defining the display screens (forms) that you have designed. You can subsequently include these statements in an AIX VS COBOL application program using the COPY verb. The file is generated as *file-name.DDS*.
- A source file of a checkout program incorporating the data description statements defining your display screens. After compilation, you can verify the data entry form before building the actual application. The file is generated as *file-name.CHK*.
- Display screen image files of exact copies of the display screen that you have designed. The files are generated as *file-name.Snn*.
- A file of the source of an index program based on a display screen that you have designed. After compilation, the generated program can be used for storing, retrieving, updating, and deleting data entered through the display screen. The file is generated as *file-name.GEN*.

Phases

FORMS-2 processing is divided into a number of logically distinct units. Two main phases can be identified: the initialization phase and the work phase.

Initialization Phase

The initialization phase is performed only once and establishes the characteristics of this particular run of the program. It is a series of display screens containing self-explanatory prompts to which you reply as necessary.

Work Phase

At least two work phases are performed for each data entry display screen that you design.

The FORMS-2 display screen is analogous to a paper form, where the printed fixed text is used as a guide to entering the variable data in the space provided. To the human eye it is obvious where the variable data entry areas occur on the form, but the computer needs to have these areas defined explicitly. There are, therefore, two types of work phases: one in which you specify fixed text, and one in which you specify variable data fields.

Operator Interface

FORMS-2 is written in COBOL and uses the extended ACCEPT and DISPLAY AIX VS COBOL features. These two verbs are described in the *Language Reference*, as are the cursor control features.

Advantages of this console interface are:

- Corrections can be directly overtyped.
- Numeric fields accept only numeric characters.
- The full stop or period (.), when typed in a numeric field, automatically zero-fills the field from the left.

You have the ability to move the cursor quickly and easily about the display screen. The functions of the cursor control keys are summarized in Table 12-1.

Keys	Function
→	Position cursor right one data character
←	Position cursor left one data character
↓	Position cursor down at start of next line
↑	Position cursor up at start of previous line
HOME	Move cursor to start of first line
TAB	Position to next tab stop

You can correct text either by overtyping or by switching into command mode and using the editing commands.

FORMS-2 Validation

After you install FORMS-2, check that you have installed the components correctly by going through the following validation sequence:

1. Enter:

forms2 ↵

2. The program runs and shows the first display screen as follows:

```
FORMS2 V1.3           INITIALIZATION PHASE           SCREEN 101

FORMS2 PARAMETERS:

  DATA-NAME & FILE-NAME [  ] (1-6 alphanumeric characters)

  CRT lines               [24] (22 or 23 or 24)

SPECIAL-NAMES clause:

  CURRENCY SIGN          [$] (ANSI currency signs only)

  DECIMAL-POINT          [.] ("period." or ".")

Press RETURN when complete
```

FORMS-2 asks for file names and data-names. You must answer the question on console size if your console is a nonstandard size. Otherwise, FORMS-2 accepts all the default replies (the values inside []). You need only type a name (for example, "demo") followed by ↵.

3. FORMS-2 then shows display screen I02 to request the output file option type and directory prefix:

```
FORMS2 V1.3           INITIALIZATION PHASE           SCREEN I02

FILES TO BE CREATED:

FILE COMBINATIONS    [C]           (A = DDS)
                                     (B = DDS & CHK)
                                     (C = DDS & CHK & Snn)
                                     (D = DDS & Snn)
                                     (E = Snn)
                                     (F = No files output)
                                     (G = DDS & Snn & GEN)

DEVICE/DIRECTORY PREFIX (0-40 Chars) [           ]

Press RETURN when complete
```

Type **F** for the output file selection and press **↵**. If you wish the files output by FORMS-2 to be stored in a directory other than your current directory, enter the path name in the field marked **DEVICE/DIRECTORY PREFIX**. You must provide the final slash in the prefix.

4. FORMS-2 shows display screen W01 to request the display screen type option, as follows:

```
FORMS2 V1.3          WORK PHASE          SCREEN W01

WORK SCREEN SELECTION:

SCREEN TYPE  [A]      (A = Fixed text on clear screen)
                (B = Fixed text on last screen)
                (C = Variable data redefines last screen)
                (D = Variable data without redefinition)
                (! = Complete this FORMS run)

Fixed Text allows:  All characters

Variable Data allows: X or Y to define alphanumeric fields
                    9 or 8 to define numeric fields
                    edit chars to define numeric edit fields

Press RETURN when complete
```

Note the default, A, and press \leftarrow .

5. FORMS-2 shows a blank display screen. You are currently in edit mode, and should be able to position the cursor at any point on the display screen. Use the cursor control keys and the normal character keys to set up the following text on the display screen:

```
Name [-----]
```

Press \leftarrow .

6. FORMS-2 puts “_” in the top left of the display screen, indicating that you are now in command mode. At this point you would usually proceed directly to step 16. However, now type ? and press \leftarrow .

7. FORMS-2 shows display screen H01:

```
FORMS2 V1.3                HELP SCREEN                SCREEN H01

GENERAL COMMAND SUMMARY:
    SPACE = Process the work screen
    _     = Reenter EDIT mode
    ?     = Display the next HELP screen
    ?n    = Display the nth HELP screen
    Q     = Reenter WORK PHASE screen selection
    !     = Terminate FORMS run immediately
    X     = Position commands at EDIT mode cursor
    *     = Indicate Index Form's data area start

NOTE:  SPACE is the command to process the EDIT mode screen

HELP option [ _ ]  ( _ = Reenter EDIT mode)
                  (? = Display next HELP screen)
                  (! = Abandon FORMS2 run immediately)

Press RETURN when complete
```

Type ?, then press \leftarrow .

8. FORMS-2 shows display screen H02:

```
FORMS2 V1.3                HELP SCREEN                SCREEN H02

MANIPULATION COMMAND SUMMARY:
    F  = Invoke FOREGROUND/BACKGROUND manipulation
    Fx = Invoke FOREGROUND/BACKGROUND option "x"
    O  = Turn on automatic WORK screen preparation
    O1 = Turn off automatic WORK screen preparation
    Cn = Insert n spaces at cursor position
    Dn = Delete n chars at cursor position
    In = Insert n blank lines before cursor line
    Kn = Delete n lines including cursor line
    An = Overwrite n lines with data of cursor line
    Un = Move cursor up n lines
    Vn = Move cursor down n lines

HELP option [ ] (_ = Reenter EDIT mode)
              (? = Display next HELP screen)
              (! = Abandon FORMS2 run immediately)

Press RETURN when complete
```

Type ?, then press **↵**.

9. FORMS-2 shows display screen H03:

```
FORMS2 V1.3                HELP SCREEN                SCREEN H03

PROGRAMMING COMMAND SUMMARY:
    G = Give data names screen coordinates suffix
    G1 = Give data names sequential number suffix
    Jn = Allow up to n consec. spaces in fixed text
    Mx = Interpret "x" as "space"
    S = Cancel previous Sn command
    S1 = Inhibit DDS & CHK output at next processing
    S2 = Inhibit Snn output at next processing
    S3 = Prompt for Snn file name at next processing
    S9 = Line edit DDS output at next processing
    P = Display cursor position coordinates

HELP option [ ] ( _ = Reenter EDIT mode)
              (? = Display next HELP screen)
              (! = Abandon FORMS2 run immediately)

Press RETURN when complete
```

Type ?, then press **↵**.

10. FORMS-2 shows display screen H04:

```
FORMS2 V1.3           HELP SCREEN           SCREEN H04

WINDOW COMMAND SUMMARY:
    W = Position cursor to current window start
    W1 = Start window at cursor line
    W2 = End window at cursor line
    W3 = Start window at cursor line, no delimiters
    W4 = End window at cursor line, no delimiters
    W5 = Display start window delimiters
    W6 = Display end window delimiters
    W7 = Redisplay data overwritten by start delimiters
    W8 = Redisplay data overwritten by end delimiters
    W9 = Position cursor to current window end

HELP option [ ] ( _ = Reenter EDIT mode)
              (? = Display next HELP screen)
              (! = Abandon FORMS2 run immediately)

Press RETURN when complete
```

Press **←**.

11. FORMS-2 displays again the fixed text that you keyed in at step 5. Press **←**.

12. FORMS-2 puts “_” in the top left of the display screen. Type **F** then press **←**.

13. FORMS-2 shows display screen W02:

```
FORMS2 V1.3                WORK PHASE                SCREEN W02

FOREGROUND/BACKGROUND OPERATIONS:

OPTION [ ] (A = Reenter EDIT MODE)
          (B = Clear FOREGROUND)
          (C = Clear BACKGROUND)
          (D = Merge BACKGROUND into FOREGROUND)
          (E = Merge FOREGROUND into BACKGROUND)
          (F = Merge screen image into FOREGROUND)
          (G = Merge screen image into BACKGROUND)
          (H = Display FOREGROUND)
          (I = Display BACKGROUND)
          (J = Display screen image)

NOTE:      (H & I & J display until RETURN pressed)

FILE NAME  [                ]
           (F & G & J only)

Press RETURN when complete
```

- Type A and press \leftarrow .
14. Again FORMS-2 displays again the fixed text entered at step 5. Press \leftarrow .
15. FORMS-2 puts “_ _” in the top left of the display screen. Press the space bar and then \leftarrow .
16. FORMS-2 shows display screen W01 again to prompt for the display screen type option. The default is C. Press \leftarrow .
17. FORMS-2 shows the fixed text display screen. Use the cursor control keys and key in Xs alone to set up the display screen:
- ```
Name [XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX]
```
- Press  $\leftarrow$ .
18. FORMS-2 puts “\_ \_” in the top left of the display screen. Press the space bar and then  $\leftarrow$ .
19. FORMS-2 shows display screen W01 again. This time, type ! and press  $\leftarrow$  to complete the run.



---

20. FORMS-2 terminates with the following message:

END OF FORMS2 RUN

FORMS2 V1.3.8  
IBM AIX VS COBOL Compiler/6000 LP  
5601-258 (C) Copyright IBM Corp. 1987, 1990  
All Rights Reserved  
Licensed Material - Property of IBM

You have now used all the FORMS-2 display screens. Everything is in place and ready to be used.

---

## Initialization Phase

The first step when running FORMS-2 is the initialization phase. In this phase you specify the names of the files to use, the size of the display screen, and the type of output you want.

### Initialization Display Screen I01

Display screen I01 is the first screen shown when the FORMS-2 program is run. It asks you for the following information.

### Data-Name and File Name Base

The file name base that you type in now is used in the following ways:

- It is taken as the first part of all the data-names and record names generated in this run. Uniqueness is achieved by adding a two-digit sequence number for new records and adding the sequential number of the field within the form for data-names within records. Uniqueness may also be achieved by adding the display screen coordinates by means of a work phase command.
- It is taken as the main file name for generated files. These can consist of the following:

|                      |                                                          |
|----------------------|----------------------------------------------------------|
| <i>file-name.DDS</i> | AIX VS COBOL data description statements                 |
| <i>file-name.CHK</i> | Checkout programs                                        |
| <i>file-name.Snn</i> | Display screen images ( <i>nn</i> = 00, 01, 02, ..., 99) |
| <i>file-name.GEN</i> | Index programs                                           |

Only one DDS file is output per FORMS-2 run, whereas a separate display screen image file is output for each display screen built.

### Lines per Console Display Screen

You can use FORMS-2 with display screens of 22, 23, or 24 lines. The default for this entry is 24 lines. If your console has fewer than 24 lines, FORMS-2 will not function correctly with this value. With some consoles, you may have to specify one fewer than the number of lines actually present on the display screen to avoid having the display screen "scroll up" when an entry is made on the last line.

### Currency Sign

This entry allows you to override the default currency sign (\$). FORMS-2 will generate an appropriate SPECIAL-NAMES entry in either the checkout or index programs. Use the specified currency sign when specifying numeric-edited fields in the work phase, and FORMS-2 will use it in the generated data description statements.

The character that you specify here is not validated. See the *Language Reference* for a list of valid characters.

### Decimal Point

This option allows you to exchange the roles of the period or full-stop sign (.) and the comma sign (,). If you specify ",", FORMS-2 will generate a DECIMAL-POINT IS COMMA clause in the checkout or index programs. The default is ".". Use the decimal point sign when specifying numeric-edited fields in the work phase, and FORMS-2 will use it in the generated data description statements.

---

## Initialization Display Screen I02

Display screen I02 is shown immediately after you terminate display screen I01 entries by pressing  $\leftarrow$ .

FORMS-2 now prompts you for the following:

- Types of files to be created
- Directory into which files are to be written.

When you release display screen I02 by pressing  $\leftarrow$ , you enter the work phase. You can no longer amend information specified during the initialization phase.

FORMS-2 offers options for all valid combinations of the following types of files, each identified by a unique file name extension as follows:

- **.DDS**

You may generate AIX VS COBOL source data description statements (DDS) corresponding to the display screens you have created. These are output to a standard ASCII text file and may subsequently be compiled into any program using the standard COBOL COPY facility. In particular, they are used by the checkout and index programs.

If you are unfamiliar with display screen-handling in AIX VS COBOL, see the *Language Reference*, especially the ACCEPT/DISPLAY, FILLER, and REDEFINES sections.

- **.CHK**

In addition to generating DDS, FORMS-2 can also generate a checkout program. This consists of the Procedure Division statements (ACCEPT and DISPLAY) that correspond to the display screens that you have created. These statements are contained in *file-name.CHK*, and they are combined with the following COPY files:

*file-name.DDS*, FORMS2.CH1, FORMS2.CH2

The checkout program allows you to demonstrate on the display screen exactly how the system will operate by successively displaying the display screens you have just created, and by allowing you to enter data just as you would do under actual operating conditions.

- **.Snn**

You can also output the text of the display screen you have just designed to file on fixed disk in the form of a display screen image. You can retrieve this file later in this run or in subsequent FORMS-2 runs for further amendment if required.

Alternatively, you can print the display screen images and use the hard copies as a means of communicating between different individuals at different times (for example, the end user and the programmer).

- **.GEN**

FORMS-2 can generate an index program. This includes all the code necessary to set up and maintain an indexed sequential file with records corresponding to the structure of your form. The code is output to *file-name.GEN* and is combined with the following copy files:

*file-name.DDS*, FORMS2.GN1, FORMS2.GN2

---

Index program generation places certain constraints on you during the FORMS-2 run. The creation and operation of the index program is discussed in "Index Program" on page 12-39.

If you enter the **Q** command at this point, FORMS-2 will "quit" back to display screen I01, allowing you to amend the information given there. This can be useful if you inadvertently pressed **←** before completing display screen I01. This command is not mentioned on the display screen.

---

## Work Phase

You define the display screen layouts (forms) to be used in an AIX VS COBOL application by typing text at the keyboard to produce model forms on the display screen. You can define as many forms as you wish in a single FORMS-2 run. To define one form requires at least two work phases: one to define the fixed text of the form, and another to define the variable data entry fields.

Usually you will use the first work phase to specify the fixed text form and the second work phase to specify the variable data fields within the form; however, this need not always be the case. FORMS-2 requires information on which type of text you are going to input in a particular phase. Therefore, the work phase is introduced by a display screen presenting the various options (W01).

### Display Screen W01

Display screen W01 is shown immediately after you terminate display screen I02 by pressing **←**. Here you are prompted for the type of work screen you want to produce.

### Display Screen Type Selection

Fixed text selections offered at this display screen are as follows:

- A** The console is cleared to spaces in preparation for you to type the fixed text for a new form.
- B** The previous display screen is displayed again to assist you in defining additional fixed text.

Text from the previous display screen is used only as a background in this case, and is not included in the record definition for the fixed text you are about to type. You must therefore ensure that if any part of the previous display screen is inadvertently overtyped, the original characters must not be replaced but cleared to spaces.

Variable data selections offered are:

- C** The previous display screen is displayed again to assist you in the redefinition of the form to incorporate variable data field specifications. In the application the data is typed into the fixed text form itself.
- D** The previous display screen is displayed again to assist you in the definition of variable data area in the application program. This may sometimes be of assistance, even though it results in larger application programs.

---

## Terminating the Run

Display screen W01 is displayed again after completion of each work phase, and is the display screen used to terminate the program. You do this by typing ! and pressing **↵**.

**Warning:** Use of the ! command at any other time causes immediate abandonment of the run.

On termination, FORMS-2 closes the DDS file and displays an identification message. If you requested the checkout facility during initialization, FORMS-2 completes output of the checkout program to fixed disk, closes the CHK file, and displays an identification message.

FORMS-2 terminates automatically after the second work phase if an index program is being generated.

## Work Display Screen

After you have selected the display screen type, the appropriate work display screen for the text to be typed is displayed. For example, if you selected option A (fixed text on clear display screen), a blank display screen is shown. For the other options, the previous display screen is displayed again to allow correct alignment of the current input.

## Background/Foreground

To process only the data entered in this phase, FORMS-2 must keep this data separate from previously entered data, which is displayed purely for alignment purposes. FORMS-2 does this by constructing the displayed work screen from two separate data areas, termed background and foreground. The foreground holds the data entered during the current work phase. The background holds previously entered data that has been retained for alignment of the data entered in the current work phase. At the end of each work phase FORMS-2 processes the foreground data only.

If you select option B, C, or D, the foreground is overlaid on the current background contents when display screen W01 is next displayed. Then the foreground is cleared to spaces. If you select option A, both background and foreground are cleared to spaces.

In this way the new work display screen is prepared automatically. You can override this automatic work display screen in preparation for the next phase by means of a work display screen command and leave both areas unchanged.

Generally, you type text into the foreground from the keyboard, and it is moved into the background only from the foreground. The F work display screen command provides facilities for further manipulation of these areas. In particular, you can input a display screen image file from a previous run into the foreground, thus enabling you to amend existing forms.

---

While entering data onto the work display screen (that is, foreground), you can work in either of two modes:

- Edit mode, which is the mode in which you enter data to create the model form. The initial mode is always edit mode.
- Command mode, in which commands are available to assist you in creating and processing the edited work display screen.

## Edit Mode

Edit mode is the mode in which you are free to move the cursor to any part of the display screen by use of the cursor control keys. You may also make entries into any part of the display screen, in accordance with the display screen type that you selected at the start of this work phase.

**Fixed Text:** In the design of the fixed text of a form (that is, the fixed fields analogous to the preprinted text on a paper form), you can enter any legible characters anywhere on the display screen. This text will be displayed as “prompt” text during a data entry run of the application.

**Variable Data:** In designing the variable data fields of a form (that is, the fields analogous to the entry spaces on a preprinted form), you can type the characters X, Y, 8, and 9.

When typing in variable data, X denotes an alphanumeric character and 9 denotes a numeric character. If you need to have two alphanumeric fields contiguous with each other, place Ys in the character positions of the second field. Similarly, for contiguous numeric fields use 8s in the character positions of the second field.

Suppose in an application the operator must type in an invoice number. The fixed text in this example could be “INVOICE NO .....”. For example, an invoice number could be “CA3021”. You must define the size and type of this variable data explicitly. Therefore, if the invoice number always had two alphanumerics followed by four numerics, you would type XX9999 at the point on the display screen (the dots in this example) where you wish the operator to type the actual invoice number when the application itself is running. AIX VS COBOL provides automatic validation of numeric fields.

Additionally, you can input special editing characters to specify numeric-edited fields. These fields should be separated by spaces. Numeric-edited fields are described in the *Language Reference*. The valid characters are as follows:

Z, \*, +, -, CR, DB, ,(period), ,(comma), B, /, 0, \$

The \$ sign is the currency sign, which may be replaced by another sign as specified in the SPECIAL-NAMES clause of the AIX VS COBOL program, either directly or as specified during the initialization phase of the FORMS-2 run. The picture characters S, V, and P are not allowed.

FORMS-2 checks variable data fields for validity, but only when a DDS file is being created. See the *Language Reference* for information on how display screen-handling works.

---

## Command Mode

To switch to command mode from edit mode, press  $\leftarrow^j$ . Command mode is the mode in which two underline characters initially bound the cursor, and the cursor is constrained to stay within these two characters.

Invoke a command by typing the command and pressing  $\leftarrow^j$ . When execution of a command is complete, all commands (except **SPACE**, **1**, and **Q**) return you to edit mode.

The default command is the underline character ( ). This causes immediate reentry to edit mode.

The commands available to you during the work phase fall into three main groups.

1. General work display screen commands

General commands perform such functions as releasing the work display screen for processing.

2. Work display screen manipulation commands

Work display screen manipulation commands help you to prepare and edit the work display screen.

3. Programming commands

Programming commands have been introduced mainly for the convenience of the COBOL programmer, and some of them will not be meaningful without an understanding of COBOL. They include commands to assist in producing efficient code, and to give you more control over the output files.

Enter a command by typing the command character(s) and pressing  $\leftarrow^j$ .

Commands in groups 1 and 2 are summarized within Help display screens 1 and 2 (H01 and H02). Commands in group 3 are summarized on Help display screens 3 and 4 (H03 and H04).

**General Work Display Screen Commands:** A description of each general command is given below.

**?** Display help screens

If you type **?** and then press  $\leftarrow^j$ , the first Help display screen, which includes a summary of the general commands, is shown. This display screen remains until the next input command is entered.

Pressing  $\leftarrow^j$  alone at this stage returns you to edit mode.

Typing **?** and then pressing  $\leftarrow^j$  again displays the next Help screen, which is a summary of the work display screen manipulation commands. If you repeat the sequence, each help display screen will appear in sequence until the end of the series, when the first display screen appears again.

The **?** command is also available from the first display screen, W01.

**?n** Display help screen *n*

If you type **?** followed by a numeric digit and then press  $\leftarrow^j$ , the *n*th help display screen in the series will be shown.

---

**(SPACE)** Terminate the work phase

If you press the space bar, you will terminate the current work phase and initiate processing of the data you have just entered on that work display screen.

**\_ (underline)**

Return to edit mode (default).

To exit from the command mode back to edit mode, enter the underline character.

The default command was introduced for convenience in case you enter command mode inadvertently.

**Q** Quit

If you type **Q**, you return to display screen W01.

The current foreground/background components of the work display screen are unchanged when you reenter edit mode, regardless of the text type you then select.

The most likely use for this command is where you select the default option **C** (variable data fields) at display screen W01 and then use the work display screen incorrectly to set up a fixed text form. Validation errors then occur, and you are returned automatically to edit mode. To return to display screen W01 and correct the selection without loss of the text you have just typed, press **←** to enter command mode and type **Q**. You can now correct the display screen type, and the work display screen has been preserved for reprocessing (as fixed text). **Q** is also available at display screen I02, though it is not mentioned there, and its function is to return you to a step within the phase (in this case back to display screen I01).

**!** Terminate FORMS-2 run

If you type **!**, you will terminate the program. This command is available throughout the FORMS-2 program, but you will normally use it on automatic return to display screen W01 when you have completed a work phase. When used elsewhere, it abandons the run.

**X** Reposition command area

The standard command area is in columns 1 and 2 of line 1, as indicated by the two underline characters displayed on entry to command mode. FORMS-2 always attempts to restore any data in these positions upon return to edit mode. However, to enable these positions to be preserved intact at all times, the facility to reposition the command area is provided. To do this, place the cursor at the desired location before entering command mode. Type **X** to cause the required change. The next time you enter command mode, the prompting underline characters will appear at the new specified location.



---

**\* Define key/data split for index program**

This command is used only in connection with generation of the index program. If you select the index program option **G** at display screen **I02**, you should position the cursor at the first nonkey variable data position and enter **\*** before termination of the second work phase.

If you do this incorrectly, **FORMS-2** will continually return you to edit mode until you either do it correctly or type **!** to terminate the session and abandon the work in progress.

**Work Display Screen Manipulation Commands:** **F** and **O** are preparation commands.

**F Invoke foreground/background menu display screen (W02)**

Display screen **W02** contains options to assist you in setting up the foreground component of the work display screen.

The options are:

- A** Return to edit mode.
- B** Clear foreground to spaces.
- C** Clear background to spaces.
- D** Overlay background data onto foreground.
- E** Overlay foreground data onto background.
- F** Overlay a display screen image file onto foreground. You can amend forms defined earlier in this run or in previous runs with this option. You are prompted for the name of the required file.
- G** Overlay a display screen image file onto background.
- H** Show foreground. This displays just the foreground component of the work display screen for examination. The full work display screen will be restored on return to edit mode.
- I** Show background. This displays just the background component of the work display screen for examination.
- J** Show a display screen image file. You are prompted for the file name and the specified file is displayed, but without corrupting the current contents of either foreground or background. This enables you to make a check prior to using option **D**.

The options **H**, **I**, and **J** produce a display that remains until you press **↵**.

**Fx Specifies required foreground/background option**

*x* is the option code as contained in display screen **W02** above. The specified option is executed and control returned to edit mode without display of screen **W02**.

**01 Switch off automatic background/foreground preparation**

The background/foreground preparation sequence is described earlier. Use this command to prevent the current foreground from being merged into the background or from being cleared for the next phase.

The **01** command remains effective until you enter **0**.

---

**0** or **00**    Reset background/foreground preparation

The background/foreground preparation sequence is reset to automatic (starting at the beginning of the next work phase).

**G** has a similar effect, beginning at the next phase.

**C**, **D**, **I**, **K**, and **A** are editing commands and are controlled by the position of the cursor at the time command mode is entered (that is, the current cursor position) and operate only on the foreground data. Background data remains in the same position.

**Cn**            Insert *n* spaces

Insert *n* (1-9) spaces before the character at the current cursor position. Only the current line is affected.

**Dn**            Delete *n* characters

Delete *n* (1-9) characters including the character at the current cursor position. Only the current line is affected.

**In**            Insert *n* blank lines

Insert *n* (1-9) blank lines before the line containing the current cursor position, irrespective of the column. You can only insert whole lines.

**Kn**            Delete *n* lines

Delete *n* (1-9) lines including the line containing the current cursor position. You can only delete whole lines using this command.

**An**            Repeat current line *n* times

Repeat the line containing the current cursor position *n* (1-9) times.

This does not act as an insert. Any foreground data in the next *n* lines will be overwritten.

**U** and **V** are cursor positioning commands.

Position the cursor horizontally using the **Cursor right** and **Cursor left** keys.

Position the cursor vertically with the **Cursor up** and **Cursor down** keys after you have moved the cursor to the first position of the current line.

Vertical tabulation within the same column may be required when setting up a form. Two tabulation commands are:

**Un**            Move cursor up *n* lines

This command moves the cursor up *n* (1-9) lines from the current cursor position. Cursor position within the line is maintained.

**Vn**            Move cursor down *n* lines

This command moves the cursor down *n* (1-9) lines from the current cursor position. Cursor position within the line is maintained.

---

**Programming Commands:** **G** is the data-name structuring command.

The default record name format generated by FORMS-2 for inclusion in your AIX VS COBOL source program for display screen formatting is:

*bbbbbb-rr* (01 level)

where:

*bbbbbb* is the 1-6 character base that you specified at display screen I01 and *rr* is the record number, starting at 00 in the first work phase and increasing by one for each subsequent work phase.

If you use the window commands to define a window starting in a line other than line 1, the default record name generated is:

*bbbbbb-rr-ll*

where *ll* is the line number.

The default elementary data-name structure generated by FORMS-2 for inclusion in the AIX VS COBOL source program for display screen formatting is:

*bbbbbb-rr-nnnn*

where *nnnn* is the sequence of this field within the display screen, starting at 0001.

Alternatively:

- **G (G0)** causes *nnnn* within the data-name to be the display screen coordinates of the start of the field. This can sometimes be of use as a reference guide when using AIX VS COBOL facilities to set cursor position.
- **G1** restores the default data-name generation to using sequential field numbers.

**J** and **M** are multiple spaces and FILLER commands.

The AIX VS COBOL interactive ACCEPT and DISPLAY verbs operate only on named fields; FILLER areas are left alone. The time taken to show a display screen depends both on the size and the number of constituent fields.

When processing fixed text display screens, FORMS-2 generates FILLER wherever multiple spaces appear. On some forms this can result in many small fields separated by small FILLER fields. The problem can be alleviated by:

**Jn**            Reset multiple spaces

**Jn** resets the number of contiguous spaces FORMS-2 will allow within the VALUE clause of a named field. This is initially set to 1 (*n* can be 0-9).

**J** or **J0** will force FILLERS even for single spaces.

An alternative method of forcing spaces within named fields is by use of the underline; its use in a field results in an actual space in the corresponding position in the generated VALUE clause.

If you need to change the designated character from underline to something else (presumably because you need to generate VALUE “\_”), use **M** as follows:

- 
- Mx** Change default FILLER
- Mx** changes the default “  ” character (underline) to that specified by *x*. If **SPACE ( )** is specified, this will force generation of named fields for the entire display screen without any FILLERS.
- Sn** File output control command
- S (or S0)** Cancels any other **Sn** commands in effect at the time.
- S1** Suppresses DDS (and CHK) text generation for this work display screen. Generation of this text resumes for the next work display screen unless the same command is repeated in the next phase.
- S2** Suppresses display screen image (*Snn*) text generation for this work display screen. This is commonly used to suppress display screen images of variable data fields. Again, the effect only lasts for the current phase.
- S3** Enables you to override the default display screen image file identifier for the current work display screen. Normally, if a file already exists with the default identifier, you are given the option of overriding it. If you reject this option, you are prompted for an alternative file identifier. This command forces the alternative file identifier to be requested even when no file exists with the default identifier.
- S9** Causes FORMS-2 to halt after display of each line of code during DDS generation. FORMS-2 re-ACCEPTs the line before outputting it to the .DDS file. This provides you with a limited editing capability that may prove useful under special circumstances. This option is not available if you select option **C** or **B** at display screen I02.
- P** Cursor control command
- Causes the coordinates of the current cursor position to be displayed at the command area position. This display lasts a few seconds, after which the contents of the work display screen at the command area are restored and control is returned automatically to edit mode. Where sequential field numbers are used within data-names, this command provides you with an easy alternative method of ascertaining the coordinates of any field.
- Wn** Window commands
- The “window” defines the area (full lines) to be processed by FORMS-2 when generating DDS text. By default the window is the full display screen. Where window start or end is other than the start or end of the display screen, a delimiting line of hyphens may optionally be displayed on the line just outside the window. For example, if a window starts in line 4, delimiters appear along the length of line 3.
- The principal use of the window is to allow you to create a form that begins below the top of the display screen but saves memory by avoiding the description of blank lines at the top of the display screen.

---

Where window is used in this way the generated record name incorporates the start line number of the window, which can then act as a guide to the programmer, using the AIX VS COBOL ACCEPT/DISPLAY AT coordinates facility.

The detailed commands give you very comprehensive window formatting capability, as follows:

- W (or W0)**      Positions the cursor at current window start. This is the equivalent of the **HOME** key when the window facility is in use.
- W1**              Sets start of window to current line with delimiters on previous line.
- W2**              Sets end of window to end of current line with delimiters on next line.
- W3**              Sets start of window to current line without delimiters.
- W4**              Sets end of window at end of current line without delimiters.
- W5**              Displays delimiters preceding current window start.
- W6**              Displays delimiters following current window end.
- W7**              Erases start delimiters and restores any work display screen data to the display.
- W8**              Erases end delimiters and restores any work display screen data to the display.
- W9**              Positions the cursor at current window end.

Delimiters do not corrupt background/foreground contents.

The current display screen image will include the full foreground part of the work display screen without delimiters, regardless of whether a window has been defined. You could use this to include annotation on the display screen image that does not affect DDS generation.

One use of the window facility is to display a form in two stages: the first 10 lines, followed by the second 10 lines. You can create this as a single display screen image including both sections of the form, and you can "window in" on the relevant portions as required when the DDS text is generated.

---

## Work Phase Completion

To complete the work phase of FORMS-2, select command mode, press **SPACE**, and then **↵**. **SPACE** is the command to release the work display screen for processing.

FORMS-2 completes the work phase (depending on the file selection at display screen I02):

1. If this is a variable data field definition work phase (option C or D at display screen W01), validation occurs with the message:  

```
WORK SCREEN VALIDATION in progress
DO NOT press RETURN.
```
2. If you selected DDS file generation at display screen I02, the source code produced is echoed to the display screen as it is written to fixed disk. If you used the **S9** command, processing stops after each line of code to enable you to make changes as required. This is recommended only if special requirements dictate its use.
3. If you requested a display screen image file at display screen I02, the display screen image is echoed to the display screen as it is written to the fixed disk file. The name of the created file is displayed. Press **↵** to continue.
4. Display screen W01 is displayed again so the run can be terminated or continued.

During validation of variable data only those characters listed in the description of text types are permitted (plus space). If any other character is encountered, the validation routine signals an error by alternately displaying “?” and the offending character to give a flashing effect. This error indication then ceases and FORMS-2 returns to edit mode with the cursor positioned under the erroneous character. You must repeat the **SPACE** command after making any corrections.

FORMS-2 will allow you to edit characters but will not verify that the combinations of these are valid; AIX VS COBOL editing rules must therefore be obeyed to ensure error-free code. These fields should be separated by spaces.

Only foreground data is output to the display screen image file.

---

## Data Descriptions

The AIX VS COBOL data descriptions that FORMS-2 generates in the .DDS file are described in the following text.

The AIX VS COBOL extensions to the ACCEPT and DISPLAY verbs allow comprehensive display screen-handling to be included in a user application. See the *Language Reference* for more information. Programming the necessary data description statements can be tedious and expensive in terms of programmer time, particularly since it is prone to simple errors.

FORMS-2 simplifies the production of error-free data descriptions by allowing you to specify display screen layouts (forms) in the most convenient way, namely by setting them up on the display screen. If you invoke the facility by selecting an appropriate option at display screen I02 during the initialization phase, FORMS-2 automatically converts this input to the necessary AIX VS COBOL statements and outputs these to a DDS file. You then incorporate these statements in your application source code by means of the AIX VS COBOL COPY verb and use record names consistent with those generated by FORMS-2.

## Record Name and Data-Name Generation

Initialization display screen I01 prompts you for a base name. This is a 6-character field into which you enter any name of your choice consistent with COBOL data naming. This base is then used to generate the COBOL data-names.

For detailed information on record naming and data naming, see "Programming Commands" on page 12-25.

## Record Naming

The default record name format generated by FORMS-2 for inclusion in your AIX VS COBOL source program for display screen formatting is as follows:

```
bbbbbb-rr (01 level)
```

where:

*bbbbbb* is the one- to six-character base that you specified at display screen I01. *rr* is the record number, starting at 00 in the first work phase and increasing by one for each subsequent work phase.

If you use the window commands to define a window starting in a line other than line one, the record name generated will be as follows:

```
bbbbbb-rr-ll
```

where *ll* is the line number. This serves as a useful reminder when coding the appropriate ACCEPT/DISPLAY statements.

## Data Naming

The elementary data naming structure generated by FORMS-2 for inclusion in your AIX VS COBOL source program for display screen formatting is as follows:

```
bbbbbb-rr-nnnn
```

where *nnnn* is the sequence of this field within the display screen, starting at 0001.

---

Sometimes it may be more convenient to have the display screen coordinates incorporated in the data-name rather than a field sequence number. You can do this by using the **G** command during the work phase.

## Picture Generation

Generation of **PICTURE** clauses by FORMS-2 depends on the type of text you select at display screen **W01** at the start of each work phase. FORMS-2 will force field boundaries at the end of each line in order to be compatible with certain types of consoles.

## Fixed Text

At the end of a fixed text work phase FORMS-2 generates only **FILLER** areas or named alphanumeric fields with associated **VALUE** clauses.

The AIX VS COBOL interactive **ACCEPT** and **DISPLAY** verbs operate only on named fields; **FILLER** areas are left alone. The time taken to show a display screen depends both on the size and also the number of constituent fields.

When processing fixed text display screens, FORMS-2 generates **FILLER** wherever multiple spaces appear. You can alter this default by using the **J** command. Alternatively, you can use **\_** (underline) to force inclusion of spaces within a **VALUE** clause. You can alter the default character used for this purpose by using the **M** command.

## Variable Data Fields

At the end of a variable data work phase, FORMS-2 generates alphanumeric, numeric, or numeric-edited fields depending on the actual characters that you typed in. These are usually the AIX VS COBOL characters **9** and **X**, but you can use **8** and **Y** as alternatives to **9** and **X**. Note also the exclusion of **S**, **V**, and **P**.

## Editing the DDS File

Normally the DDS output from FORMS-2 should be all that you require. Where special circumstances dictate the use of particular data-names or the disallowed pictures characters, the **S9** command will allow you to edit DDS lines prior to output. Alternatively, you can use a conventional text editor to edit the file. However, this editing process must be repeated if you amend the forms using FORMS-2.

You can also completely suppress the DDS output for a particular work phase by using the **S1** command. If you use this, the record number incorporated in data-names will be stepped up by 1 for the next work phase.

## Incorporation of DDS File Contents

To incorporate the generated data descriptions into your application program, you need only copy in the DDS file using the **COPY** statement available in AIX VS COBOL.

The **COPY** statement to incorporate the demo1 sample forms illustrated in "FORMS-2 User Display Screen Generation Example" on page 12-35 is:

```
000000 COPY "demo1.DDS".
```

and would be coded within the Data Division.

This statement is included in all checkout or index programs generated, and you can refer to any of these for an example.



---

## Checkout Program

The checkout program that FORMS-2 can generate automatically while generating the created forms enables you to do the following:

- Validate the DDS file.
- Demonstrate the operation of the proposed application.
- Check the use of your forms for data entry.
- Check the use of your forms for data amendment.

The checkout source code, which is in AIX VS COBOL, includes a COPY statement for the DDS file exactly as it would be coded in your application, and is therefore a true validation of the DDS file when compiled.

If you include numeric-edited fields in the variable data fields of a form, error free code is not guaranteed with standard AIX VS COBOL. Compilation is necessary to fully validate numeric-edited fields. If numeric-edited fields do cause compilation errors, you can use the FORMS-2 display screen image facility to recall the offending display screen and alter the variable text numeric-edited fields as necessary.

### Checkout Program Generation

The checkout program logic is a sequence of DISPLAY or ACCEPT statements for the display screens that you defined in the FORMS-2 run in the order in which they were created. A demonstration program using all forms can be created rapidly without any programming by entering all required forms in a single FORMS-2 run. For a complex application, the best method is to create each form in isolation, using only display screen image output. FORMS-2 can then be run again to produce the required checkout program, using the facility to re-input display screen images (use the F command and the D option in the subsequent display screen). Use of this facility also enables you to set up a complex sequence of display screens for demonstration purposes, incorporating the same display screen more than once.

After the sequence of display screens, checkout gives you the option of repeating the entire sequence. On the second pass previously entered data is displayed again, allowing you to check your forms for both initial data entry and data amendment.

### Checkout Program Compilation

Compile the checkout program in the usual way by entering the following:

```
cob -ik CHK basename.CHK ↵
```

See Chapter 4, “The COBOL Interface” for information on submitting programs to the AIX VS COBOL using **cob**.

### Checkout Program Running

Run the executable program output by the **cob** command by entering:

```
cobrun basename ↵
```

---

## Checkout Processing

The basic function of the checkout program is to display the fixed text fields of your form so that data can be entered into the variable data fields of the form in the sequence in which the display screens were created.

However, the detailed logic is slightly more sophisticated. The following notes make references to the options taken for display screen type at display screen W01.

### Fixed Text Display Screens

The fixed text of a form is displayed. If there are two consecutive fixed text forms, the checkout program pauses after the first display until you press **↵**.

1. Fixed text on a clear display screen

If you selected option **A** when you created the form, checkout clears the console before the display screen.

2. Fixed text on last display screen

If you selected option **B** when you created the display screen, any text displayed remains on the console, except where it is overwritten by the text of the new display screen.

### Variable Data Display Screens

An ACCEPT statement is issued for a variable data display screen, allowing you to enter data in the unprotected areas (that is, the fields specified by Xs and 9s, and so on).

You can check the extents of the fields. For numeric fields you can also check that only numeric characters may be entered, and the effect of entering the left zero-fill character “.”. See the *Language Reference* for information on use of the “.” character.

On other than the first pass through the sequence of display screens, the previously entered data is displayed again before the ACCEPT is issued.

If the variable data display screen includes numeric-edited fields, the ACCEPT for the display screen is followed by a corresponding DISPLAY to show the effect of the editing or normalization by the AIX VS COBOL Run Time Environment. The normalized fields are not automatically echoed to the console.

## Checkout Completion

After the entire sequence of display screens has been passed, the checkout program displays:

```
CHECK-OUT completed
Repeat ? [N] (Y=Yes)
```

If you wish to repeat the sequence of display screens, type **Y** and press **↵**. Otherwise, press **↵** to take the default to terminate the program.

---

## Display Screen Image File

FORMS-2 can generate a display screen image file that contains exact text images of the forms that you have designed. These form images can:

- Provide the basis for amendments to the form
- Be printed to yield printed copies of the form
- Provide a means of communication between the system designer and the applications programmer.

## Display Screen Image File Generation

Invoke this facility by selecting an appropriate option at display screen I02 during the initialization phase. The default option will cause display screen image output.

Display screen images are output to files named:

*basename.Snn*

where *basename* is the name that you entered in the initialization phase (*nn* is a number 00-99).

You can override the default file name by issuing the **S3** command during the work phase. This causes FORMS-2 to request input of the required file name during processing of this work display screen.

A separate file is created at the end of each work phase. The numeric part of the name (*nn*) is incremented by one each time. A display screen image file is structured as a standard line-sequential file with a record for each line of the display screen. Each display screen image contains only text entered during the work phase in which it is generated (that is, foreground data). Consequently, for a variable data work phase the output display screen image contains only Xs, 9s, Ys, and 8s.

You can suppress the display screen image output from any work phase by issuing the **S2** command during that phase. If you use this command, the numeric part of the file name extension will still be updated for the next phase to keep in line with the record numbering within the generated data description statements (DDS).

## FORMS-2 Maintenance

AIX VS COBOL data description statements that have been generated from a user-designed form by FORMS-2 are held in a DDS file. You will probably need to make corrections and adjustments to maintain the form. You can maintain a DDS file using a conventional text editor, but this involves the high risk of simple but expensive errors, which FORMS-2 eliminates.

Your form is output to a display screen image file as an exact image, and FORMS-2 provides you with the facility to read display screen images back from fixed disk to allow you to amend them. You can do this by running FORMS-2 and issuing the **FF** command once the first work display screen is reached. You are then prompted for the identity of the display screen image required. FORMS-2 reads the screen image file into the foreground area of the work display screen and returns you to edit mode. The form is then displayed as if it had just been typed, and you can make any required amendments before releasing the display screen for processing by pressing the space bar to get the **SPACE** command.

---

When you use FORMS-2 for maintenance, it will overwrite the existing files, but only after issuing warnings that the files already exist, and asking you for confirmation to proceed. For display screen image files, FORMS-2 offers you the facility of specifying an alternative file name if you wish to retain the old version.

## **Printed Forms**

The display screen image files are created as line-sequential files in accordance with the conventions of the operating system. Therefore, you can use standard software to print them, and the resultant hard copy will be an exact image of your form with no risk of transcription error.

## **Form Images in the Design Process**

Form images can be used as a step in the applications design process, providing a valuable part of the designer/programmer interface.

For interactive applications, design of the user interface (that is, the display screen layouts or forms) may take place well in advance of the actual program being written, and the forms designer need not have any detailed knowledge of COBOL.

FORMS-2 enables a nontechnical user to generate valid AIX VS COBOL statements. An experienced COBOL programmer can make use of commands available to generate the most efficient code (for example, by influencing the number of fields to be displayed).

It may sometimes be advantageous to use display screen image output alone as an intermediate stage in the design process, with the programmer using the image files as input to FORMS-2 to produce the final DDS file. If you use FORMS-2 in this way, both fixed text and variable areas could be conveniently indicated on a single fixed text display screen. You can then use this display screen to generate the DDS file, and the forms designer need not know any details of COBOL data field specifications.

---

## FORMS-2 User Display Screen Generation Example

---

In this example, FORMS-2 is used to build the data entry form:

```
NAME []
ADDRESS []
 []
 []
TEL [-]
```

NAME and ADDRESS are alphanumeric fields and TEL is a numeric field. Use the checkout program generated by FORMS-2 to experiment with data entry. Change the ADDRESS field name to ABODE. Afterwards, the display screen will appear as follows:

```
NAME []
ABODE []
 []
 []
TEL [-]
```

To do this, you must follow these steps:

1. Invoke FORMS-2 with the command:

```
forms2 ␣
```

2. FORMS-2 shows display screen I01 requesting a 6-character base for file names and data-names followed by four other questions. If the console is standard (24 lines), no further questions need be answered for this display screen. Type **demo1** and press ␣ if the default display screen size is correct.
3. FORMS-2 shows display screen I01 to prompt for the output file option type and device/directory prefix. Press ␣ to accept the default values.
4. FORMS-2 shows display screen W01 to prompt for the display screen type option. Note the default option A, and press ␣.
5. FORMS-2 shows a blank display screen. Use the cursor control keys and the normal character keys to set up the following text on the display screen:

```
NAME [-----]
ADDRESS [-----]
 [-----]
 [-----]
TEL [-----]
```

The underline characters are treated as spaces when the form is displayed by an applications program, and are provided only for your convenience for indicating the fields to be redefined as variable fields.

Press ␣.

6. FORMS-2 puts “\_ \_” in the top left of the display screen. Press SPACE and then ␣.
7. FORMS-2 processes the display screen to create a fixed text form. FORMS-2 displays the DDS source code as generated, followed by a redisplay of the fixed text as it is written to the display screen image file.

A message is then displayed giving the name of the fixed text display screen image file that is created. Press ␣ as directed.

8. FORMS-2 shows display screen W01 to request the display screen type option. Note the default C, and press `↵`.
9. FORMS-2 shows the fixed text display screen as background data. Use the cursor control keys and enter Xs and 9s alone to set up the display screen as follows:

```

NAME [XXXXXXXXXXXXXXXXXXXXX]
ADDRESS [XXXXXXXXXXXXXXXXXXXXX]
 [XXXXXXXXXXXXXXXXXXXXX]
 [XXXXXXXXXXXXXXXXXXXXX]
TEL [999-9999]

```

Press `↵`.

10. FORMS-2 displays "--" in the top left hand corner of the display screen. Press `SPACE` and then `↵`. There is a short pause while FORMS-2 validates the display screen content, during which the following message is displayed:

```

WORK SCREEN VALIDATION in progress
DO NOT press RETURN

```

11. FORMS-2 processes the Xs and 9s to create a variable data form displaying the DDS source code as generated, followed by a redisplay of the variable text as it is written to the display screen image file.

A message is then displayed giving the name of the variable data display screen image file created. Press `↵` as directed.

12. FORMS-2 shows display screen W01 again. Enter ! and press `↵` to terminate the run. FORMS-2 displays the names of the DDS and CHK files created and displays an End of Run message.

13. Compile the checkout program by entering the following:

```
cob -ik CHK demo1.CHK ↵
```

14. When the compilation is finished, check the two display screens by typing:

```
cobrun demo1 ↵
```

15. The demonstration program will then run. The fixed data form is shown on the display screen. The variable data form is used to accept data.

Satisfy yourself that the cursor can only be placed in the variable fields, and that the data accepted into the fields depends on whether X or 9 was specified. You may also test the effect of left fill character ".".

When satisfied, press `↵` to complete. A message is displayed:

```

CHECK-OUT completed
Repeat ? [NO] (Y=Yes)

```

Press `↵` to accept the NO default and complete.

16. The checkout program displays:

```
END OF FORMS-2 CHECK-OUT
```

The variable form is used in the demonstration for ACCEPTing data. In practice the form can be used for DISPLAYing data as well as ACCEPTing it. The demonstration shows the extent and type of each field, which will be the same in DISPLAY as in ACCEPT. A useful technique for clearing just the variable data fields on the display screen is to move spaces to the ACCEPT record and then DISPLAY it.

- 
17. You can now examine the fixed disk files:

**demo1.DDS**  
**demo1.CHK**  
**demo1.S00**  
**demo1.S01**  
**demo1.int**  
**demo1.lst**

to check the output from FORMS-2 during this use.

You now know how to use FORMS-2 to create display screens of fixed and variable data automatically for inclusion in your AIX VS COBOL program.

If you continue with the next step through the end, you will learn to update both the fixed and variable data display screens already created by moving them from background to foreground.

18. Reload FORMS-2 by entering:

forms2

19. FORMS-2 shows display screen I01 prompting for the 6-character file and data-name base in step 2. Answer the questions as necessary in step 2, and press .

20. FORMS-2 shows display screen I02 prompting for the output file option type and directory; press  to accept the default values.

21. FORMS-2 displays the following message:

```
File already exists: demo1.DDS
 overwrite? [N] (Y=Yes)
```

Type **Y** and press .

If you enter the **NO** default here, the run is abandoned.

22. FORMS-2 displays the message:

```
File already exists: demo1.CHK
 overwrite? [N] (Y=Yes)
```

Type **Y** and press .

If you enter the **NO** default here, the run is abandoned.

23. FORMS-2 shows display screen W01 again. Press  to accept the default option **A**.

24. FORMS-2 shows a blank display screen in edit mode. Press  to enter command mode, then **F** followed by  to invoke the foreground/background selection display screen. (You want to update your form, so it must be in foreground.)

25. FORMS-2 shows the foreground option display screen. Type **F**, followed by the file name **demo1.S00**, and press .

26. FORMS-2 shows display screen W02 again. Select option **A** to return to edit mode.

27. FORMS-2 shows the fixed text display screen (previously created at step 5). Move the cursor to the word **ADDRESS** and overwrite it with **ABODE**. Remember to overwrite the extra characters **SS** with spaces, and then press .

28. Press **SPACE**, and then .

29. FORMS-2 shows a message reminding you that your altered fixed text display screen image is about to overwrite your previous display screen image in the file:

```
File already exists: demo1.S00
 overwrite? [N] (Y=Yes)
```

Type **Y** and press **↵**.

If you type the **NO** default here, you are prompted for a file identity for a new display screen image.

30. FORMS-2 shows the display screen image and then displays the file name:

```
File created = demo1.S00
```

Press **↵** to continue.

31. FORMS-2 shows display screen W01 with option **C** as default to enable specification of variable data fields. Press **↵** to accept the default.
32. FORMS-2 displays the altered fixed text as follows to assist in defining the variable fields:

```
NAME [-----]
ABODE [-----]
 [-----]
 [-----]
TEL [-----]
```

Press **↵** to enter command mode, then the **F** command, then **↵**.

33. FORMS-2 shows display screen W02 again. Type the option **F**, then the file named **demo1.S01**, then press **↵** to retrieve your variable text created at step 9.
34. FORMS-2 shows display screen W02 with option **H** as default. If you press **↵** to accept this default, FORMS-2 displays the current foreground contents. This is only the **Xs** and **9s** that define the variable data fields (the fixed text is in the background area). Press **↵** to re-invoke display screen W02.
35. FORMS-2 shows display screen W02 with option **A** as default. Press **↵** to accept this default.
36. FORMS-2 displays the whole form again. You can now alter the variable text fields, if required.

You have seen facilities to retrieve fixed text and variable text from previously created files. With a small number of variable data fields such as in this example it would, in practice, be easier to rekey them.

37. Press **↵**, then **SPACE**, then **↵** to process the altered form. Again, there is a pause while FORMS-2 validates the variable fields.
38. FORMS-2 produces the DDS file, then displays:

```
File already exists: demo1.S01
 overwrite? [N] (Y=Yes)
```

Type **Y** and press **↵**.

39. A message is displayed:

```
File created = demo1.S01
Press RETURN to continue.
```

40. Press **↵** FORMS-2 shows display screen W01 again. This time enter **!** and press **↵** to complete the run.



- 
41. Repeat steps 13 through 16 if you wish to run the checkout program again to verify the altered form.

---

## Index Program

FORMS-2 provides facilities for automatically generating a COBOL program to create and maintain an indexed sequential file. You supply the input required to generate the index program and use it to maintain files interactively through the console.

You can design a data entry display screen using FORMS-2 by specifying the fields that will comprise the indexed sequential file records in the usual fixed text and variable text work phases.

The user interface to the generated index program is the form you design that reflects the desired record structure. You need to consider only the data requirements.

You must have access to the AIX VS COBOL software to compile the source index program that FORMS-2 generates.

The generated index program is written to the file *file-name.GEN* and provides you with the following facilities required for creating and maintaining an indexed sequential file:

- Select records by key field for display (enquiry by key field).
- Select records sequentially for display (sequential entry).
- Amend existing records.
- Delete existing records.
- Insert new records.

The program is designed so that you do not have to explicitly state the facility to be invoked at any time; the program is able to follow the logic from the way you manipulate the data and cursor position.

Only the variable text data is written to the file, and the fixed text data is merely a template to enable each field to be entered separately at data entry time. A record in the indexed sequential file is constructed by linking the variable fields of the form in the order in which they appear.

The record must include a key area by which it can be uniquely accessed. The index program logic requires that this key area must be at the beginning of the record. That is, it must be the first integral field(s) in the form and must not exceed 32 characters in length. This key area constitutes part of the record data. For convenience, the remaining fields are known as the data fields.

Refer to “User Index Program Example” on page 12-43 to see how the sample application is adapted to create and maintain a file of names, addresses, and telephone numbers.

---

## Index Program Generation

You can generate an index program using FORMS-2. All existing FORMS-2 facilities are present, but logic is incorporated to prevent the use of inappropriate features if you select the index program option. The steps involved are as follows:

### 1. Initialization

#### a. Display Screen I01

Specify file and name data base as normal.

#### b. Display Screen I02

Specify option **G** for index program generation.

### 2. Work phase one

#### a. Display Screen W01

Work display screen selection. The program forces the default option **A** for fixed text entry by refusing to accept anything else, except **!** to abandon the run or **?** to display help screens.

#### b. Fixed text work display screen

A blank work display screen is shown for input of the fixed text form.

All FORMS-2 commands are available with the following exceptions:

**G** The generated program relies on the default data-name structure. This command is rejected.

**S** It would be inappropriate to switch off DDS generation, so this command is rejected.

**W** This feature is not available, and the command is rejected. However, the program reserves the bottom line for use in the generated program for system messages, and a delimiting line of hyphens marks this line.

Release the display screen for processing by the sequence **↵**, **SPACE**, **↵**, when the fixed text display screen has been completely entered.

The work display screen selection screen is again shown.

### 3. Work phase two

#### a. Display Screen W01

#### b. This time the program forces the default option **C**.

Specify the variable fields, that is, **X/Y/8/9** and editing characters. At some point before releasing this display screen you must define the end-of-key/start-of-data boundary within the record. You do this by positioning the cursor on the first data field, entering command mode and typing the **\*** command (that is, the sequence **↵**, **\***, **↵**).

A variable field cannot exceed 32 characters.

Release the display screen by the usual **↵**, **SPACE**, **↵** sequence. If the program is not satisfied with the specification of the key/data boundary, it will return to edit mode.

---

Upon completion of the variable text display screen, FORMS-2 completes its processing and terminates automatically without any need for the termination ! command. In fact, the ! command is only used to abandon the run when generating the index program.

## Index File Generation

The following files are written to the fixed disk by FORMS-2:

|                     |                                             |
|---------------------|---------------------------------------------|
| <i>basename.S00</i> | Display screen image file                   |
| <i>basename.DDS</i> | COBOL data description statement file       |
| <i>basename.GEN</i> | Source file for the generated index program |

## Index Program Compilation

Compile the index program in the usual way by entering the command:

```
cob -ik GEN basename.GEN ↵
```

After compilation, you can run the generated program.

## Index Program Running

Run the program immediately after compilation by entering the command:

```
cobrun basename ↵
```

## Data Processing Facilities

Immediately after the program is invoked, your form is displayed. The form remains on the display screen throughout the run, processing being controlled by manipulation of the data in the variable fields.

A display screen reflects the structure of a single record. Initiate the required processing function by entering data and positioning the cursor as described, then pressing ↵. Index program messages are displayed in an unused area of the display screen as necessary.

The basic operator functions and index program messages are described below, and will suffice in general use. Details of the index program interpretation of data manipulation and cursor position follow this description.

**Enquiry by Key Field:** Amend key fields only and press ↵. The required record is displayed. If the record is not found (that is, key cannot be found) the message Record not found is displayed.

**Sequential Enquiry:** Press ↵ to show the next record. If the end of the file is reached, the message End of file reached - return will terminate is displayed.

**Amend Displayed Record:** Amend data fields only and press ↵. The message Record amended is displayed.

**Delete Displayed Record:** Press the HOME key and press ↵. The message Record deleted is displayed.

**Insert New Record:** Amend the key and data fields as required and press ↵. If the currently displayed data fields do not need changing, press HOME before pressing ↵.

---

The message `New record written` is displayed if insertion takes place. If a record already exists with the specified key, the current display is retained, and the warning `Record already exists with this key` is displayed. The facilities available on the subsequent input are as follows:

- Force replacement of existing record by pressing **HOME** and then `↵`.  
The record is replaced and the message `Record replaced` is displayed.
- Amend key field and attempt the insertion again by amending the key fields and press `↵` (cursor position is irrelevant).
- Abandon insertion attempt and display existing record by pressing `↵` only.

**Terminate Run:** Enquire up to the end-of-file by means of continual sequential enquiry or a combination of enquiry by key to a specific record, then sequential enquiry.

When end-of-file is reached, this message is displayed:

End of file reached - return will terminate

Press `↵` to terminate the run.

## Interpretation of User Requirements

The index program interprets your requirements according to the status of key and data fields and the cursor position as follows:

**Key and Data Fields Unchanged:** The function performed depends on cursor position:

- If an end-of-file condition has been reported, a request to terminate the run is assumed regardless of cursor position.
- Otherwise, if you have moved the cursor to **HOME** position and a record is currently displayed, a delete request is assumed.
- If neither of these conditions exists, a request to display the next record relative to the current position in the file is assumed.

**Key Changed and Data Unchanged:** The function performed depends on cursor positions as follows:

- If you have moved the cursor to either the **HOME** position or the last data character position, an attempt to insert a record is assumed, and processing is as described in "Key Unchanged and Data Changed."
- Otherwise, an enquiry with respect to this key is assumed, and either the record is displayed or its absence is reported.

**Key Unchanged and Data Changed:** This is a request to update the file. Either a new record is written or the existing record is amended, as is appropriate.

---

**Key and Data Changed:** This is a request to insert a new record. However, it is assumed that you should not overwrite a record without at least being informed of its presence. Therefore, if a record exists with the specified key, a warning message is displayed. One of the following three functions can be performed depending on the status of key and data fields and the cursor position:

- **Key and Data Unchanged**

The function required depends on cursor position:

- If you have moved the cursor to **HOME** position (or the last data character position), insertion of the new record is forced, and the existing record is overwritten.
- If the cursor is at any other position, a request to abandon the insertion attempt and display the existing record is assumed.

- **Data Unchanged and Key Changed**

An attempt is made to insert the data under the new key regardless of cursor position. If necessary, the warning message will be repeated.

- **Key and Data Changed**

A normal insert request as described above is assumed.

---

## User Index Program Example

The following example shows how to generate an indexed sequential file that contains records of names, addresses, and telephone numbers with **NAME** as key field, and process these records.

```
NAME []
ADDRESS []
 []
 []
TEL [-]
```

**NAME** and **ADDRESS** are alphanumeric fields and **TEL** is a numeric field.

If you use the form for data entry and key in John Smith, 500 Chestnut St., Santa Cruz CA 95060, 425-7222, the form will appear as:

```
NAME [John Smith]
ADDRESS [500 Chestnut St.]
 [Santa Cruz]
 [CA 95060]
TEL [425-7222]
```

To do this, carry out the following steps:

1. Type:

```
forms2 ◀
```

2. **FORMS-2** shows display screen I01 prompting you for a 6-character base for file names and data-names followed by four other questions. If the console is standard, no further questions need be answered for this display screen. Key **demo2** and press ◀, if the default display screen size is correct.
3. **FORMS-2** shows display screen I02 prompting for the output file option type and directory. Press **G**, then ◀, to accept the option for the index program.

4. FORMS-2 shows display screen W01 to request the display screen type option. Note the default option A and press `↵`.
5. FORMS-2 shows a blank display screen with the end of the window one line up from the bottom of the display screen and delimiters in the bottom line. Use the cursor control keys and the normal character keys to set up the following text on the display screen:

```

NAME [-----]
ADDRESS [-----]
 [-----]
 [-----]
TEL [---]

```

Press `↵`.

6. FORMS-2 puts “\_” at the top left of the display screen. Press `SPACE` and then `↵`.
7. FORMS-2 processes the display screen to create a fixed text form. FORMS-2 displays the DDS source code as generated, followed by a redisplay of the fixed text as it is written to the display screen image file.  
  
A message is then displayed giving the name of the fixed text display screen image file created. Press `↵` as requested.
8. FORMS-2 shows display screen W01 prompting for the display screen type option. Note the default option C and press `↵`.
9. FORMS-2 shows the fixed text display screen as background data; use the cursor control keys and fill the NAME variable data field with Xs. Move the cursor to the first character position in the address variable data field and then press `↵` to enter command mode. Type \* to set the first character position in the ADDRESS variable data field as the start of data position and then press `↵`. Continue to enter Xs and 9s to fill the data fields as shown:

```

NAME [XXXXXXXXXXXXXXXXXXXXX]
ADDRESS [XXXXXXXXXXXXXXXXXXXXX]
 [XXXXXXXXXXXXXXXXXXXXX]
 [XXXXXXXXXXXXXXXXXXXXX]
TEL [999-9999]

```

You have now specified the NAME variable data field as the key field. Press `↵`.

10. FORMS-2 displays “\_” at the top left of the display screen; press `SPACE` and then `↵`. A message is displayed showing that validation is in progress.
11. FORMS-2 processes the Xs and 9s to create a variable data form and displays the DDS source code as generated, followed by a redisplay of the variable text as it is written to the display screen image file.
12. FORMS-2 terminates automatically after displaying the end of the run display screen:

```

File created = demo2.DDS
File created = demo2.GEN
END OF FORMS2 RUN

```

13. Compile the index program. Type:

```
cob -ik GEN demo2.GEN ↵
```

- 
14. When the compilation is finished, run the generated index program **demo2** by typing:

cobrun demo2 ◀

15. The generated index program will run. Your display screen, as designed in step 9, is displayed. The fixed text form is shown on the display screen. The variable data fields are used to accept data.

You are now ready to practice all the file maintenance commands. The next steps show all of these in use, but you can vary the sequence or add steps to these once you have gained confidence.

1. To insert the first record into the new indexed sequential file, type the name and address into the display screen format; terminate the record by pressing ◀. Remember to enter surname first before initials to keep the application feasible.
2. Enter two more complete records, overtyping all data from the previous record, because all displayed data is written to the file.
3. When you have inserted three records you can amend the second record as follows:

Enter the name field as for the second record added, followed by ◀. The whole record is displayed because the name is the key that finds the record. You have now seen the enquiry facility in operation. You can recall any record in this manner.

4. Change the town field and press ◀. The message RECORD AMENDED is displayed.
5. Press ◀ and the third record is displayed. You could progress through a whole file in this way.
6. To delete the third record entered, move the cursor to **HOME** position and press ◀. The fields clear showing deletion of that record, and the message RECORD DELETED is displayed.
7. Press ◀. The index program attempts to show the next record, but one does not exist, so an end-of-file message is shown:  
END OF FILE REACHED - RETURN WILL TERMINATE.
8. Press ◀ with the END OF FILE message showing to terminate the program.

This is the end of the record handling method. The following files now exist in your directory:

|                      |                                                     |
|----------------------|-----------------------------------------------------|
| <b>demo2.DDS</b>     | Data description statements for form (COBOL source) |
| <b>demo2.GEN</b>     | Source code of index program demo2                  |
| <b>demo2.int</b>     | Intermediate code of index program                  |
| <b>demo2.DAT.idx</b> | Index file                                          |
| <b>demo2.DAT</b>     | Data file                                           |

The two files **demo2.DAT.idx** and **demo2.DAT** constitute the indexed sequential files created by the generated index program, and in any further runs of this program these two files will be used.





---

---

## **Chapter 13. Ryan-McFarland COBOL: Conversion Series 3**

---

## Contents

|                                                                |       |
|----------------------------------------------------------------|-------|
| About This Chapter                                             | 13-5  |
| Converting RM/COBOL Applications to AIX VS COBOL               | 13-6  |
| Submitting RM/COBOL Source Programs to AIX VS COBOL            | 13-6  |
| Converting Data Files                                          | 13-6  |
| Enhancing Your Converted Application                           | 13-6  |
| Other Considerations for Conversion                            | 13-7  |
| Submitting an RM/COBOL Application to the AIX VS COBOL System  | 13-7  |
| Migrating from the RM/COBOL Environment                        | 13-7  |
| tabx Program                                                   | 13-8  |
| Source Compatibility                                           | 13-9  |
| RM Directive                                                   | 13-9  |
| SPZERO Option                                                  | 13-9  |
| Perform Statements                                             | 13-9  |
| Types of Data                                                  | 13-10 |
| COMPUTATIONAL-1 (COMP-1) Data Types                            | 13-10 |
| COMPUTATIONAL-6 (COMP-6) Data Types                            | 13-10 |
| COMPUTATIONAL (COMP) Data Types                                | 13-10 |
| Conversion Problem Solving                                     | 13-11 |
| Length of Nonnumeric Literals                                  | 13-11 |
| Source Code in Columns 73 to 80                                | 13-12 |
| Reserved Words                                                 | 13-12 |
| Numbering Segments                                             | 13-12 |
| Program Identification and Data-Names                          | 13-13 |
| Column Number Specification                                    | 13-13 |
| End-of-File Notification                                       | 13-13 |
| HIGH-VALUES                                                    | 13-13 |
| Duplicate Paragraph Names                                      | 13-14 |
| Display of Input Data in Concealed ACCEPT Fields               | 13-14 |
| Executable Code Problems                                       | 13-14 |
| Trailing Blanks in Line-Sequential Files                       | 13-15 |
| Undefined Results of MOVE and Arithmetic Operations            | 13-15 |
| Embedded Control Sequences in DISPLAY Statements               | 13-15 |
| Redefinition of COMPUTATIONAL or COMPUTATIONAL-6 Data<br>Items | 13-16 |
| ON SIZE ERROR Clause                                           | 13-17 |
| Field Wrap-Around                                              | 13-17 |
| COMPUTATIONAL-1 Data Items with a Picture Other Than S9(4)     | 13-18 |
| File and Record Locking                                        | 13-19 |
| Initialization of the WORKING-STORAGE                          | 13-19 |
| Converting Data Files for Use with Converted Programs          | 13-20 |
| Supported Data File Types                                      | 13-20 |
| COMP/COMPUTATIONAL Data                                        | 13-20 |
| COMP-3/COMPUTATIONAL-3 Data                                    | 13-21 |
| COMP-6/COMPUTATIONAL-6 Data                                    | 13-22 |
| DISPLAY Data                                                   | 13-22 |
| Program Modifications Required by convert3                     | 13-23 |
| Running convert3                                               | 13-24 |
| Running convert3 in Interactive Mode                           | 13-24 |
| File Details                                                   | 13-25 |
| Print File Name                                                | 13-25 |
| Record Type Specification                                      | 13-26 |
| Binary Sequential Files                                        | 13-28 |
| Generate Program                                               | 13-28 |

---

|                                                        |       |
|--------------------------------------------------------|-------|
| Escape                                                 | 13-29 |
| Running convert3 in Batch Mode                         | 13-29 |
| Running convert3 with a Parameter File                 | 13-33 |
| Using the File Conversion Program                      | 13-33 |
| Creating an Executable File Conversion Program         | 13-34 |
| Running the File Conversion Program                    | 13-34 |
| Indexed Sequential Files with Duplicate Alternate Keys | 13-34 |
| convert3 and File Conversion Program Error Messages    | 13-35 |
| convert3 Error Messages                                | 13-35 |
| File Conversion Program Error Messages                 | 13-36 |



---

## About This Chapter

This chapter details how you can use your IBM AIX VS COBOL to process Version 2 RM/COBOL source programs directly.

This chapter also describes the convert3 file conversion utility, which converts data files created by RM/COBOL programs into data files that can be accessed by the same programs under IBM AIX VS COBOL.

---

## Converting RM/COBOL Applications to AIX VS COBOL

There are two steps in converting an RM/COBOL application:

1. Submit your RM/COBOL source programs to the AIX VS COBOL compiler.
2. Use `convert3` to convert existing data files from RM/COBOL format to AIX VS COBOL format.

### Submitting RM/COBOL Source Programs to AIX VS COBOL

Your AIX VS COBOL compiler includes certain language additions which allow you to submit source programs written in the RM/COBOL language. You must set the `-C rm` option when you use the `cob` command to compile these programs. See “Source Compatibility” on page 13-9 for details.

You will need to alter your source program only if it contains RM/COBOL features which are not supported by the AIX VS COBOL compiler. You may also want to change your source program if it contains features which behave differently under RM/COBOL and AIX VS COBOL, in order to force the AIX VS COBOL system to emulate the behavior of the RM/COBOL system. See “Conversion Problem Solving” on page 13-11 for details.

You must run the `tabx` program before submitting any source programs containing TAB characters to the AIX VS COBOL compiler. See “tabx Program” on page 13-8 for details.

### Converting Data Files

`Convert3`, the RM/COBOL to AIX VS COBOL file conversion utility, converts specified data files from RM/COBOL format to AIX VS COBOL format. This ensures that when you run your RM/COBOL programs in the AIX VS COBOL environment, you can still access existing data files that they produced. You must supply `convert3` with the syntactically correct RM/COBOL source program which produced the data files.

---

## Enhancing Your Converted Application

Once you have successfully submitted your RM/COBOL source programs to the AIX VS COBOL compiler, you may wish to take advantage of some of the advanced features offered by the AIX VS COBOL system. These include:

- Enhanced screen-handling
- ANSI 85 HIGH syntax
- IBM VS COBOL II syntax
- Report writer syntax.

Full details on these features and their associated syntax can be found in the *Language Reference*.

In order to use these features, you must specify certain compiler options when recompiling your programs. For example, you must set the `cob -C rw` option if you use report writer syntax. See Chapter 5, “Compiler Options” for details of system options and the features they enable.

---

## Other Considerations for Conversion

The following features may affect compilation and run-time behavior of your converted RM COBOL code:

- Compiler options (See Chapter 5, “Compiler Options”)
- Run Time Switches (See Chapter 7, “Running an AIX VS COBOL Program”)
- **adisf** (See Chapter 10, “Configuring Your AIX VS COBOL System”).

---

## Submitting an RM/COBOL Application to the AIX VS COBOL System

To successfully transfer your source programs and their associated data files from RM/COBOL to AIX VS COBOL, you must be familiar with:

- The operation of the application which you wish to transfer to AIX VS COBOL. This knowledge is necessary to ensure that the results given by the application are the same in both environments.
- The design and implementation of the application.
- The COBOL language.

## Migrating from the RM/COBOL Environment

Follow these steps to migrate from the RM/COBOL environment to the AIX VS COBOL environment:

1. Transfer all source programs and their associated data files to the IBM AIX system.
2. If the source programs contain any TAB characters, run the **tabx** program, which expands them into a form that is acceptable to the AIX VS COBOL compiler. See “tabx Program” on page 13-8 for details.
3. Submit source programs to the AIX VS COBOL compiler using the **cob** command described in Chapter 4, “The COBOL Interface.”
4. Investigate the cause of any problems you may experience when you submit source programs to the AIX VS COBOL compiler. “Conversion Problem Solving” on page 13-11 describes problems which you may experience and gives hints on how you can recover from them. Resubmit your corrected source programs to the AIX VS COBOL compiler.
5. Test any sections of the source programs which do not use data files. If you receive unexpected results see “Conversion Problem Solving” on page 13-11 for a description of known problems you may experience in executing your source programs, and hints on how to correct these. Alternatively, use the ANIMATOR debugging tool to isolate and correct any problems.
6. Ensure that the ACCEPT/DISPLAY module ADIS is configured correctly. See Chapter 10, “Configuring Your AIX VS COBOL System” for details.
7. Run **convert3** to convert any existing data files used by your source programs. See “Converting Data Files for Use with Converted Programs” on page 13-20 for full details on how to use **convert3**.
8. Finish system testing.
9. Archive your original data files to disk or tape.

---

If disk space on your system is limited, you may not be able to have both the old and new copies of all your data files present on your system at the same time. If this is the case, gradually load and unload the data files onto your system during the conversion process.

## tabx Program

If your RM/COBOL source programs contain any TAB characters, you must run the **tabx** program before you can successfully submit them to the AIX VS COBOL compiler. This is necessary because the RM/COBOL and the AIX VS COBOL compilers handle TAB characters differently. Under the RM/COBOL compiler the first TAB stop is at character position 8, while subsequent TAB stops are at 4-character intervals up to position 72. However, under the AIX VS COBOL compiler, the first TAB stop is at character position 9, while subsequent TAB stops are at 8-character intervals.

The **tabx** program expands any TAB characters in your RM/COBOL source programs to spaces. The resulting source code can be submitted successfully to either the RM/COBOL or the AIX VS COBOL compiler.

To run **tabx** enter the following command line:

```
tabx -options input-filename output-filename ◀
```

where *options* can be any of the following:

**v** Sets verbose mode. Any messages are displayed on your screen.

**l** Parameters are read from the file specified as *input-filename*.

**t** (*tab-spec*)

Informs *tabx* of the positions at which TAB characters are set. For example, `-t(16-8, 64)` assumes tab positions are initially at position 16, and then at 8-character intervals up to column 64. By default this is set to `-t(8-4, 72)`, which is suitable for RM/COBOL source programs.

*input-filename*

The file containing the RM/COBOL source program.

*output-filename*

The file to which the source program output is directed by **tabx**. If you do not specify *output-filename*, **tabx** directs its output to *input-filename*, which would overwrite the original contents of *input-filename*.

The following example expands the TAB characters in the file *myfile.cbl* and outputs a new file, *myfile.new*:

```
tabx -v myfile.cbl myfile.new ◀
```

In the following example **tabx** treats each line of the file *filelist* as a command line:

```
tabx -l filelist ◀
```

This example expands the TAB characters in the file *myfile.cbl* and overwrites *myfile.cbl* with the resulting source program:

```
tabx myfile.cbl
```



---

## Source Compatibility

This section describes the options you may need to set to successfully submit your RM/COBOL programs to the AIX VS COBOL compiler. It also describes the treatment of data types by the AIX VS COBOL compiler.

### RM Directive

Although the standard AIX VS COBOL language, as documented in the *Language Reference*, already supports most of the RM/COBOL syntax, it does not support it all. To enable the additional syntax in the AIX VS COBOL language which allows you to process RM/COBOL source programs on the AIX VS COBOL compiler, you must set the **cob -C rm** option when compiling your program. This ensures that most of your RM/COBOL source programs are accepted by the AIX VS COBOL compiler the first time they are submitted. This additional syntax is documented in the *Language Reference*.

If you normally set the ANSI switch when you submit your RM/COBOL source programs to the RM/COBOL system, use the **cob -C rm = ansi** option when compiling your program.

Note that setting the **rm** option automatically sets additional compiler options, See Chapter 5, “Compiler Options” for full details.

It is recommended that you use the **cob -C nomf** option when you compile your RM/COBOL source programs. This ensures that only those words that are treated as reserved words under ANSI 85 HIGH COBOL are regarded as reserved words under the AIX VS COBOL system.

### SPZERO Option

Under RM/COBOL, comparing uninitialized numeric items to zero would be true. This comparison will be false under AIX VS COBOL if the **spzero** directive is not set. However, using the **spzero** option will seriously affect performance and will cause numeric items containing alphanumeric values not to have their most significant half-byte validated in comparison operations in native code. This means that emulation of RM/COBOL’s handling of alphanumeric values in numeric fields would not be complete if the **spzero** option was set.

If you need to emulate RM/COBOL behavior for comparisons of uninitialized numeric fields with zero in native code, you can do this by setting the **spzero** option explicitly. However, you should be aware of the effect on performance that this will have, and the possible effects on other results if you move alphanumeric values to numeric fields in your program. We recommend, therefore, that you ensure that your numeric fields are correctly initialized to zero before they are used, rather than use the **spzero** option.

### Perform Statements

PERFORM statements are not treated in the same way by the AIX VS COBOL and RM/COBOL systems. The AIX VS COBOL system uses a stack-based perform handling system, while the RM/COBOL system associates a return address with a specific procedure name. As a result, under the RM/COBOL system, all end-points of perform statements are always active until they are used. However, under the AIX VS COBOL system, only the end-point of the last perform statement is active at any one time.

---

You must set the **cob -C perform-type = rm** system directive as well as **rm**, if the AIX VS COBOL system is to emulate the behavior of RM/COBOL PERFORM statements.

## Types of Data

The treatment of data types by AIX VS COBOL is compatible with the action of the file conversion utility convert3. AIX VS COBOL always allocates the same number of bytes to the item in question. However, you must be aware that if these items are redefined and the program logic expects to find a certain binary value in the redefinition, you may not receive the behavior you are expecting at run time. You must also note that at run time, AIX VS COBOL may treat the ON SIZE ERROR clause differently from the RM/COBOL system. See "Conversion Problem Solving" on page 13-11 for further details.

The following sections define how the AIX VS COBOL system treats COMPUTATIONAL-1, COMPUTATIONAL-6, and COMPUTATIONAL types of data.

### COMPUTATIONAL-1 (COMP-1) Data Types

The AIX VS COBOL system allocates a 2-byte signed binary data item capable of holding hexadecimal values in the range -32768 to +32767 for each data item declared as USAGE COMP-1 in the source program, regardless of its picture string. That is, the AIX VS COBOL compiler treats each RM/COBOL USAGE COMP-1 data item as though it had a standard AIX VS COBOL picture string of S9(4) COMP. See the *Language Reference* for full details on the standard AIX VS COBOL language.

### COMPUTATIONAL-6 (COMP-6) Data Types

AIX VS COBOL treats any COMP-6 data items in your RM/COBOL source program as AIX VS COBOL COMP format. If, as a result of this, less data space is allocated to each item than would be under the RM/COBOL system, AIX VS COBOL pads the space with null bytes.

1. Behavior in truncation of data moved to COMP-6 items differs between AIX VS COBOL and RM/COBOL. In AIX VS COBOL, should truncation occur when you move data into a COMP-6 item, the truncation will occur according to the rules for moves to AIX VS COBOL COMP fields.
2. If you specify the CALL BY CONTENT statement using COMP-6 fields, the results may not be as you expect.

### COMPUTATIONAL (COMP) Data Types

AIX VS COBOL treats any COMP data items in your RM/COBOL source program as AIX VS COBOL DISPLAY format. The difference in the internal representation of such data in the two systems is that AIX VS COBOL always sets the most significant four bits of each byte to the value 3, while the RM/COBOL system always sets such bits to the value 0.

---

For example, under the RM/COBOL system:

```
PIC 999 COMP VALUE 123
```

is held in three bytes as hexadecimal value 01 02 03 while under the AIX VS COBOL system:

```
PIC 999 VALUE 123
```

is held in three bytes as hexadecimal value 31 32 33.

---

## Conversion Problem Solving

Although most of the RM/COBOL source programs which you supply to the compiler will be accepted syntactically and run successfully, there are a number of areas in which problems may occur. These may cause the AIX VS COBOL compiler to reject some of the syntax contained in your original RM/COBOL source program, or alternatively may cause your program to behave other than expected at run time.

This section details the known features which may give errors or which may not behave as you expect at run time. Hints are also given on how to correct the cause of such errors, and how to emulate the RM/COBOL type of behavior with AIX VS COBOL.

### Length of Nonnumeric Literals

RM/COBOL allows you to write source programs which contain nonnumeric literals in the Procedure Division which are up to 2047 characters long. However, AIX VS COBOL accepts only nonnumeric literals within the Procedure Division which are no longer than 160 characters long.

**Solution:** Amend your source program by creating a new data item in the WORKING-STORAGE SECTION, and assign the literal to the VALUE clause. If you then use the data item in the Procedure Division in place of the original long literal, the AIX VS COBOL compiler will accept your source program.

**Example:** The RM/COBOL system will accept the following line of code, but the AIX VS COBOL compiler will not:

```
MOVE "ABC...AA" TO SCREEN-BUFFER.
```

where ABC...AA represents a literal with 1500 characters. Change the line of code to:

```
MOVE LONG-LITERAL-1 TO SCREEN-BUFFER.
```

Define a new item in the WORKING-STORAGE SECTION of your source program:

```
01 LONG-LITERAL-1 PIC X(1500) VALUE "ABC...AA".
```

You can now compile your amended RM/COBOL source program with AIX VS COBOL.

---

## Source Code in Columns 73 to 80

The AIX VS COBOL compiler ignores any of the code in your source programs which lies within columns 73 to 80.

**Solution:** The illegal COBOL code has probably resulted from expanding TAB characters in your source program to standard TAB stops. If your source program contains any TAB stops, run the **tabx** program before compiling. This expands TAB characters into a form which is acceptable to AIX VS COBOL. See “tabx Program” on page 13-8 for full details.

To solve this problem, remove enough blanks or split the long line, then recompile.

## Reserved Words

When you set compiler options such as **-C mf** or **ans85**, it enables various features and their associated reserved words within the AIX VS COBOL language. As a result, you may receive errors when you compile source programs with such options set, because data items within them have the same name as words which are enabled as reserved words.

**Solution:** Edit your source programs to rename the offending data item(s), or use the **remove** system option to remove a specified reserved word from the reserved word list. See Chapter 5, “Compiler Options” for details of the **remove** option.

**Example:** Your RM/COBOL source program may contain the following lines of code:

```
....
03 SORT PIC 99.
....
MOVE 1 TO SORT
```

If you compile this you will receive an error, as the AIX VS COBOL compiler supports the SORT verb, while the RM/COBOL compiler does not. However, if you use **cob -C 'remove=sort'** to compile you will not receive this error.

## Numbering Segments

The RM/COBOL system allows you to specify segment numbers greater than 99. However, the AIX VS COBOL compiler allows you to specify segment numbers only in the range 0 to 99 inclusive.

**Solution:** Amend your source programs so that segment numbers greater than 99 become less than or equal to 99, making sure that any new segment numbers you allocate do not duplicate an already existing segment number. You should note that segment numbers between 0 and 49, inclusive, are used for the fixed portion of the object program, while segment numbers 50 to 99, inclusive, are used for independent segments. For details on the use of segment numbers, see the *Language Reference*.

---

## Program Identification and Data-Names

The RM/COBOL system allows the PROGRAM-ID and a data item in that program to have the same name. However, the AIX VS COBOL compiler does not allow the use of the same name for the PROGRAM-ID and a data item within the program, requiring instead that each be unique.

**Solution:** Change either the program name in the PROGRAM-ID, or the name of the data item to something that follows the conventions of AIX VS COBOL as given in the *Language Reference*.

## Column Number Specification

AIX VS COBOL allows you to specify column numbers up to and including 255, but the RM/COBOL system allows you to specify column numbers greater than 255. If you attempt to run an RM/COBOL source program containing a column number greater than 255 under AIX VS COBOL, the compiler issues a severe error message.

**Solution:** Amend your source programs so that column numbers greater than 255 become less than or equal to 255. If you want an item on the screen that has a column number greater than 255 to remain in the same position under AIX VS COBOL as under the RM/COBOL system, you will need to recode your program. See the *Language Reference* for details on how to do this.

## End-of-File Notification

The first time the READ of a sequential file is unsuccessful in either AIX VS COBOL or the RM/COBOL system, the FILE STATUS is set to 10 to indicate that there is no next logical record. When a READ of the same file is again attempted, without it having been previously CLOSED and reOPENed or successfully STARTed, AIX VS COBOL continues to indicate that there is no next logical record. However, when a READ of the same file is attempted again under the RM/COBOL system, the FILE STATUS is set to 96, which results in the RTE error message NO CURRENT RECORD DEFINED FOR SEQUENTIAL READ being displayed.

**Solution:** Although the solution to the different FILE STATUSes returned for the circumstances given above depends on the way in which your source program is coded, it is suggested that you include a test for the characters 10 at the same time as the test for the characters 96 in the FILE STATUS portion of your code. You should note that the inclusion of this test for the characters 10 may provide different results under AIX VS COBOL from those under the RM/COBOL system, if the test is used the first time the READ of a sequential file is unsuccessful.

## HIGH-VALUES

The RM/COBOL system allows you to move HIGH-VALUES to a COMP-1 data item, which results in a value of -1. AIX VS COBOL will not allow you to move HIGH-VALUES to a numeric data item because it is an alphanumeric figurative constant.

**Solution:** To initialize COMP-1 data items with HIGH-VALUES, you must change your program by substituting HIGH-VALUES with -1. Note that if the -F run-time switch is set, the move of HIGH-VALUES into a COMP-1 data item would be allowed in AIX VS COBOL, but the result would not be -1.

---

## Duplicate Paragraph Names

If your program contains duplicate paragraph names, AIX VS COBOL and RM/COBOL will resolve references to the duplicate paragraph names in the same way, provided that the duplicate paragraph names are only referenced from within the sections in which they are declared. If, however, you reference a duplicate paragraph name from a different section to the one it is declared in, RM/COBOL will assume that the reference is to the next declaration of the duplicate paragraph name, whereas AIX VS COBOL will give a compilation error.

**Solution:** To ensure that references to duplicate paragraph names are correctly resolved, you must qualify a reference to a duplicate paragraph name by adding the section name in which it is declared.

**Example:** If your source code contains the following:

```
...
PERFORM para-2.
...
sect-1 SECTION.
para-1.
...
para-2.
...
sect-2 SECTION.
para-2.
...
```

RM/COBOL will resolve the reference to para-2 in the PERFORM statement by using the declaration of para-2 in the sect-1 SECTION. Under AIX VS COBOL, however, you must qualify the reference to the duplicate paragraph name in your source code by using the PERFORM para-2 OF sect-1 statement.

## Display of Input Data in Concealed ACCEPT Fields

If you have specified OFF and ECHO clauses for the same ACCEPT statement in your program, RM/COBOL will conceal any data entered during input for that statement, but on completion of input will display the data. AIX VS COBOL, however, will not display the data for this ACCEPT statement once input has been completed.

**Solution:** If you wish to display the data input for an ACCEPT statement with the OFF and ECHO clauses both specified, you must add a DISPLAY statement after the ACCEPT statement.

---

## Executable Code Problems

Once you have successfully submitted your RM/COBOL source program to the AIX VS COBOL compiler and produced executable code, you may encounter problems when you attempt to run this code under AIX VS COBOL. Alternatively, the code may run, but you may find that its behavior under AIX VS COBOL is not exactly the same as under the RM/COBOL system.

---

## Trailing Blanks in Line-Sequential Files

AIX VS COBOL always removes trailing blanks from line-sequential records before writing the record. The RM/COBOL system removes trailing blanks from such records only if the FD entry contains 01 level records of different sizes. This will not cause you any problems when you run your converted RM/COBOL programs under AIX VS COBOL. However, you may receive errors at run time if any REWRITE operations on line-sequential files change the length of the records.

**Solution:** Change the file organization to sequential, or move an alternative padding character (for example, LOW-VALUES) to the end of the record before it is written. This ensures that full-length records are written.

Also, ensure that the T run-time switch is not set, as this may also change the size of the record. See Chapter 7, "Running an AIX VS COBOL Program" for details of this switch.

## Undefined Results of MOVE and Arithmetic Operations

The results of MOVE statements involving numeric and alphanumeric data items may differ under AIX VS COBOL and RM/COBOL systems. So may the results of arithmetic operations or comparisons on numeric items which contain nonnumeric data.

**Solution:** You can overcome most of these incompatibilities by redefining the data items involved, or by recoding the comparisons.

**Example:** If you submit a source program containing the following data items and procedural statements, the specified test will fail at run time under AIX VS COBOL:

```
01 NUMERIC-FIELD PIC 9(5).

PROCEDURE DIVISION.

 MOVE "ABC" TO NUMERIC-FIELD.
 IF NUMERIC-FIELD = "00ABC"

```

When the **rm** directive is set, AIX VS COBOL partially emulates the behavior of the RM/COBOL system for alphanumeric to numeric MOVES by treating the numeric item as an alphanumeric item which is right justified. However, the above example will still fail because the RM system will treat the literal ABC as numeric, and place 00ABC in the numeric item. To make the statement work in AIX VS COBOL, amend the test in the source program to:

```
 IF NUMERIC-FIELD = " ABC"
```

and resubmit the source program.

## Embedded Control Sequences in DISPLAY Statements

In your RM/COBOL source program, you may have embedded control sequences within data items which you want to be displayed. One of the most commonly used sequences is that for selecting underline:

```
 \E[4m
```

---

AIX VS COBOL ignores such control characters at run time because they are hardware dependent. It will attempt to display them as literals but the results are undefined.

**Solution:** Remove the control sequences from your source program and replace them with the equivalent AIX VS COBOL syntax, which in the above example would be **WITH UNDERLINE**. See the *Language Reference* for a full description of syntax. Resubmit your corrected source program to AIX VS COBOL. If any of the syntax which you have added to your source program needs to have additional options set before the AIX VS COBOL system will accept it, you must set these options. In the above example you would need to set the **mf(3)** option. See Chapter 5, "Compiler Options" for a full description of all the available options, and the *Language Reference* for a list of reserved words enabled by each option.

## **Redefinition of COMPUTATIONAL or COMPUTATIONAL-6 Data Items**

AIX VS COBOL fully supports the size and capacity of RM/COBOL type COMPUTATIONAL and COMPUTATIONAL-6 data items, provided the source program containing such items is submitted to AIX VS COBOL with the **rm** option set. However, the internal representation of such data items in AIX VS COBOL and the RM/COBOL system is not the same. See "Source Compatibility" on page 13-9 for full details. This may cause problems if you want to redefine these data items in order to take advantage of their internal format.

**Solution:** MOVE the data items concerned to other data items which are not defined as COMPUTATIONAL or COMPUTATIONAL-6. This converts the data automatically.

**Example:** The following piece of source program is coded to take advantage of the internal representation of COMPUTATIONAL-6 data items under the RM/COBOL system, and to analyze a date field:

```
01 BIRTHDATE-1 PIC 9(6) COMP-6.
01 BIRTHDATE-2 REDEFINES BIRTHDATE-1.
 03 MONTH-2 PIC 99 COMP-6.
 03 DAY-2 PIC 99 COMP-6.
 03 YEAR-2 PIC 99 COMP-6.

PROCEDURE DIVISION.
START-UP SECTION.
PARA-1.

 MOVE 082462 TO BIRTHDATE-1.

 IF YEAR-2 = 62
 DISPLAY "RECORDS NOT AVAILABLE FOR 1962."
```



---

Amend your source program to use the DISPLAY format instead of redefining COMPUTATIONAL-6 data items before submitting it to AIX VS COBOL:

```
01 BIRTHDATE-1 PIC 9(6) COMP-6.
01 BIRTHDATE-2 REDEFINES BIRTHDATE-1.
 03 MONTH-2 PIC 99 COMP-6.
 03 DAY-2 PIC 99 COMP-6.
 03 YEAR-2 PIC 99 COMP-6.
```

```
01 BIRTHDATE-1A PIC 9(6).
01 BIRTHDATE-2A REDEFINES BIRTHDATE-1A.
 03 MONTH-2A PIC 99.
 03 DAY-2A PIC 99.
 03 YEAR-2A PIC 99.
```

....

```
PROCEDURE DIVISION.
START-UP SECTION.
PARA-1.
```

....

```
MOVE 082462 TO BIRTHDATE-1.
MOVE BIRTHDATE-1 TO BIRTHDATE-1A.
```

....

```
IF YEAR-2A = 62
 DISPLAY "RECORDS NOT AVAILABLE FOR 1962."
```

This overcomes any potential problems.

## ON SIZE ERROR Clause

AIX VS COBOL and the RM/COBOL systems treat the ON SIZE ERROR condition differently. The ON SIZE ERROR condition exists under the RM/COBOL system when the value resulting from an arithmetic operation exceeds the capacity for the associated data item. However, the ON SIZE ERROR condition exists under AIX VS COBOL when the value resulting from an arithmetic operation exceeds the capacity of the specified PICTURE string. You may thus have problems if your source programs contain data items whose capacity is not specified by a PICTURE string, for example COMPUTATIONAL-1 data items.

## Field Wrap-Around

If, when using binary data items (RM/COBOL COMPUTATIONAL-1 format items) an arithmetic operation gives a value which exceeds the capacity of the data item, and there is no ON SIZE ERROR clause, AIX VS COBOL wraps around the value of the item. However, under the same conditions the RM/COBOL system sets the data item to the limit of its capacity.

---

**Example:** Under the RM/COBOL system, the following lines of code result in the value +32767 being stored in the data item CALC-ITEM. However, under AIX VS COBOL the value -32768 is stored in CALC-ITEM:

```
01 CALC-ITEM PIC S9(4) COMP-1.

PROCEDURE DIVISION.

 MOVE 32767 TO CALC-ITEM.
 ADD 1 TO CALC-ITEM.
```

### COMPUTATIONAL-1 Data Items with a Picture Other Than S9(4)

The RM/COBOL system notes the PICTURE string for the COMPUTATIONAL-1 data item when it is used as the source of a MOVE statement to an alphanumeric item. However, AIX VS COBOL always assumes a COMPUTATIONAL-1 data item has a PICTURE string of S9(4).

**Solution:** To produce the result you must **redefine** the target of the MOVE statement.

**Example:** The following piece of source program causes TEST-RECORD to hold 99 under the RM/COBOL system, but 0099 under the AIX VS COBOL system, which treats the data item as though it had a PICTURE definition of PIC S9(4).

```
01 TEST-RECORD PIC X(10).
01 COMP-1-ITEM PIC 99 COMP-1.
PROCEDURE DIVISION.
 ...

 MOVE 99 TO COMP-1-ITEM.
 MOVE COMP-1-ITEM TO TEST-RECORD.
```

To overcome this problem, redefine TEST-RECORD as shown below:

```
01 TEST-RECORD.

 03 TEST-NUMERIC-FIELD PIC 99.
 03 FILLER PIC X(8).

01 COMP-1-ITEM PIC 99 COMP-1.

PROCEDURE DIVISION.
 MOVE 99 TO COMP-1-ITEM.
 MOVE COMP-1-ITEM TO TEST-NUMERIC-FIELD.
```

This avoids moving the COMPUTATIONAL-1 data item directly to an alphanumeric field.

---

## File and Record Locking

Certain versions of the RM/COBOL system contain some software errors in the way locks for files and records are handled. For example:

- Indexed files do not detect or acquire locks if they are opened for output, regardless of whether you specify the WITH LOCK phrase.
- Relative and sequential files cannot be locked exclusively.
- Files which are opened for input can detect record locks, although the RM/COBOL documentation states that they cannot.
- The first record in sequential files opened for input-output is locked whenever any other record in that file is.

These errors are not emulated in AIX VS COBOL.

## Initialization of the WORKING-STORAGE

The RM/COBOL system initializes all WORKING-STORAGE to SPACES, unless you have placed numeric data items between data items with value clauses. This behavior is not emulated by AIX VS COBOL.

**Solution:** If this feature causes any problems, add a VALUE clause with the appropriate value to your source program and resubmit it to AIX VS COBOL. This will resolve any problems which may occur if your program relies on the initial value given to the system.

**Example:** The RM/COBOL system initializes the following group item to SPACES:

```
01 GROUP-ITEM.
 03 ITEM-1 PIC X.
 03 ITEM-2 PIC 99.
 03 ITEM-3 PIC X.
```

However, if ITEM-1 and ITEM-3 have value clauses associated with them, the RM/COBOL system initializes the second byte of ITEM-2 to hexadecimal value 0 if ITEM-2 is defined as USAGE COMP (signed or unsigned) or USAGE DISPLAY (unsigned only).

---

## Converting Data Files for Use with Converted Programs

In order to run `convert3`, you must supply it with the name of an existing RM/COBOL source program (or copy file) that is syntactically correct. This source program must contain the `FD` and `SELECT . . . ASSIGN` entries, together with any associated record definitions concerning the RM/COBOL data file you want to convert. The following sections describe the data definition changes that `convert3` makes to the source program.

### Supported Data File Types

The formats of the data and index portions of files under AIX VS COBOL and the RM/COBOL systems are not the same.

`convert3` can convert `COMPUTATIONAL`, `COMPUTATIONAL-3`, `COMPUTATIONAL-6` and `DISPLAY` data from RM/COBOL format to AIX VS COBOL format. The following sections describe how these types of data are converted.

### COMP/COMPUTATIONAL Data

The RM/COBOL system represents `COMP` (or `COMPUTATIONAL`) data in packed decimal format with one character per byte stored in each least significant four-bits. The most significant half-byte always contains zero. If the `PICTURE` string specifies a signed representation, an additional byte is added to the least significant end of the string: a negative value is represented by the hexadecimal value `0D`, and a positive value is represented by the hexadecimal value `0B`.

Consider the following examples:

| Value | Picture Clause | RM Representation (Hexadecimal) |
|-------|----------------|---------------------------------|
| 1234  | PIC 9(5) COMP  | 00 01 02 03 04                  |
| 1234  | PIC S9(5) COMP | 00 01 02 03 04 0B               |
| -1234 | PIC S9(5) COMP | 00 01 02 03 04 0D               |

The conversion program produced by `convert3` converts `COMP` data fields into AIX VS COBOL `DISPLAY` format, with sign trailing separate. This is compatible with the AIX VS COBOL compiler's treatment of RM/COBOL `COMP` fields in the source program when the `rm` option is set. See "Source Compatibility" on page 13-9 for details. If the data item is signed, the sign byte has the most significant half-byte set to hexadecimal value 2.

After conversion, the previous examples are represented as follows:

| Value | Picture Clause    | AIX VS COBOL Representation (Hexadecimal) |
|-------|-------------------|-------------------------------------------|
| 1234  | PIC 9(5) DISPLAY  | 30 31 32 33 34                            |
| 1234  | PIC S9(5) DISPLAY | 30 31 32 33 34 2B                         |
| -1234 | PIC S9(5) DISPLAY | 30 31 32 33 34 2D                         |

### COMP-3/COMPUTATIONAL-3 Data

The RM/COBOL system represents COMP-3 data in packed decimal format with the least significant half-byte holding the sign.

This sign half-byte contains the following values:

| Field            | RM Sign Half-byte Value (Hexadecimal) |
|------------------|---------------------------------------|
| Unsigned         | F                                     |
| Signed, Positive | B or F                                |
| Signed, Negative | D                                     |

Consider the following examples:

| Value | Picture Clause   | RM Representation (Hexadecimal) |
|-------|------------------|---------------------------------|
| 1234  | PIC 9(5) COMP-3  | 01 23 4F                        |
| 1234  | PIC S9(5) COMP-3 | 01 23 4F                        |
| -1234 | PIC S9(5) COMP-3 | 01 23 4D                        |

The only requirement for conversion is that the sign half-byte has to be changed for signed positive fields to hexadecimal value C.

After conversion, the examples above are represented as follows:

| Value | Picture Clause   | AIX VS COBOL Representation (Hexadecimal) |
|-------|------------------|-------------------------------------------|
| 1234  | PIC 9(5) COMP-3  | 01 23 4F                                  |
| 1234  | PIC S9(5) COMP-3 | 01 23 4C                                  |
| -1234 | PIC S9(5) COMP-3 | 01 23 4D                                  |

---

## COMP-6/COMPUTATIONAL-6 Data

The RM/COBOL system holds COMP-6 data in a similar format to COMP-3 data, except that there is no sign half-byte. If a sign is indicated in the picture clause it is ignored and has no effect. The value held is always positive.

Consider the following examples:

| Value  | Picture Clause   | RM Representation<br>(Hexadecimal) |
|--------|------------------|------------------------------------|
| 1234   | PIC 9(5) COMP-6  | 00 12 34                           |
| 123456 | PIC S9(5) COMP-6 | 12 34 56                           |

In order to maintain the size and capacity of the data items, the AIX VS COBOL system treats COMP-6 data items as AIX VS COBOL COMP fields and pads the field with binary zeros where necessary.

After conversion, the examples above are represented as follows:

| Value  | Picture Clause   | AIX VS COBOL<br>Representation<br>(Hexadecimal) |
|--------|------------------|-------------------------------------------------|
| 1234   | PIC 9(5) COMP-6  | 00 04 D2                                        |
| 123456 | PIC S9(5) COMP-6 | 01 E2 40                                        |

Note that the 9(5) COMP field is extended by one byte containing binary zero in order to maintain the size of the original item.

## DISPLAY Data

You should be aware of the following differences between the representation of numeric DISPLAY format data items, with SIGN INCLUDED, under RM/COBOL and AIX VS COBOL:

- AIX VS COBOL does not encode a sign on the data if the data is positive, whereas RM/COBOL increments the value of the most significant half-byte by one to denote a positive value.
- AIX VS COBOL increments the value of the most significant half-byte by 4 to denote a negative value. RM/COBOL increments the value of the most significant half-byte by 1, and increments the value of the least significant half-byte by 9, to denote negative values.

When there is no sign clause associated with a DISPLAY format data item, AIX VS COBOL treats these data items as though you had specified the SIGN TRAILING IS INCLUDED clause. By default, RM/COBOL treats such data items as though you had specified the SIGN TRAILING IS SEPARATE clause. You can force AIX VS COBOL to emulate the behavior of the RM/COBOL system by setting the ANSI parameter with the **rm** system options. Similarly, **convert3** allows you to specify the type of sign used for DISPLAY format data items. See Chapter 10, "Configuring Your AIX VS COBOL System" for details on how you can do this.

Consider the following examples:

| Value | Picture Clause    | RM Representation<br>(Hexadecimal) |          |
|-------|-------------------|------------------------------------|----------|
|       |                   | Leading                            | Trailing |
| 123   | PIC 9(3) DISPLAY  | 31 32 33                           | 31 32 33 |
| 123   | PIC S9(3) DISPLAY | 41 32 33                           | 31 32 43 |
| -123  | PIC S9(3) DISPLAY | 4A 32 33                           | 31 32 4C |

After conversion, these examples are represented as follows:

| Value | Picture Clause    | AIX VS COBOL<br>Representation (Hexadecimal) |          |
|-------|-------------------|----------------------------------------------|----------|
|       |                   | Leading                                      | Trailing |
| 123   | PIC 9(3) DISPLAY  | 31 32 33                                     | 31 32 33 |
| 123   | PIC S9(3) DISPLAY | 31 32 33                                     | 31 32 33 |
| -123  | PIC S9(3) DISPLAY | 71 32 33                                     | 31 32 73 |

### Program Modifications Required by convert3

The following sections list the areas in which you may need to make modifications to an RM/COBOL source program before you use it as input to convert3.

#### The REDEFINES Clause

convert3 cannot process data files whose definition includes a REDEFINES clause. If a record description in the FILE SECTION contains a REDEFINES clause, you must divide this record description into separate record descriptions.

You must also make sure that each record type in a file with multiple record types is identifiable by either:

- A user-written subroutine
- An item that is common to each record type.

See "Record Type Specification" on page 13-26 for more information about handling multiple record files.

#### The USAGE IS INDEX Clause

convert3 cannot process items with USAGE IS INDEX in a record description. If a record description contains such an item, you can alter the item to have a PICTURE string of S9(4) COMP-1.

#### The USAGE Clause with Group Items

convert3 cannot process group items with a USAGE clause in a record description. To overcome this, add a USAGE clause to each elementary item within the group.

---

## Continuation Columns

convert3 cannot process source program statements with a continuation marker in column 7. This limitation applies only from the beginning of your source program to the end of the file section. To overcome this limitation, alter the layout of your source program so that it does not require continuation markers.

## The DECIMAL POINT IS COMMA Clause

convert3 cannot process the DECIMAL POINT IS COMMA clause in the SPECIAL-NAMES paragraph. To overcome this limitation remove the DECIMAL POINT IS COMMA clause.

## Uniqueness of Names in Record Descriptions

All data-names in record descriptions must be unique. Therefore, you must remove all qualified data-names from record descriptions in the RM/COBOL source programs.

## DEPENDING Names

If a record description contains a DEPENDING phrase, the data-name in the DEPENDING phrase must occur in the same record.

## PICTURE Strings

The maximum length of a PICTURE string in a record description is 20 characters. However, you can overcome this limitation by splitting any PICTURE strings which exceed this limit into two and defining a FILLER item with a PICTURE string which corresponds to the size of the second half of the original string.

---

## Running convert3

The first step in converting your RM/COBOL data files is to produce a file conversion program. This program reads RM/COBOL data files and converts them to AIX VS COBOL data files.

Use the convert3 utility to produce a file conversion program. The input to convert3 is the RM/COBOL source program that created the files you are converting. The output is the file conversion program.

You can use convert3 in either of two modes: *interactive mode* or *batch mode*.

If you run convert3 in interactive mode, the system prompts you for various parameters that control the production of the file conversion program on your display screen.

If you run convert3 in batch mode, you must supply the necessary control parameters in a file.

## Running convert3 in Interactive Mode

The convert3 utility is entirely menu driven when run in the interactive mode. It has an on-line help facility on each menu. This displays a screen of information on the facilities available for each menu.

To invoke convert3 enter the command:

```
convert3 ◀
```



---

Once you have invoked `convert3`, the main menu is displayed. From this menu you can select any of the following functions:

- Help
- File details
- Printfile name
- Record type specification
- Generate program
- Escape.

To select the function of your choice, press the associated function or character key indicated in the menu.

The following sections describe these functions.

## Help

This function is available to you when the main menu is displayed, and when you select either the File Details, Printfile Name, or Record Type Specification functions.

When you select the Help function, a help screen is displayed for either the main menu or the function you have selected.

## File Details

When you select this function, a screen appears which prompts you to identify which data files produced by the RM/COBOL source program are to be converted.

Enter the following information on the display screen:

- The FD name of the file to be converted as it appears in the RM/COBOL source program.
- The name of the RM/COBOL source program which wrote the original data file.
- The name of the file conversion program which `convert3` is to generate. This name cannot be the same as that of the RM/COBOL source program.
- The setting of the `rm` directive. This can be either R (the default) or A (RM(ANSI)).

Press `←` to enter the data on this display screen and return to the main menu. If you specify an invalid parameter, the display screen is displayed again so you may correct the parameter.

Press **Escape** to return to the main menu without saving your entries.

## Print File Name

Selecting this function from the main menu displays a screen which prompts you to enter the name of a file to which `convert3` will write all status or error messages. If you choose not to enter this parameter, all messages are output to the display screen.

---

## Record Type Specification

Selecting this function on the main menu displays a screen which allows you to specify the information needed by convert3 to process data files that contain more than one record type; that is, the FD for the file has more than one 01 level entry. You can uniquely determine the type of each record in such multiple-record type files by entering one of the following parameters:

- The name of a subprogram you have written which determines the type of the records. convert3 will call this subprogram when it generates a file conversion program.
- The name of an item within the data file record whose value determines the type of the records.

You cannot enter both of these parameters.

Press **←** to enter the data on this display screen and return to the main menu. If you specify an invalid parameter, the display screen is displayed again so you may enter a valid parameter.

Press **Escape** to return to the main menu without saving your entries.

## Identifying Record Types by Subroutine

You can write an AIX VS COBOL subroutine to determine each record type in a data file with multiple types of records. The file conversion program calls this subroutine each time it reads a record from the RM/COBOL data file. The program passes the contents of the record to the subroutine, which must use some method to determine the type of record. The subroutine then returns a value to the file conversion program indicating the record type. This value is an index to the 01 level entries in the file's FD entry. For example, if a record corresponds to the first 01 level entry in the FD, the subroutine should return the value 1. If a record corresponds to the third 01 level entry in the FD, the subroutine should return the value 3.

The format of the CALL statement in the file conversion program is:

```
call "name" using record-name, record-number, record-length
```

where:

*name* is the subroutine name that you have supplied on this display screen.

*record-name* is an alphanumeric data item referring to the record that has just been read from the RM/COBOL data file.

*record-number* is a PIC 99 field into which your subroutine will return the number identifying the record type.

*record-length* is a PIC 9(6) COMP item containing the length of the record. This is supplied only if you are converting a binary sequential file.

---

The following is an example of a subroutine that you could write to identify a type of record:

```
linkage section.
01 record-name.
 02 filler pic x(6).
 02 rec-id-field pic 9(6).
01 record-type pic 99.
01 record-length pic 9(6) COMP.

procedure division using record-name, record-type,
 record-length.
main-para.
 if rec-id-field < 10
 move 1 to record-type
 else
 if rec-id-field > 9 and < 80
 move 2 to record-type
 else
 move 3 to record-type.
 exit program.
```

### **Identifying Record Types by Unique Record Item**

The file conversion program may be able to determine a type of record in a file with multiple types of records by examining the value of a particular data item. However, it can only do this if the value uniquely determines the record type.

If so, enter the name of the record item on this screen as it appears in the FD entry in the RM/COBOL source program. Now you must enter a list of level 88 conditions and the record numbers that each of these conditions identify. The record number, as with the value returned by a subroutine, indexes the appropriate 01 level entry in the FD parameter.

For example, you might make the following entries on this display screen:

```
Identifying Data Item OR User Subprogram Name
[REC-TYPE-ITEM]

record value(s)
number
[1] [1 thru 15]
[2] [16]
[3] [17 19 21 24 thru 30]
[4] [18 20]
[5] [22 23]
[] []
[] []
[] []
[] []
[] []
```

You can specify the conditions in any order of record number.

## Binary Sequential Files

In a binary sequential file with multiple record types, the file conversion program can identify a record type by its length as long as no two record types have the same length. However, if they do, you will have to use either the subroutine or unique identifier method to identify the record type.

## Generate Program

When you select this function from the main menu, the file conversion program is generated. You must have previously supplied convert3 with all of the necessary parameters to enable it to generate this program.

If any errors occur during generation, relevant error messages are displayed on the screen. If you have specified the name of a print file, these messages are also written to that file.

When a file conversion program has been successfully generated, the menu is displayed.

---

## Escape

This function returns you to your main system prompt from the convert3 utility. You are asked to confirm that this is what you want to do.

## Running convert3 in Batch Mode

You can run convert3 in batch mode by placing all necessary control parameters in a *parameter file* and then running convert3 so that it reads parameters from this file rather than from the display screen.

You can include parameters for several runs of convert3 in the same parameter file. This means that you can write a single parameter file to convert all of your data files at once.

## The Parameter File

The parameter file is a free format line-sequential file. You can specify one parameter per line. You can leave blank lines in the parameter file to improve readability, since the blank lines are ignored by convert3. You can also insert comment lines in the parameter file by using an asterisk (\*) as the first non-space character in the line.

The first word on each line of the parameter file identifies the type of parameter you are specifying. This can be any of the following (in upper- or lowercase characters):

```
LISTFILE
SOURCEFILE
FD
PROGRAM
SUBROUTINE
SIGN
IDENTIFIER
RUN
```

In addition, a line may start with a record number followed by a valid 88-level condition.

## The LISTFILE Parameter

The LISTFILE parameter specifies the name of the file to which convert3 will write any status value and error messages. The parameter has the following format:

```
listfile file-name
```

If you do not specify a LISTFILE parameter or a file name after LISTFILE, messages are written to **stderr**.

## The SOURCEFILE Parameter

The SOURCEFILE parameter specifies the name of the RM/COBOL source file containing the description of the data file to be converted. The parameter has the following format:

```
sourcefile file-name
```

You must supply a SOURCEFILE parameter.

---

### The FD Parameter

The FD parameter specifies the name in the FD entry of the data file to be converted as it appears in the RM/COBOL source program. The parameter has the following format:

FD *file-name*

You must supply an FD parameter.

### The PROGRAM Parameter

The PROGRAM parameter specifies the name of the file conversion program that convert3 will generate. The parameter has the following format:

PROGRAM *program-name*

The *program-name* must not be the same as the name used in the SOURCEFILE or LISTFILE parameters.

You must supply a PROGRAM parameter.

### The SUBROUTINE Parameter

The SUBROUTINE parameter specifies the name of a user-supplied AIX VS COBOL subroutine that the file conversion program can call to determine a type of record in a file with more than one record type. The parameter has the following format:

SUBROUTINE *program-name*

### The SIGN Parameter

The SIGN parameter specifies how the sign is represented in items with USAGE DISPLAY. The parameter has the following format:

SIGN { INCLUDED }  
      { SEPARATE }

If you do not supply a SIGN parameter, SEPARATE is assumed as the default.

### The IDENTIFIER Parameter

The IDENTIFIER parameter specifies the name of an item in the file record whose value can be used to uniquely determine a type of record in a file with more than one record type. The parameter has the following format:

IDENTIFIER *item-name*

---

## Record Number Parameters

If you have specified an IDENTIFIER parameter, you must specify a number of parameters that indicate which values of the item named in the IDENTIFIER parameter correspond to which record types. Each parameter has the following format:

*record-number condition*

where:

*record-number* indicates a record type (1 means the first 01 level entry in the FD parameter, 2 means the second 01 level entry, and so on). The record identification is carried out in the order in which the record-numbers are specified.

*condition* is an 88-level record-type which, if true, indicates that the associated *record-number* gives the correct record type. The otherwise condition indicates the correct record type for those values of the item named in the IDENTIFIER parameter which you have not previously specified in the record number parameter. These must be entered in quotation marks.

You can specify these parameters in any order of *record-number*, but all record number parameters must immediately follow the IDENTIFIER parameter.

See "Example Parameter File" on page 13-32 for an example of the use of these parameters.

## The RUN Parameter

The mandatory RUN parameter invokes convert3 with those parameters that have already been read. The parameter has the following format:

RUN

It does not matter whether convert3 has successfully generated a file conversion program for one set of parameters. convert3 continues to read the next set until the next RUN parameter is encountered, at which point convert3 again attempts the program generation process. This cycle continues until all parameters in the parameter file have been read.

---

## Example Parameter File

The following is an example of a parameter file that generates file conversion programs for three RM/COBOL data files:

```

* Parameters for first run *

LISTFILE history.lst
SOURCEFILE payroll1.cb1
FD employee-file
PROGRAM progemp.cb1
SIGN separate
IDENTIFIER employee-status
2 "99" "103" "200" "201"
1 "1" thru "2000"
3 "6786" "9999"
2 "0" thru "9999"
4 otherwise
RUN

* Parameters for second run *

SOURCEFILE payroll2.cb1
* listing will go to history.lst
FD branch-file
SIGN included
PROGRAM program.cb1
SUBROUTINE branchek
RUN

* Parameters for third run *

LISTFILE logfile
SOURCEFILE payroll2.cb1
FD history-file
PROGRAM history.abc
SIGN included
RUN
```



---

Note the following about this example:

- In the parameters for the first run, the record number parameters following the IDENTIFIER parameter do not have to be in record number order.
- You can use the word OTHERWISE as the 88-level condition in a record number parameter. If none of the previous record number parameters has successfully determined the record type, the record type associated with the OTHERWISE condition is assumed.
- Record identification using record numbers is carried out in the order in which the parameters are presented in the parameter file. Thus, in the first run:
  - An item with one of the values 99, 103, 200, or 201 is in record type 2.
  - An item with any other value in the range 1 to 2000 is in record type 1.
  - An item with value 6786 or 9999 is in record type 3.
  - An item with any other value in the range 0 to 9999 is in record type 2.
  - An item with any other value is in record type 4.

Notice that you can specify the same record number more than once in the same set of record number parameters.

- The LISTFILE parameter applies to all the runs in the parameter file. The same list file will be used until another LISTFILE parameter is read. Thus, the file *history.lst* is used for the first two runs, and the file *logfile* is used for the third run.

## Running convert3 with a Parameter File

To run convert3 in batch mode, enter the command:

```
convert3 parameter-filename ◀
```

Where *parameter-name* is the name of your parameter file.

Each set of parameters is validated by convert3 before the file conversion program is generated. If any parameters are invalid, that particular file conversion program is not generated, and convert3 passes to the next set of parameters in the parameter file.

You can validate the parameters in a parameter file by entering:

```
convert3 parameter-filename VALIDATE ◀
```

The contents of the parameter file are validated, but no file conversion programs are generated.

---

## Using the File Conversion Program

When you have generated the file conversion program for an RM/COBOL data file, the next steps in converting the data file are to:

1. Compile the file conversion program with the AIX VS COBOL compiler to produce an executable file.
2. Run the file conversion program.

---

## Creating an Executable File Conversion Program

To create an executable file, compile your program with `cob -x file.cbl`.

If the file conversion program requires a subroutine to determine a type of record in a file with different types of records, remember to write and create executable code for the subroutine before you attempt to run the file conversion program.

When compiling the file conversion program generated by `convert3`, any informational messages produced may be ignored.

## Running the File Conversion Program

You can run the file conversion program in either interactive or batch mode.

To run the file conversion program in interactive mode, you supply the name of the file after the run command for AIX VS COBOL. This is the same as running any other program.

This command causes the following prompt to be displayed:

Please enter the input *data-file-name*

Type the name of the RM/COBOL data file to be converted and press `↵`. This file must be the one for which this file conversion program was generated.

Now the following prompt is displayed:

Please enter the output *data-file-name*

Type the name of the AIX VS COBOL data file to which the RM/COBOL data file is to be converted and press `↵`. This name must not be the same name as the RM/COBOL data file.

If any errors occur during conversion, an error message appears on the display screen.

To run the file conversion program in batch mode, use the run command for AIX VS COBOL, but before you press `↵`, type: *program-file input-file output-file*.

The file names used in this command are defined as follows:

*program-file* is the intermediate or native code file for the file conversion program.

*input-file* is the name of the RM/COBOL data file to be converted.

*output-file* is the name to be given to the converted data file.

## Indexed Sequential Files with Duplicate Alternate Keys

If you convert an RM/COBOL indexed sequential data file with duplicate alternate keys, the time order of records with duplicate alternate keys is not preserved in the conversion.

---

## convert3 and File Conversion Program Error Messages

The following sections describe error messages for both convert3 and file conversion programs.

### convert3 Error Messages

convert3 returns a result code to the calling program once it has finished a run. This code is in the form of a 3-byte number in ASCII format. If the first byte of this number is a 0, it denotes that convert3 has completed its run successfully and has reported no errors. Any other number appearing as the first byte in the result code indicates that convert3 has detected an error. If an error is reported, the first byte of the result code indicates the type of that error, as shown in Table 13-1.

| 1st Byte In Result Code | Result Value In Remaining Bytes | Error Type                             |
|-------------------------|---------------------------------|----------------------------------------|
| 1                       | <i>nn</i>                       | Print file error                       |
| 2                       | <i>nn</i>                       | Created file error                     |
| 3                       | <i>nn</i>                       | Source file error                      |
| 4                       | <i>yy</i>                       | File entries (SELECT ... ASSIGN) error |
| 5                       |                                 | Record description (FD) error          |
| 6                       |                                 | Parameter description error            |
| 7                       | <i>nn</i>                       | Dynamic stream error                   |

*nn* is the relevant RTE error number (see Chapter 15, "Error Messages").

*yy* is one of the following:

- 01 Multiple assign clauses found in file control entry.
- 02 Multiple reserve clauses found in file control entry.
- 03 Multiple organization clauses found in file control entry.
- 04 Multiple access clauses found in file control entry.
- 05 Multiple record clauses found in file control entry.
- 07 Multiple file status clauses found in file control entry.
- 08 Keyword expected in file control entry but word found was "*xxx*" (where *xxx* is the actual word found).
- 10 External file name missing in assign statement.
- 11 Data-name defined in assign clause is a reserved word.
- 12 Quote expected at end of external file name.
- 20 Non numeric entry in reserve clause.
- 30 Organization type not found in organization clause.
- 32 Inconsistent file control entry.

- 
- 40 Access mode not specified in access clause.
  - 41 Relative key is a reserved word in record clause.
  - 50 Record name is a reserved word in record clause.
  - 60 The word "record" expected in alternate key clause number.
  - 61 Data-name-3 is a reserved word in alternate record number.
  - 62 The word "duplicated" expected in alternate record number.
  - 71 Data-name-4 is a reserved word in file status clause.
  - 90 Continuation found in file control section.

If you receive an error code which starts with the numbers 5 or 6, then the error code is not followed by a number to identify a specific error. Instead, these errors are general and indicate that an error has occurred within the record description or the parameter description. If you receive one of these errors, you can detect its specific cause by looking through the parameter list in the list file. If you did not specify a list file, this parameter list is sent to **stderr**.

### **File Conversion Program Error Messages**

When you attempt to run the file conversion program, you may encounter one of the following unnumbered error messages, all of which are self-explanatory:

- i/p file name invalid
- o/p file name invalid
- i/p and o/p data files have the same name
- ERROR on opening i/p file
- ERROR on opening o/p file
- ERROR on reading i/p file
- ERROR on writing o/p file
- "record type" record-type "at record-number" is invalid
- Attempt to read beyond end of i/p file.

where *i/p file name* is the input file name and *o/p file name* is the output file name.

If you receive any of these errors, the file conversion program terminates immediately.

---

---

## **Chapter 14. Data General COBOL: Conversion Series 5**

---

## Contents

|                                                              |       |
|--------------------------------------------------------------|-------|
| About This Chapter                                           | 14-3  |
| Converting DG Interactive COBOL Applications to AIX VS COBOL | 14-4  |
| Submitting Source Programs                                   | 14-4  |
| Enhancing Converted Applications                             | 14-4  |
| Source Compatibility                                         | 14-5  |
| The DG Directive                                             | 14-5  |
| Reserved Words                                               | 14-5  |
| DG International Character Set                               | 14-5  |
| DG File Status and Other Exception Values                    | 14-6  |
| Calls                                                        | 14-6  |
| LINKAGE SECTION Access                                       | 14-6  |
| Arithmetic of Group Level Items                              | 14-6  |
| Run-Time Switches                                            | 14-6  |
| Program Identification and Data-Names                        | 14-6  |
| Reformatting a DG Source File                                | 14-6  |
| Using reform5                                                | 14-7  |
| Reformatting Rules                                           | 14-7  |
| Converting Data Files for Use with Converted Programs        | 14-7  |
| Supported Data File Types                                    | 14-8  |
| DG Data Types                                                | 14-10 |
| Source File Restrictions                                     | 14-11 |
| The File Conversion Process                                  | 14-12 |
| Running convert5                                             | 14-12 |
| Running convert5 in Interactive Mode                         | 14-13 |
| Running convert5 in Batch Mode                               | 14-17 |
| Example Parameter List                                       | 14-19 |
| Running convert5 with a Parameter File                       | 14-20 |
| Using the File Conversion Program                            | 14-20 |
| Creating an Executable File Conversion Program               | 14-20 |
| Running the File Conversion Program                          | 14-20 |
| Error Messages                                               | 14-22 |
| Errors Reported by convert5                                  | 14-22 |
| Errors Reported by the Conversion Program                    | 14-23 |

---

## About This Chapter

This chapter provides instructions to migrate from the DG Interactive COBOL environment to an IBM AIX VS COBOL environment. It is intended for DG Interactive COBOL users who want to:

- Retain the use of DG Interactive COBOL on some machine environments while moving to the AIX VS COBOL environment on others. If this is your goal, you need to maintain a common set of source programs suitable for all environments.
- Convert applications written in DG Interactive COBOL to the AIX VS COBOL language and enhance them using the advanced language and development features offered by AIX VS COBOL.

---

## Converting DG Interactive COBOL Applications to AIX VS COBOL

To convert a DG Interactive COBOL application to AIX VS COBOL:

1. Use `reform5` to reformat DG Interactive COBOL source programs written in DG Interactive COBOL CRT format.
2. Submit your DG Interactive COBOL source programs to AIX VS COBOL.
3. Use `convert5` to convert existing data files from DG Interactive COBOL format to AIX VS COBOL format.

### Submitting Source Programs

AIX VS COBOL includes language enhancements allowing you to submit programs written in DG Interactive COBOL directly to AIX VS COBOL. Your DG Interactive COBOL source programs must conform to the standard file format for the AIX operating system. Also, ensure that the expansion of tab characters in literals is not significant to the operation of your program before it is processed by AIX VS COBOL.

If your program is written in DG CRT format, you must reformat the source file before you submit it to AIX VS COBOL. A source reformatting utility (`reform5`) is provided. See “Reformatting a DG Source File” on page 14-6.

You must set the `dg` option when you submit your source code to AIX VS COBOL if your source code contains DG Interactive COBOL features that are not supported in AIX VS COBOL, or if user-defined words are reserved words in the AIX VS COBOL language. See the *Language Reference* for a list of COBOL reserved words, and a definition of all the syntax supported within the AIX VS COBOL language. This chapter also contains a list of the additional AIX VS COBOL features enabled within the AIX VS COBOL language when you set the `dg` compiler option. Chapter 5, “Compiler Options” contains details of how to set this option.

---

## Enhancing Converted Applications

After you have successfully submitted your DG Interactive COBOL source programs to the AIX VS COBOL compiler, you may want to use some of the advanced features of AIX VS COBOL. These features include:

- Enhanced screen-handling
- ANSI 85 syntax
- IBM VS COBOL II syntax
- Report writer syntax.

See the *Language Reference* for details on these features and their associated syntax.

To use these features, you must specify certain compiler options when submitting your source programs to AIX VS COBOL. Set the `ans85` option if you use ANSI 85 in your source programs, or set the `rw` option if you use report writer syntax. See Chapter 5, “Compiler Options” for details of all compiler options and the features they enable.



---

## Source Compatibility

All of the DG Interactive COBOL syntax as defined in revision 02, dated August 1984, of the *Data General Interactive COBOL Programmer's Reference Manual*, is supported by AIX VS COBOL with the exception of the following:

UNDELETE RECORD

Support is not guaranteed for any of the syntax of the DG Interactive COBOL language that is not documented in the stated revision of the above referenced manual.

If your DG Interactive COBOL programs perform any screen-handling operations, you must run **adisf** and select the DG screen-handling configuration option. See Chapter 10, "Configuring Your AIX VS COBOL System" for more information.

## The DG Directive

Your DG Interactive COBOL source programs may contain:

1. DG Interactive COBOL syntax that is not found in the AIX VS COBOL language
2. DG Interactive COBOL syntax that is found in AIX VS COBOL but has a different interpretation at run time.

In either case, you must specify the **dg** directive when you submit such source programs to AIX VS COBOL.

The *Language Reference* contains the DG syntax which is not found in the standard AIX VS COBOL language. When you set the **dg** directive, the DG syntax is enabled within the AIX VS COBOL language.

Features which are syntactically the same as AIX VS COBOL features but have different behavior at run time are described in the following sections of this chapter. Unless you set the **dg** directive at the time the object code is produced, this syntax behaves in the standard AIX VS COBOL manner, as documented in the *Language Reference*. You do not need to amend DG Interactive COBOL source programs which contain these features, but if you try to submit them to AIX VS COBOL without having compiled them with the **dg** directive, they may not behave as you expect.

## Reserved Words

The *Language Reference* lists all the words that are reserved in AIX VS COBOL. If you have included any of these words as user-defined words in your DG Interactive COBOL program, use the **remove** directive to disable the relevant reserved words. See Chapter 5, "Compiler Options" for details on this directive.

## DG International Character Set

The environment-dependent feature of DG Interactive COBOL, which allows you to use an extra 69 characters in addition to the 96 characters (0 x 20 through 0 x 7F) in the ASCII set, is not supported by AIX VS COBOL. The compiler option **nls** may be used to extend the character set for the AIX VS COBOL system. See Appendix B, "National Language Support" for more details.

---

## DG File Status and Other Exception Values

DG Interactive COBOL file status and other exception values are environmentally dependent and are not supported by AIX VS COBOL. To maintain a common source for DG Interactive COBOL and AIX VS COBOL, you should maintain copy books of the file status values returned by DG Interactive COBOL and AIX VS COBOL.

## Calls

DG Interactive COBOL system calls are environment dependent and are not supported by AIX VS COBOL. Under AIX VS COBOL you cannot pass a switch in a CALL statement. Use LINKAGE SECTION items to communicate between programs.

## LINKAGE SECTION Access

DG Interactive COBOL allows a main program to access its LINKAGE SECTION, but this facility is not supported by AIX VS COBOL. If a program is to be accessed as a main program, transfer the LINKAGE SECTION entries into the WORKING-STORAGE section.

## Arithmetic of Group Level Items

DG Interactive COBOL allows arithmetic of group level items, but this is not supported by AIX VS COBOL. Redefine these fields into numeric items.

## Run-Time Switches

DG Interactive COBOL run-time switches A to Z, inclusive, are mapped onto the AIX VS COBOL switches 1 to 26, respectively. See Chapter 7, "Running an AIX VS COBOL Program" for details about run-time switches.

## Program Identification and Data-Names

The DG Interactive COBOL system allows the PROGRAM-ID and a data item in that program to have the same name. However, AIX VS COBOL does not allow the use of the same name for the PROGRAM-ID and a data item in a program, requiring instead that each name be unique. Either change the program name in the PROGRAM-ID and in any programs that call this program, or change the name of the data item.

---

## Reformatting a DG Source File

Source programs written in DG Interactive COBOL CRT format cannot be accepted by AIX VS COBOL unless you change them using a source file formatter. The utility, reform5, supplied with AIX VS COBOL, reformats source programs in CRT format so they are compatible with the requirements of AIX VS COBOL. This source file formatter changes the margins and splits any lines containing more than 72 characters. You can use any similar source file formatter to reformat your programs if you do not wish to use reform5.

You must correct DG Interactive COBOL programs written in CRT format using reform5 or a similar source formatter before supplying these programs to the DG file conversion utility, convert5.

---

## Using reform5

To invoke reform5, type:

```
reform5 ↵
```

Once you have invoked reform5, the following message is displayed:

```
Please enter the DG source file name?
```

Enter the name of the source file to reformat and press ↵. The following message is displayed:

```
Please enter the MF source file name?
```

The name that you type, in reply to this message, is the name of the reformatted source file. This file is now in a format compatible with AIX VS COBOL.

To run reform5 in batch mode, type the following command:

```
reform5 input-file output-file ↵
```

where *input-file* and *output-file* are described as above.

## Reformatting Rules

The reform5 utility amends your source program according to the following rules:

1. If the character in column one is an indicator (\* / -), reform5 appends six space characters to the beginning of the line.
2. If the character in column one is any non-space character not mentioned in rule one, reform5 appends seven space characters to the beginning of the line.
3. If area A (column 8 through 11 inclusive) contains all spaces, reform5 appends seven additional space characters to the beginning of the line. However, if the entire line contains spaces, it outputs a single 72-character line containing spaces.
4. If a line is too long to fit onto a single 72-character line, reform5 breaks it into multiple lines, up to a maximum of three. Where possible, these breaks occur at a space character to preserve the program readability.
5. If a line is expanded beyond 132 characters, reform5 truncates the line to 132 characters. This prevents any pseudo comment areas from being included within the program source.

You can use any source formatter that amends your source program according to the above rules; you are not restricted to using reform5.

---

## Converting Data Files for Use with Converted Programs

The convert5 utility is provided to convert your DG Interactive COBOL data files to AIX VS COBOL data files. The convert5 utility produces a file conversion program which is used to convert the data files.

To run convert5, you must supply it with the name of an existing DG Interactive COBOL source program (or copy file) that is syntactically correct. This source program must contain the FD and SELECT...ASSIGN statements, together with any associated record definitions for the DG Interactive COBOL data file that you want to convert.

---

## Supported Data File Types

convert5 can convert sequential, relative and indexed files from DG Interactive COBOL format to AIX VS COBOL format. It is not necessary to convert line-sequential files, as these are already in a format that is compatible with AIX VS COBOL.

### Sequential Files

convert5 supports sequential files with either fixed- or variable-length records. The default is fixed-length mode. However, you can create a sequential file with variable-length records by including the `RECORDING MODE IS VARIABLE` clause within the `FD` entry.

### Relative Files

Before transferring relative files to AIX VS COBOL, you must reformat them. To do this, add a four-byte field containing each record's relative key to the beginning of every record contained within the original data file. Figure 14-1 on page 14-9 is an example of a program that performs this reformatting for a specified relative file.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. REL2SEQ.
AUTHOR. AIX COBOL
DATE-WRITTEN. 10/22/87.
*
* THIS PROGRAM CONVERTS A DG RELATIVE FILE TO A FORMAT
* SUITABLE FOR SUBSEQUENT CONVERSION TO AIX VS COBOL FORMAT.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. DG-10.
OBJECT-COMPUTER. DG-10.
*
INPUT-OUTPUT SECTION.
FILE-CONTROL.

 SELECT RELATIVE-FILE ASSIGN TO DISK "DGRELATIVE"
 ORGANIZATION IS RELATIVE
 ACCESS IS SEQUENTIAL
 RELATIVE KEY IS RELATIVE-KEY
 FILE STATUS IS FILE-STAT.

 SELECT MF-FILE ASSIGN TO DISK "MFRELATIVE"
 ORGANIZATION IS SEQUENTIAL
 ACCESS IS SEQUENTIAL
 FILE STATUS IS FILE-STAT.
DATA DIVISION.
FILE SECTION.

FD RELATIVE-FILE
 LABEL RECORDS ARE OMITTED.
01 RELATIVE-REC PIC X(20).
FD MF-FILE
 LABEL RECORDS ARE OMITTED.
01 MF-REC.
 03 MF-KEY PIC 9(9) COMP.
 03 MF-DATA PIC X(20).
WORKING-STORAGE SECTION.
01 RELATIVE-KEY PIC 9(4) COMP VALUE 0.
01 FILE-STAT PIC XX VALUE "00".
01 RELATIVE-FLAG PIC 9 VALUE 0.
01 RELATIVE-CNT PIC 9(9) VALUE 0.
01 MF-CNT PIC 9(9) VALUE 0.

```

Figure 14-1 (Part 1 of 2). Example Program to Reformat DG Interactive COBOL Relative Data File

```

PROCEDURE DIVISION.
MAIN-PROCEDURE SECTION.
MAIN-PROC1.
 OPEN INPUT RELATIVE-FILE
 OUTPUT MF-FILE.
 PERFORM READ-WRITE UNTIL RELATIVE-FLAG = 1.
 DISPLAY "RELATIVE RECORDS READ = " RELATIVE-CNT.
 DISPLAY "MF RECORDS WRITTEN = " MF=CNT.
 CLOSE MF-FILE RELATIVE-FILE.
 STOP RUN.
READ-WRITE SECTION.
READ-WRITE1.
 READ RELATIVE-FILE AT END
 MOVE 1 TO RELATIVE-FLAG
 GO TO READ-WRITE-EXIT.
 IF FILE-STAT NOT = "00"
 DISPLAY "INPUT FILE STATUS = " FILE-STAT
 STOP RUN.
 ADD 1 TO RELATIVE-CNT.
 ADD 1 TO MF-CNT.
 MOVE RELATIVE-KEY TO MF-KEY.
 MOVE RELATIVE-REC TO MF-DATA.
 WRITE MF-REC.
 IF FILE-STAT NOT = "00"
 DISPLAY "OUTPUT FILE STATUS = " FILE-STAT
 STOP RUN.
READ-WRITE-EXIT.
EXIT.

```

Figure 14-1 (Part 2 of 2). Example Program to Reformat DG Interactive COBOL Relative Data File

Once you have reformatted your data file following the guidelines in Figure 14-1, reform5 can read it sequentially and convert it to AIX VS COBOL relative format.

## Indexed Files

You must pass any indexed data files which you want to convert from DG Interactive Cobol to AIX VS COBOL through the DGCOBOL utility REORG. This enables convert5 to sequentially read the data portion of the files and convert them to the AIX VS COBOL indexed format.

## DG Data Types

The following types of DG data are supported by AIX VS COBOL:

- |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>COMP</b>    | This is identical to COMP in AIX VS COBOL.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>DISPLAY</b> | This has the same format as AIX VS COBOL DISPLAY, except any processing signs are stored differently. convert5 converts the DG format to the standard IBM AIX ASCII format. To emulate the DG behavior, manually amend your source code before you run convert5. You must change the definition of these fields from signed numeric to alphanumeric. If you do this, you must also set the <b>sign = ebcdic</b> option when you submit your code to AIX VS COBOL. |

convert5 does not support the DG INDEX data format.

---

## Source File Restrictions

Be aware of the following restrictions when using convert5:

- You must supply convert5 with a DG Interactive COBOL source program or copy file that is syntactically correct.
- Make sure that no `USAGE IS INDEX`, `REDEFINES`, or `RENAMES` clauses appear in the record descriptions contained in your source programs. The interpretation of records that contain such clauses is ambiguous, and convert5 is unable to handle them. If your source program does contain record descriptions that use `USAGE IS INDEX`, `REDEFINES`, or `RENAMES`, then before using convert5, split these record descriptions into separate descriptions. Next, before you attempt to convert the relevant files, ensure that these records are identifiable, either by a record type field or through a user subroutine.
- You need to define the record structure of multiple record files if convert5 is to run successfully.
- The convert5 program does not interpret those statements within your source program that begin with a continuation mark in column 7. convert5 returns an error if it detects such a marker in an area of the source program which it needs to analyze.
- The time order of duplicate alternate keys in ISAM files cannot be preserved when you convert such files.
- `FD` and `SELECT...ASSIGN` entries can be contained within a copy file provided the `COPY` statement is of the form:

```
COPY "filename".
```

convert5 does not support any other form of the `COPY` statement.

- The `USAGE` clause is not supported for group level data items.
- The phrase `DECIMAL POINT IS COMMA` is not supported by convert5.
- As all data-names within a record definition must be unique, you cannot supply convert5 with DG Interactive COBOL source code that includes qualifiers within a record.
- convert5 assumes that any depending name occurs in the same record.
- No picture string may be greater than 19 characters in length in a record description of a file being converted.
- Composite picture strings must not contain numeric characters.

If you supply convert5 with DG Interactive COBOL source code that is syntactically incorrect, it outputs an error message only if it detects the syntax error. It will then close any open files and abandon the conversion. However, convert5 may not detect that the supplied code is invalid and may proceed to create a file conversion program. If you then run this program, it may convert data incorrectly. Be sure the code which you supply to convert5 is syntactically correct.

You should supply convert5 with a DG Interactive COBOL source program that writes rather than reads the data file that you want to convert. This ensures that all of the necessary information concerning the data files can be found in the supplied source program.

---

## The File Conversion Process

Once you have supplied `convert5` with the necessary parameters, it reads the specified source program and searches for certain keywords. It then extracts the necessary information from the entries associated with these keywords.

The File-Control paragraph is the first entry for which `convert5` searches. This entry must be located in your main source file, and it must start in margin A. Once `convert5` has found this entry, it looks for the `SELECT...ASSIGN` statement relating to the file or record you want to convert to AIX VS COBOL format. This statement can be in the main source file or in a copy file.

Having read and extracted the relevant information from the File-Control entry and the `SELECT...ASSIGN` statement, `convert5` searches for the following keywords: `FD`, `WORKING-STORAGE`, `LINKAGE`, and `PROCEDURE`. These words, together with their associated entries, can be contained in either the DG Interactive COBOL source file or in a copy file.

`convert5` extracts the relevant information from all of the above statements. This enables it to create a file conversion program. The file conversion program is capable of loading an AIX VS COBOL type file which has the same organization as that quoted in the `SELECT...ASSIGN` statement in the original DG Interactive COBOL source file.

If the description of the DG Interactive COBOL data file does not match its actual format, when you attempt to run the file conversion program it will report an appropriate error message and abort the current file conversion. In this situation, you must provide a source file description that `convert5` is able to process.

---

## Running `convert5`

The first step in converting your DG Interactive COBOL data files is to produce a file conversion program. This program reads DG Interactive COBOL data files and converts them to AIX VS COBOL data files.

Use the `convert5` utility to produce a file conversion program. The input to `convert5` is the DG Interactive COBOL source program that created the files you are converting. The output is the file conversion program.

You can use `convert5` in either *interactive mode* or *batch mode*.

If you run `convert5` in interactive mode, the system prompts you to enter various parameters that control the production of the file conversion program on your display screen.

If you run `convert5` in batch mode, you must supply the necessary control parameters in a file.



---

## Running convert5 in Interactive Mode

The convert5 utility is entirely menu driven when run in the interactive mode. It has an on-line help facility that can be accessed on each menu. This displays a screen of information on the facilities available for each menu.

To invoke convert5, type:

```
convert5 ←↓
```

Once you have invoked convert5, the main menu appears. You can select any of the following functions from this menu:

- Help
- File Details
- Printfile Name
- Record Type Specification
- Generate Program
- Escape.

To select a function, press the associated function or character key indicated on the menu.

The following sections describe these functions.

### Help

This function is available to you when the main menu is displayed, and when you select either the File Details, Printfile Name, or the Record Type Specification functions.

When you select the Help function, a help screen is displayed for either the main menu or the function you have selected.

### File Details

When you select this function, a screen appears which prompts you to identify which data files produced by the DG INTERACTIVE COBOL source program are to be converted.

You must enter all of the following information on the screen:

- The FD name of the file to be converted, as it appears in the DG Interactive COBOL source program.
- The name of a DG Interactive COBOL source program that contains the FD and SELECT...ASSIGN statements, as well as any record definitions, for the file to be converted.
- The name of the file conversion program that convert5 is to generate. This name cannot be the same as that of the DG Interactive COBOL source program.

Press ←↓ to enter the data on this screen and return to the main menu. If you specify an invalid parameter, the screen is displayed again so you may correct the parameter.

If you wish to return to the main menu without saving your entries, press **Esc**.

---

## Printfile Name

When you select this function from the main menu, a prompt is displayed where you are to enter the name of a file to which convert5 will write all status or error messages. If you choose not to enter this parameter, all messages are output to the screen.

## Record Type Specification

When you select this function on the main menu, a screen is displayed that allows you to specify the information needed by convert5 to process data files that contain more than one record type; that is, the FD for the file has more than one 01 level entry. You can uniquely determine the type of each record in such multiple-record files by entering one of the following parameters:

- The name of a subprogram you have written that determines the types of the records. convert5 will call this subprogram when it generates a file conversion program.
- The name of an item within the data file record whose value determines the type of the records.

You cannot enter both of these parameters.

Press **←** to enter the data on this screen and return to the main menu. If you specify an invalid parameter, the screen is displayed again so you may correct the parameter.

Press **Esc** to return to the main menu without saving your entries.

**Identifying Record Types by Subroutine:** You can write an AIX VS COBOL subroutine to determine each record data type in a data file with multiple types of records. The file conversion program calls this subroutine each time it reads a record from the DG Interactive COBOL data file. The conversion program passes the contents of the record to the subroutine which must use some method to determine the type of record. The subroutine then returns a value to the file conversion program indicating the record type. This value is an index to the 01 level entries in the file's FD entry. For example, if a record corresponds to the first 01 level entry in the FD, the subroutine should return the value 1. If a record corresponds to the third 01 level entry in the FD, the subroutine should return the value 3.

The format of the CALL statement in the file conversion program is:

```
CALL "name" USING record-name, record-number, record-length
```

where:

*name* is the subroutine name that you have supplied on this screen.

*record-name* is an alphanumeric data item referring to the record that has just been read from the DG Interactive COBOL data file.

*record-number* is a PIC 99 field into which your subroutine will return the number identifying the record type.

*record-length* is a PIC 9(6) COMP item containing the length of the record. This is supplied only if you are converting a binary sequential file.

---

Below is an example of a subroutine you could write to identify a type of record.

```
linkage section.
01 record-name.
 02 filler pic x(6).
 02 rec-id-field pic 9(6).
01 record-type pic 99.
01 record-length pic 9(6) COMP.

procedure division using record-name, record-type, record-length,
main-para.
 if rec-id-field < 10
 move 1 to record-type
 else
 if rec-id-field < 80
 move 2 to record-type
 else
 move 3 to record-type.
 exit program.
```

**Identifying Record Types by Unique Record Item:** The file conversion program may be able to determine a type of record in a file with multiple types of records by examining the value of a particular data item. It can only do this if the value uniquely determines the record type.

If so, enter the name of the record item on this screen as it appears in the FD entry in the DG Interactive COBOL source program. Now you must enter a list of level 88 conditions and the record numbers that each of these conditions identify. The record number, as with the value returned by a subroutine, indexes the appropriate 01 level entry in the FD parameter.

For example, you might make the following entries on this screen:

| Identifying Data-Item OR User Subprogram Name<br>[REC-TYPE-ITEM] |                      |          |
|------------------------------------------------------------------|----------------------|----------|
| record<br>number                                                 |                      | value(s) |
| [1]                                                              | [1 thru 15           | ]        |
| [2]                                                              | [16                  | ]        |
| [3]                                                              | [17 19 21 24 thru 30 | ]        |
| [4]                                                              | [18 20               | ]        |
| [5]                                                              | [22 23               | ]        |
| [6]                                                              | [OTHERWISE           | ]        |
| [ ]                                                              | [                    | ]        |
| [ ]                                                              | [                    | ]        |
| [ ]                                                              | [                    | ]        |
| [ ]                                                              | [                    | ]        |
| [ ]                                                              | [                    | ]        |

The sequence in which these entries appear is significant. See "Example Parameter List" on page 14-19 for more information.

**Binary Sequential Files:** In a binary sequential file with multiple record types, the file conversion program can identify a record type by its length as long as no two record types have the same length. However, if they do, you will have to use either the subroutine or unique identifier method described in the previous sections to identify the record type.

## Generate Program

When you select this function from the main menu, the file conversion program is generated. You must have previously supplied convert5 with all of the necessary parameters to enable it to generate this program.

If any errors occur during generation, relevant error messages are displayed on the screen. If you have specified the name of a print file, these messages are also written to that file.

When a file conversion program has been successfully generated, the main menu is displayed.

---

## Escape

Escape leaves the convert5 utility and returns you to your main system prompt. You are prompted for a confirmation that this is what you want to do.

## Running convert5 in Batch Mode

If you wish to run convert5 in batch mode, you must first write a parameter file containing the parameters required by convert5. When you run convert5, it validates all of the parameters you have entered. If they are all valid, each invocation of convert5 extracts a full set of parameters from this file. If the parameters are not valid, no conversion takes place.

## Parameter File

This is a free-format line-sequential file that contains a parameter on each line. If you enter any blank lines, these are ignored by convert5. The first word on each line identifies the type of parameter you are specifying. This can be any of the following (in uppercase or lowercase) and should be followed by its value:

```
LISTFILE
SOURCEFILE
FD
PROGRAM
SUBROUTINE
IDENTIFIER
RUN
```

If you enter an asterisk (\*) as the first nonspace character on any line, the line is treated as a comment line.

In addition, a line may start with a record number (see “Record Number Parameters” on page 14-18) followed by a valid 88-level condition.

## LISTFILE Parameter

This parameter specifies the name of a file where any error messages or status values reported from convert5 are recorded. When you specify this parameter, it remains valid until you specify the parameter again, ensuring that all the information reported during a run can be placed in the same listing file. If you specify the parameter again with another file name, subsequent messages are sent to this second file. If you specify the parameter followed only with spaces, subsequent messages are sent to your screen.

The parameter has the following format:

```
LISTFILE file-name
```

If you do not specify either a LISTFILE parameter or a file-name after LISTFILE, messages are written to **stderr**.

## SOURCEFILE Parameter

This mandatory parameter gives the name of the DG Interactive COBOL source program that is scanned by convert5 for information about the files to be converted. The parameter has the following format:

```
SOURCEFILE file-name
```

---

## FD Parameter

This mandatory parameter gives the internal name of the file as it appears in the FD entry in the DG Interactive COBOL source program. The parameter has the following format:

FD *file-name*

## PROGRAM Parameter

The program parameter is mandatory and gives the name of the source program that convert5 creates. The parameter has the following format:

PROGRAM *program-name*

It must not be the same as the source file or the list file name.

## SUBROUTINE Parameter

The subroutine parameter gives the name of a user-supplied COBOL subroutine as it will appear in a CALL statement in the file conversion program. For files that contain multiple records, this subroutine will identify the correct record definition, allowing you to convert files of any logical complexity. The parameter has the following format:

SUBROUTINE *program-name*

## IDENTIFIER Parameter

The IDENTIFIER parameter specifies the name of an item in the file record whose value can be used to uniquely determine a type of record in a file with more than one record type. The parameter has the following format:

IDENTIFIER *file-name*

## Record Number Parameters

If you have specified an IDENTIFIER parameter, you must specify a number of parameters that indicate which values of the item named in the IDENTIFIER parameter correspond to which record types. The parameter has the following format:

*record-number condition*

where:

*record-number* indicates a record type (1 means the first 01 level entry in the FD, 2 means the second 01 level entry, and so on). Note that record identification is carried out in the order in which the record-numbers are specified.

*condition* is an 88-level record-type which, if true, indicates that the associated *record-number* gives the correct record type. The OTHERWISE condition indicates the correct record type for those values of the item named in the IDENTIFIER parameter which you have not previously specified in the record number parameter.

All record number parameters must immediately follow the IDENTIFIER parameter, and must be in the order you want record identification to be carried out.

## RUN Parameter

The mandatory RUN parameter invokes convert5 with those parameters that have already been read. The parameter has the following format:

RUN

## Example Parameter List

The parameter list shown in Figure 14-2 is an example of a parameter file that you could supply to convert5. This file generates file conversion programs for three DG Interactive COBOL data files.

```

* Parameters for first run *

LISTFILE history.lst
SOURCEFILE payroll1.cb1
FD employee-file
PROGRAM progemp
IDENTIFIER employee-status
2 99 103 200 201
1 1 thru 2000
3 6786 9999
2 0 thru 9999
4 OTHERWISE
RUN

* Parameters for second run *

SOURCEFILE payroll2.cb1
* listing will go to history.lst
FD branch-file
PROGRAM progbran.cb1
SUBROUTINE branchek
RUN

* Last run *

LISTFILE logfile
SOURCEFILE payroll2.cb1
FD history-file
PROGRAM history.abc
RUN
```

Figure 14-2. An Example Parameter File

The parameters for the first run are designed to provide for a file comprising four record types. These records are identified by a data item in one of the records. If this item has the value 6786 or 9999, it is a record type 3. If the values lie within the range 1 to 2000 excluding 99, 103, 200, and 201, it is record type 1. Any other values within the range 0 to 9999 are record type 2, while any values lying outside this range are record type 4.

The second file to be converted during the run of this parameter file also contains multiple records. However, record identification in this case is carried out through the user-supplied subroutine named *branchek*.

---

The third file converted on the final run comprises only one record type, and so record identification is not necessary.

The above example illustrates that record identification through level 88 data items is carried out in the order in which the parameters are presented. You should also note that you can identify the same record type more than once. You may find this useful if more than one parameter is required to cover all possible combinations of data value.

## Running convert5 with a Parameter File

To invoke convert5 in batch mode, type the following command:

```
convert5 parameter-filename ◀
```

where *parameter-filename* is the name of your parameter file.

convert5 now runs in batch mode, extracting a full set of parameters from the parameter file with each invocation.

The parameters supplied in the file are always validated completely. If any errors are detected during this validation, no conversion takes place.

You can validate the entries in a parameter file by entering:

```
convert5 parameter-filename validate ◀
```

If you enter this command, the contents of the parameter file are validated, but no conversion takes place, even if all of the parameters are valid.

---

## Using the File Conversion Program

When you have generated the file conversion program for a DG Interactive COBOL data file, the next steps in converting the data file are:

1. Compile the file conversion program with the AIX VS COBOL compiler to produce an executable file.
2. Run the file conversion program.

## Creating an Executable File Conversion Program

To create an executable file, compile your program with `cob -x file.cbl`.

If the file conversion program requires a subroutine to determine a type of record in a file with different types of records, remember to write and create executable code for the subroutine before you attempt to run the file conversion program.

## Running the File Conversion Program

When you run the file conversion program, it converts those data files that conform to the description given in the FD and SELECT...ASSIGN entries in the original DG Interactive COBOL source program.

The command to run your file conversion program is the same command used to run any other AIX VS COBOL intermediate code file. See Chapter 7, "Running an AIX VS COBOL Program" for more information.



---

When you run the program interactively, the following prompt is displayed:

Please enter the *input-data-filename*

where *input-data-filename* is a complete specification (including the file extension) of the specific data file that you want to convert from DG Interactive COBOL format to AIX VS COBOL format. This file must comply with the file type as specified in the FD and SELECT...ASSIGN entries in your original DG Interactive COBOL source program. It must be in the standard file format for AIX. If it is in CRT format, you must supply it to a source file formatter before passing it to the file conversion program as described in "Reformatting a DG Source File" on page 14-6.

Once you have entered the name of the DG Interactive COBOL data file that you want to convert, the following prompt appears on the screen:

Please enter the *output-data-filename*

The name that you enter in response to this message will be the name that the converted data file will take. This file name must not be the same as the input data file name.

The file conversion program now converts the data file from DG Interactive COBOL to AIX VS COBOL format. If any problems are encountered during this process, an error message is displayed on the screen. See "Errors Reported by convert5" on page 14-22 for a list of error messages.

Alternatively, you can run the file conversion program in batch mode. You must then add to the end of the command line the input-data-file name and the output-data-file name.

---

## Error Messages

Error messages are reported by convert5 and by the conversion program.

### Errors Reported by convert5

convert5 returns a result code to the calling program once it has finished a run. Refer to "Running convert5" on page 14-12 for details of the calling program. This code is in the form of a 3-byte number in ASCII format. If the first byte of this number is a 0, it denotes that the creation program has completed its run successfully and has reported no errors. Any other number appearing as the first byte in the result code indicates that the file conversion utility has detected an error.

If an error is reported, the first byte of the result code indicates the type of that error. Table 14-1 identifies the convert5 error messages.

| 1st Byte in Result Code | Result Value in Remaining Bytes | Error Type                           |
|-------------------------|---------------------------------|--------------------------------------|
| 1                       | <i>nn</i>                       | Print file error                     |
| 2                       | <i>nn</i>                       | Created file error                   |
| 3                       | <i>nn</i>                       | Source file error                    |
| 4                       | <i>yy</i>                       | File entries (SELECT...ASSIGN) error |
| 5                       |                                 | Record description (FD) error        |
| 6                       |                                 | Parameter description error          |
| 7                       | <i>nn</i>                       | Dynamic stream error                 |

where *nn* is the relevant RTE error number (see Chapter 15, "Error Messages"), and *yy* is one of the following types of error:

- 01 Multiple assign clauses found in file control entry.
- 02 Multiple reserve clauses found in file control entry.
- 03 Multiple organization clauses found in file control entry.
- 04 Multiple access clauses found in file control entry.
- 05 Multiple record clauses found in file control entry.
- 07 Multiple file status clauses found in file control entry.
- 08 Keyword expected in file control entry but word found was "xxx" (where xxx is the actual word found).
- 10 External file name missing in assign statement.
- 11 Data-name defined in assign clause is a reserved word.
- 12 Quote expected at end of external file name.
- 20 Nonnumeric entry in reserve clause.
- 30 Organization type not found in organization clause.
- 32 Inconsistent file control entry.

- 
- 40 Access mode not specified in access clause.
  - 41 Relative key is a reserved word in a record clause.
  - 50 Record name is a reserved word in a record clause.
  - 60 The word record expected in alternate key clause number.
  - 61 Data-name-3 is a reserved word in alternate record number.
  - 62 The words 'with duplicates' expected in alternate record number.
  - 71 Data-name-4 is a reserved word in file status clause.
  - 90 Continuation found in file control section.

If you receive an error code that starts with the number 5 or 6, it is not followed by a number to identify a specific error. Instead, these errors are general and indicate that an error has occurred within the record description or the parameter description, respectively. If you receive one of these errors, you can detect its specific cause by looking through the parameter list in the print file. If you did not specify a print file, this parameter list is sent to **stderr**.

### **Errors Reported by the Conversion Program**

When you attempt to run the file conversion program, you could receive one of the following unnumbered, self-explanatory messages. In the following messages, *i/p* stands for input and *o/p* stands for output.

- *i/p* file name invalid
- *o/p* file name invalid
- The *i/p* and *o/p* data files have the same name
- ERROR on opening *i/p* file
- ERROR on opening *o/p* file
- ERROR on reading *i/p* file
- ERROR on writing *o/p* file
- "record type" record-type "at record-number" is invalid
- Attempt to read beyond end of *i/p* file.

where *i/p file name* is the input file name and *o/p file name* is the output file name.

If you receive any of these errors, the file conversion program terminates immediately.



---

## **Chapter 15. Error Messages**

---

## Contents

|                                           |       |
|-------------------------------------------|-------|
| About This Chapter                        | 15-3  |
| Introduction                              | 15-4  |
| Compiler Messages                         | 15-4  |
| Severe Compiler Messages                  | 15-7  |
| Compiler Error Messages                   | 15-40 |
| Compiler Warning Messages                 | 15-43 |
| Compiler Information Messages             | 15-46 |
| Compiler Flags                            | 15-47 |
| Errors Encountered During Code Generation | 15-54 |
| Native Code Generator Messages            | 15-54 |
| Run Time Environment Errors               | 15-60 |
| Types of Errors                           | 15-60 |
| Run Time Environment Error Messages       | 15-62 |
| cob Command Errors                        | 15-85 |

---

## About This Chapter

This chapter describes the messages that you may receive while compiling, generating, or running your COBOL programs.

---

## Introduction

There are three types of error messages you might encounter.

- **Compiler messages**

Describes the messages you can receive from the compiler. Most of these messages indicate that your COBOL syntax is incorrect or that there are inconsistencies in your program.

- **Native Code Generator messages**

Describes the messages you may receive when you are using the native code generator.

- **Run Time Environment Error messages**

Describes the errors reported by the Run Time Environment and looks at how you may code your programs both to trap these errors and recover from them when possible.

---

## Compiler Messages

Compiler messages are produced while you are compiling a COBOL program. They indicate that your COBOL syntax is incorrect or that inconsistencies exist within your program.

Compiler messages have the following format:

```
LINE OF COBOL CODE.
nnn-A*** (mmmm)**
** id# compiler message here
```

where:

*nnn* Is the message number

*A* Is a one-letter identifier that shows the severity of the message

*mmmm* Is the page on which the previous message occurred.

*id#* At the beginning of each message is the AIX VS COBOL component identifier. This component number is 1103 for compiler messages and for flagging messages. It is 1203 for RTE messages.

*A* is one of:

**U** Unrecoverable

An unrecoverable fault causes the compilation to stop. A message of this severity is actually produced by the Run Time Environment, not the compiler.

**S** Severe

A severe fault means that compilation of some of the syntax failed, and you will not be able to produce generated code from the intermediate code. Neither will you be able to run code containing severe faults, unless you set the E run-time switch on (by default, this switch is off). You can use the ANIMATOR debugging tool on intermediate code that contains severe faults.



---

**E** Error

When an error occurs in your source code, the compiler attempts to correct the error and continue compilation. The compiler makes assumptions about what was intended. If this varies from your expectations, you should correct the source code that is in error. In any case, you may wish to correct the source code so that you can compile the code with no errors.

You can animate intermediate code that contains errors, produce generated code from it, or run it.

**W** Warning

A warning indicates that there may be an error in the source code, although the statement is syntactically correct.

You can animate intermediate code that contains warnings, produce generated code from it, or run it.

**I** Information

An information message draws your attention to something in the source code of which you need to be aware. This kind of message does not imply that there is a fault, nor are you required to take any action.

You can animate intermediate code that contains information messages, produce generated code from it, or run it.

Error, warning, and information messages may or may not be produced by the compiler, depending on the setting of the **warning** compiler option. By default, all messages are produced.

Many of these severe faults and errors have a cumulative effect. Thus, if a fault is reported at an early stage in your coding, it will probably produce a series of messages as the compiler goes through the rest of your code. Often a simple omission (such as failing to put a period at an expected place) will cause a series of messages to be given, all of which originate from the initial fault. In such cases, it is often true that only one simple correction to your code may be all that is required to recover from a run of severe faults and errors.

In addition to the various types of messages described here, the compiler may also produce *flags*. These are similar in format to the message:

```
LINE OF COBOL CODE.
-- nnn-level--
-- compiler flag message here
```

where:

*nnn* Is the flag number

*level* Shows the level of syntax as described below.

---

When you use the compiler option **flag**, those areas of syntax not supported at the level you choose are highlighted in this way. The **flagstd** type can be certain combinations of the following list of types. See Chapter 5, "Compiler Options" for a full explanation on how to select certain combinations of the following:

- SAA** Full implementation of IBM System Application Architecture COBOL Reference
- MF** Micro Focus extensions to ANSI COBOL Standard X3.23 1974
- OSVS** IBM OS/VS COBOL
- VSC2** IBM VS COBOL II
- ANS74** ANSI COBOL Standard X3.23 1974
- ANS85** ANSI COBOL Standard X3.23 1985

The compiler produces these flags only when the **flag** option is on. These flags are for your information, and indicate those areas of potential incompatibility if you intend to use your program in a different operating environment. Flags do not affect the running of your program, nor do they prevent you from producing generated code or from using the ANIMATOR debugging tool.

When you use the compiler option **flagstd**, those areas not supported at the level you choose are highlighted in this same way. The **flagstd** type can be certain combinations of the following list of types. See Chapter 5, "Compiler Options" for a full explanation on how to select certain combinations of the following:

- m** ANSI 85 defined Minimum COBOL subset
- i** ANSI 85 defined Intermediate COBOL subset
- h** ANSI 85 defined High COBOL subset
- c1** Communications optional module level 1
- c2** Communications optional module level 2
- d1** Debug optional module level 1
- d2** Debug optional module level 2
- s1** Segmentation optional module level 1
- s2** Segmentation optional module level 2
- r** Report Writer optional module
- o** All Obsolete language elements

The **flag** and **flagstd** options provide similar function ability and thus only one may be used at any time.

---

## Severe Compiler Messages

**001-S      Undefined error**

**Cause:** Your program contains an error that the compiler failed to recognize.

**Action:** Follow your local procedures for reporting software problems. Save a copy of your source code to find the cause of the error.

**002-S      Data name too long**

**Cause:** A name that you have given to an item of data within your program does not conform to the rules governing the construction of such names. A data name may include letters, digits, and hyphens of up to 30 characters, providing that at least one character is contained within it and the last character is not a hyphen. Each data name must be unique, or if not unique, must be qualified.

**Action:** Change the data name in error according to the above rules and recompile your program.

**003-S      Illegal format: Literal**

**Cause:** Either you have used the wrong class of literal for the context of the sentence, or the sequence of characters forming a literal within your source code does not conform to the rules governing the construction of such names. A literal can be either nonnumeric or numeric. If numeric, it can be up to 18 digits in length, but it must not contain more than one sign character or more than one decimal point. A nonnumeric literal can include any allowable character in the computer's character setup up to 160 characters in the Procedure Division and 2048 characters in the Data Division. A nonnumeric literal must be enclosed in quotation marks. If you have used a figurative constant as the literal, make sure that it is referenced by an allowable reserved word (such as ZERO) which you have spelled correctly. A figurative constant and a numeric literal must not be enclosed within quotation marks. You may also have used the wrong class of literal for the context of the sentence.

Alternatively, if you have used the figurative constant ALL in your code, you have not coded it in accordance with the rules governing the use of this constant. ALL must be followed by a nonnumeric literal and not by a numeric one.

**Action:** Correct your code to comply with the above rules and recompile your program. See the *Language Reference* for further details on the use of figurative constants.

**004-S      Illegal character**

**Cause:** Your program contains a character that is not part of the COBOL language set.

**Action:** Replace the illegal character with a valid one and recompile your code.

**005-S      User name not unique**

**Cause:** You have given the same user name without qualification to more than one data-item or procedure name within your source code.

**Action:** Rename or qualify the duplicated data-items or procedure names, ensuring their reference uniqueness, and recompile your program.

- 006-S      Too many data or procedure names declared**  
**Cause:** The dictionary space has been exhausted.  
**Action:** Shorten or delete some of the procedure names in your program and recompile your code. If you are using the `EXTERNAL` attribute on some of your data objects, you may be able to eliminate the problem by increasing the value for the `linkcount` option.
- 007-S      Illegal character in column 7 or continuation error**  
**Cause:** You may have incorrectly typed one of the characters allowed in column 7. You can use only an `*`, `/`, `-`, `SPACE`, or `D` characters in this column.  
**Action:** Change the character to a permitted character and recompile your program.
- 008-S      Unknown COPY file specified**  
**Cause:** A file-name specified in conjunction with a `COPY` statement cannot be found.  
**Action:** Change the file-name and recompile your program, or rename the `COPY` file to match your declaration.
- 009-S      ‘.’ missing**  
**Cause:** Your code does not contain a period in a place where one is expected by the rules of COBOL syntax.  
**Action:** Insert a period at the relevant place (usually at the end of a line).
- 011-S      Reserved word missing or incorrectly used**  
**Cause:** You have either used a reserved word in a place where a user-defined word is expected, or you have failed to use a reserved word where one is needed.  
**Action:** Alter the reserved word to a user-defined word or insert a reserved word according to the context of this message.
- 012-S      Operand is not declared**  
**Cause:** You are attempting to use a data name that you have not declared, or which you have misspelled.  
**Action:** Ensure that the relevant data is declared.  
  
This error may not always occur directly below the data-item that is not declared. This is because the compiler continues checking through the source code to find out whether the data-item is qualified. To find the operand that is in error, work backward through the source to the immediately preceding data-item.
- 013-S      User-name required**  
**Cause:** You have not supplied a user-defined name at the specified place in your program.  
**Action:** Insert a name, ensuring that it conforms to the rules of COBOL syntax and that it is the correct type of name.

- 014-S Invalid operand**  
**Cause:** The operand you have specified is in some way incorrect, and cannot be processed by the compiler. For example, you may have specified a negative integer where only positive integers are allowed.  
**Action:** See the *Language Reference* for details of the operands allowed for this syntax.
- 015-S Procedure Division too large**  
**Cause:** Your program's Procedure Division exceeds the maximum size allowed.  
**Action:** Redesign your program as a set of smaller independent programs that call one another.
- 016-S Data space too large**  
**Cause:** Your program's Data Division exceeds the maximum size allowed.  
**Action:** Redesign your program as a set of independent programs with smaller Data Divisions.
- 017-S DIVISION missing**  
**Cause:** You have omitted the word DIVISION from a division header.  
**Action:** Add the word DIVISION to the relevant Identification, Environment, Data, or Procedure Division heading.
- 018-S SECTION missing**  
**Cause:** You have omitted the word SECTION from a section heading.  
**Action:** Add the word SECTION following the name of the section.
- 019-S BASIS mechanism not supported**  
**Cause:** You have attempted to use a feature from the BASIS mechanism, which is not supported by AIX VS COBOL.  
**Action:** Remove any BASIS mechanism syntax from your code.
- 020-S Numeric literal expected**  
**Cause:** You must specify a numeric literal in this context.  
**Action:** See the *Language Reference* for details on the format of valid numeric literals. Ensure that your numeric literal complies with the rules.
- 021-S Too many qualifiers**  
**Cause:** You have used too many qualifiers when referring to a qualified data name, procedure name, or text name. See the *Language Reference* for details on the number of qualifiers you can specify.  
**Action:** Change the offending reference so that it uses no more than the permitted number of qualifiers.

- 022-S SKIP 1/2/3, EJECT, and TITLE must be alone on line**  
**Cause:** Source program lines containing these words must not contain any other words.  
**Action:** Correct your program so that these words are on a line of their own.
- 023-S Nonnumeric literal expected**  
**Cause:** You must specify a nonnumeric literal in this context.  
**Action:** Ensure that the literal you have specified conforms to the rules for nonnumeric literals.
- 024-S Illegal qualifier**  
**Cause:** You have specified a qualified data name, procedure name, or text name incorrectly.  
**Action:** See the *Language Reference* concerning correct syntax for qualification and correct the qualified reference.
- 025-S Qualification not permitted**  
**Cause:** You cannot qualify a data name, procedure name, or text name in this context.  
**Action:** Make the reference an unqualified reference.
- 026-S Literal too long**  
**Cause:** The literal value you have specified is longer than the maximum literal length permitted.  
**Action:** If your literal is numeric, it can be up to 18 digits in length. If it is nonnumeric, it can contain up to 160 characters in the Procedure Division, or 2047 characters in the Data Division. Ensure that the literal value you have specified in your program is no longer than the maximum length permitted.
- 027-S Number too large**  
**Cause:** You have specified a numeric value larger than the compiler can handle.  
**Action:** See the *Language Reference* for the permitted size of numeric values and alter the value accordingly.
- 028-S Data item too long**  
**Cause:** You have declared a data item that is too long for the specified data type.  
**Action:** See the *Language Reference* for the maximum sizes of data items of various types.
- 029-S Not a data name**  
**Cause:** You have specified an operand that is not a data item where a valid data item is expected. For example, you may have specified an FD name or a condition name instead of a data name.  
**Action:** Ensure that the item that is in error is a data name, and that it is declared.

- 030-S      Should be a group**  
**Cause:** You have specified an elementary item as the sending or receiving field in a MOVE CORRESPONDING statement. Both fields must be group items.  
**Action:** Ensure that both the sending and receiving fields are group items.
- 031-S      Should be elementary**  
**Cause:** You have specified the name of a group data item in a context in which an elementary item must be used.  
**Action:** Correct the reference so that it is a reference to an elementary item.
- 032-S      Should be unitary**  
**Cause:** You have specified a subscripted or indexed data item where one is not allowed.  
**Action:** Correct the reference so that it is a reference to a unitary (that is, nonsubscripted and nonindexed) data item.
- 033-S      Should be procedure name**  
**Cause:** A procedure name (that is, a paragraph or section name) is expected in this context. You have probably specified a data-item name.  
**Action:** Check and correct the procedure name.
- 034-S      Operand should be numeric**  
**Cause:** A numeric value is required in this context, and you have specified a nonnumeric value.  
**Action:** Make the value a numeric value.
- 035-S      Integer required**  
**Cause:** An integer value is required in this context, and you have specified a noninteger value.  
**Action:** Make the value an integer value.
- 036-S      Should be alphanumeric**  
**Cause:** An alphanumeric value is required in this context, and you have specified a numeric value.  
**Action:** Make the value an alphanumeric value.
- 037-S      Should have USAGE DISPLAY**  
**Cause:** The data item should have USAGE DISPLAY.  
**Action:** Change the data item's USAGE to DISPLAY.
- 038-S      Paragraph or phrase repeated illegally**  
**Cause:** You have specified a paragraph or phrase more than once, when you may only specify it once.  
**Action:** Delete the repeated paragraph or phrase.

- 039-S Too many COPY ... REPLACING statements**  
**Cause:** You have exceeded the maximum number of COPY ... REPLACING statements (limit is 150).  
**Action:** Delete some of the COPY ... REPLACING statements and recompile your program.
- 040-S Missing or illegal file name**  
**Cause:** The file name you have specified does not conform to COBOL rules for file names, or it has not been declared in the file control paragraph. This may be due to a misspelled valid file name.  
**Action:** Correct the file name (or, if necessary, add a file description entry to the file control paragraph).
- 041-S Fileshare syntax error**  
**Cause:** You have incorrectly specified the syntax for a file input-output operation involving file or record locking.  
**Action:** See the *Language Reference* for the correct file and record-locking syntax.
- 042-S Must be non-zero**  
**Cause:** The numeric value you specify here must not be zero.  
**Action:** Specify a nonzero value.
- 043-S Literal or figurative constant expected**  
**Cause:** You must specify a literal value or a figurative constant here.  
**Action:** Alter the value you have specified to be a literal or a figurative constant.
- 044-S Literal expected**  
**Cause:** You must specify a literal value here.  
**Action:** Alter the value you have specified to be a literal value.
- 045-S Operand has wrong size**  
**Cause:** The operand in this statement is the wrong length (for example, you may have specified a prompt character more than one character long).  
**Action:** See the *Language Reference* for the correct length of the operand and correct your program.
- 046-S Alphabet name required**  
**Cause:** You must specify the name of a user-defined collating sequence here.  
**Action:** Specify the name of an alphabet that you have defined in the SPECIAL-NAMES paragraph.
- 047-S Numeric literal or ZERO expected**  
**Cause:** You must specify a numeric literal or the figurative constant ZERO.  
**Action:** Specify a numeric literal or ZERO.



**048-S Missing or extra right parenthesis**

**Cause:** The numbers of left and right parentheses in an arithmetic expression are not equal.

**Action:** Check the format of the arithmetic expression and ensure that there is a matching right parenthesis for each left parenthesis.

**049-S Illegal use of Index-name or Index Data-item**

**Cause:** You have used an item with USAGE INDEX in a context where it is not allowed. See the *Language Reference* for details of where you can use such items.

**Action:** Change the USAGE of the item, or use an item that does not have USAGE INDEX.

**050-S Illegal use of Pointer Data-item or ADDRESS OF**

**Cause:** You have tried to perform an illegal operation on a data item with USAGE POINTER. Alternatively, you have tried to apply the ADDRESS OF phrase to an item that is not a 01 or 77 level item in the WORKING-STORAGE or LINKAGE SECTIONs.

**Action:** See the *Language Reference* for details of what operations you can perform on pointer items and correct your program accordingly.

**051-S Not a report-name**

**Cause:** You must use a report name in this context. You have probably misspelled a valid report name.

**Action:** Correct the reference.

**052-S Only allowed with SEQUENTIAL files**

**Cause:** You have performed an operation that is only permitted if the file has SEQUENTIAL organization (for example, CLOSE REEL/UNIT).

**Action:** Change the file organization to SEQUENTIAL and recompile the code.

**053-S Invalid directive**

**Cause:** You have specified an invalid compiler option in a \$SET statement within your program.

**Action:** Correct the compiler option and recompile the code. See Chapter 5, "Compiler Options" for the correct form of compiler options.

**054-S Class name required**

**Cause:** You have failed to define the class name in the SPECIAL-NAMES paragraph, or you have misspelled a COBOL class name.

**Action:** Either define the condition name in the SPECIAL-NAMES paragraph, or use one of the COBOL class names as specified in the *Language Reference*.

- 055-S Word COPY may not be continued when in library text**  
**Cause:** You have allowed the word COPY to be split over two lines in your COBOL code.  
**Action:** Ensure that the entire word COPY is put on the second line.
- 056-S COPY is recursive**  
**Cause:** You have attempted to COPY a file that you had already started copying.  
Alternatively, you may have incorrectly spelled the name of either the file you have previously started to COPY, or the file you currently wish to copy.  
**Action:** Correct your program so the file you have already started to COPY is copied completely before you attempt to copy the same file again.  
Correct the spelling error and recompile your code.
- 057-S Not a report group**  
**Cause:** You must use a report group in this context. You have probably misspelled a valid report group name.  
**Action:** Correct the reference.
- 058-S Not a report name or a report group**  
**Cause:** You have specified neither a valid report name nor a valid report group in a GENERATE statement. You have probably misspelled the report name or the report group.  
**Action:** Check your *Language Reference* for details of the syntax allowed for the GENERATE statement. Correct your reference to the report name or the report group.
- 059-S Cannot GENERATE this report name**  
**Cause:** The report name that you have specified does not contain all of the following:
  - A CONTROL clause
  - Only one DETAIL report group
  - At least one body group.**Action:** Check your *Language Reference* for details of the contents you require for a report name when it is used in a GENERATE statement.
- 060-S Not a detail group**  
**Cause:** You have specified a report group that is not of TYPE DETAIL.  
**Action:** Correct your program so that the report group is of TYPE DETAIL.
- 061-S Pseudo-text incorrectly specified**  
**Cause:** A two-character delimiter for pseudo-text is missing from either a REPLACE or COPY REPLACING statement.  
**Action:** Insert the missing pseudo-text delimiter (that is, "=") at the appropriate point in your program.

- 062-S Cannot have COPY REPLACING within REPLACE or vice versa**  
**Cause:** It is not possible to specify text replacement when text replacement is already active.  
**Action:** Modify the syntax so that only one of the statements (REPLACE or COPY REPLACING) will be active at a time, but not both.
- 063-S Cannot be used in nested program**  
**Cause:** An entry-point is not valid in a nested program.  
**Action:** Remove the entry-point from this code, or do not use nested programs.
- 064-S If file is EXTERNAL, then PADDING CHARACTER must be also**  
**Cause:** If a file is defined as EXTERNAL, then the data item specified in the associated PADDING CHARACTER clause must also be defined as EXTERNAL.  
**Action:** Add the EXTERNAL phrase to the declaration of the PADDING CHARACTER clause.
- 065-S Unsigned positive integer required**  
**Cause:** An unsigned integer was expected but not supplied.  
**Action:** Be sure you have coded an integer that has no sign in this context.
- 066-S Data item must have fixed location**  
**Cause:** An item which is part of an OCCURS...DEPENDING ON (ODO) table entry is being used as a subscript to the table, or another ODO table earlier in the group. This is not allowed.  
**Action:** Move the item to a location outside the ODO and use that item in this statement.
- 067-S Please recompile using a larger value for the LINKCOUNT directive**  
**Cause:** The number of linkage section items required by your program exceed the default limit.  
**Action:** Use the `linkcount` directive to increase this limit.
- 070-S Invalid argument**  
**Cause:** You have specified a COBOL system directive with an invalid argument in a \$SET statement within your program. Refer to Chapter 5, "Compiler Options" for the valid arguments for this directive.  
**Action:** Correct the argument for the COBOL system directive and recompile your program.
- 093-S User-name not unique. Assumed qualified by current 01 level record**  
**Cause:** The specified user-name is not unique but has been assumed to be qualified by the current 01 level item.  
**Action:** You should make the user-name unique or explicitly qualify it.

- 105-S PROGRAM-ID missing**  
**Cause:** You have omitted the word PROGRAM-ID from the PROGRAM-ID paragraph.  
**Action:** Add the word PROGRAM-ID to the PROGRAM-ID paragraph.
- 106-S PROGRAM-ID has illegal format**  
**Cause:** You have specified an invalid program name in the PROGRAM-ID paragraph.  
**Action:** See the *Language Reference* for the correct form of program names.
- 109-S Paragraphs or phrases in non-standard order or repeated**  
**Cause:** You have specified paragraphs or phrases in the wrong order, or you have specified them twice.  
**Action:** See the *Language Reference* for the correct order of paragraphs. If you have repeated a paragraph, delete the repetition.
- 113-S SPECIAL-NAMES clause error**  
**Cause:** You have not specified the SPECIAL-NAMES paragraph correctly.  
**Action:** See the *Language Reference* for the correct format of the SPECIAL-NAMES paragraph.
- 115-S OBJECT-COMPUTER clause not recognized**  
**Cause:** You have specified the OBJECT-COMPUTER paragraph incorrectly.  
**Action:** See the *Language Reference* for the correct format of the OBJECT-COMPUTER paragraph.
- 116-S Character specified twice in alphabet**  
**Cause:** You have specified at least one character twice in the ALPHABET clause. For example, you may have included a character within a range and also specified it as a literal.  
**Action:** Edit your source code so that the characters specified in the ALPHABET clause are referenced only once, and then recompile your code.
- 117-S SWITCH clause error or system-name/mnemonic name error**  
**Cause:** There is an error in the coding of the SWITCH clause of the SPECIAL-NAMES paragraph within the Environment Division of your program.  
**Action:** See the *Language Reference* for the details of the correct syntax for this clause and correct your program. Ensure that each condition named within this clause is declared in the Data Division.

- 118-S      COMMA expected**
- Cause:** You have omitted the word COMMA from the DECIMAL-POINT clause.
- Action:** Add the word COMMA.
- 119-S      CRT expected**
- Cause:** You have used a COBOL word or user-defined word where the compiler expected the reserved word CRT.
- Action:** Edit your code so that the reserved word CRT is used.
- 120-S      Illegal currency symbol**
- Cause:** The literal which you have specified in the CURRENCY SIGN IS clause in the SPECIAL-NAMES paragraph of your program is not one of those permitted under the rules of COBOL syntax, or if it is, it is not enclosed within quotation marks. The literal must be a single character, not a digit, and *cannot* be any of the following: A B C D P R S V X Z \* + - , . ; ( ) " / = or SPACE.
- Action:** Change the literal to a permitted character or digit, ensure that it is enclosed by quotation marks, and recompile your program.
- 121-S      Cannot specify DYNAMIC or EXTERNAL with literal file-name**
- Cause:** You can only specify DYNAMIC or EXTERNAL in the ASSIGN clause of a file description entry if the file name is contained in a data item rather than given as a literal file name.
- Action:** Redesign your program so that the file name is contained in a data item.
- 122-S      Cannot use Double-Byte characters in alphabet definition**
- Cause:** You have used DBCS characters in an alphabet definition.
- Action:** When you define an ALPHABET, you must not use Double-Byte characters.
- 126-S      ASSIGN missing**
- Cause:** You have used a SELECT clause to give a file a file name by which the program will recognize it, but you have failed to use a corresponding ASSIGN clause to give the file an implementer name (the name by which the system will recognize the file).
- Action:** Insert the relevant ASSIGN clause after the SELECT clause.
- 127-S      [LINE] SEQUENTIAL, RELATIVE or INDEXED missing**
- Cause:** In the ORGANIZATION IS clause of the FILE-CONTROL paragraph, you have failed to specify the logical structure your file is to take.
- Action:** Insert [LINE] SEQUENTIAL, INDEXED, or RELATIVE into this clause, depending on how you wish records to be stored and accessed within your file.

- 128-S ACCESS missing on indexed/relative file**
- Cause:** The form of this clause depends on the organization of the data file. If you have specified the ORGANIZATION clause for a file as either INDEXED or RELATIVE, you must specify an ACCESS MODE clause for it indicating which mode you will use to access that file. If no mode is specified, sequential access mode is assumed.
- Action:** See the *Language Reference* for details on the use of this clause.
- 129-S Too many keys or key components**
- Cause:** If you have defined the logical structure of your file to be INDEXED, you have then defined more than 64 keys (1 primary plus 63 alternate), which is the maximum permitted.
- Action:** Delete some of the alternate keys within your program so that the number remaining is less than 64 and recompile your program.
- 130-S Illegal ORGANIZATION/ACCESS/KEY combination**
- Cause:** If you have specified the ORGANIZATION/ACCESS/KEY clauses in your program, they must be compatible, but in your program they are incompatible. See the *Language Reference* for details on permitted combinations.
- Action:** Ensure that these clauses are compatible.
- 131-S Unrecognized phrase in SELECT clause**
- Cause:** The compiler has failed to accept part of your SELECT clause. This message could be given, for example, if the file name which you have given in the SELECT clause does not conform to the rules for naming COBOL files. See the *Language Reference* for details of these and a description of correct coding for the SELECT clause.
- Action:** Correct your code.
- 133-S SAME AREA clause syntax error**
- Cause:** The optional clause, SAME AREA, by which you allow two or more files to access the same central storage space, does not conform to the relevant syntax rules.
- Action:** See the *Language Reference* for the description of the correct format for this clause and correct your code.
- 136-S Illegal use of phrase for National Language operation**
- Cause:** You have included one or more of the following COBOL clauses in your program:
- ```
PROGRAM COLLATING SEQUENCE IS
ALPHABET IS
CURRENCY SIGN IS
DECIMAL-POINT IS COMMA
COLLATING SEQUENCE IS
```
- and you have compiled the program with the **National Language Support (nls)** option on, which is not allowed.
- Action:** You should either recompile your program with the **nls** compiler option turned off, or edit your source code to ensure none of the above clauses appear in the program.

137-S Program collating sequence not defined

Cause: You have included the PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph of your code but have failed to code a corresponding ALPHABET-NAME clause in the SPECIAL-NAMES paragraph. Your code must contain an ALPHABET-NAME clause if it has a PROGRAM COLLATING SEQUENCE clause within it.

Action: Insert the necessary ALPHABET-NAME clause, ensuring it has the same user-defined name as that used within the COLLATING SEQUENCE clause. If you have specified both of these clauses within your code and have still received this message, then ensure that you have used the same user-defined name and that it is spelled the same in each clause. Correct any spelling error your code may contain.

138-S "EXCLUSIVE", "AUTOMATIC", OR "MANUAL" missing

Cause: The LOCK MODE IS clause within the FILE-CONTROL paragraph of the Environment Division does not contain one of the words EXCLUSIVE, AUTOMATIC, or MANUAL, or if it does, the compiler has failed to recognize the word, possibly due to a spelling mistake. The LOCK MODE clause controls access to files shared between a number of users within a multiple-user environment. Its form is determined by the file type with which it is used.

Action: See the *Language Reference* for further details on its use.

139-S Illegal LOCK MODE/file type combination

Cause: The form of the LOCK MODE clause depends on the file type. You have specified a LOCK MODE clause that is incompatible with the type of the specified file. Nonshareable files must have lock mode EXCLUSIVE while shareable files may have lock mode AUTOMATIC or MANUAL.

Action: See the *Language Reference* for further details on the use of this clause.

140-S For indexed file, PASSWORD phrase must follow KEY

Cause: For an indexed sequential file description entry, the PASSWORD phrase, if any, must immediately follow the KEY phrase.

Action: Correct your program so the PASSWORD phrase immediately follows the KEY phrase.

141-S File name appears in more than one SAME clause of same type

Cause: A file may only appear in one SAME AREA or SAME RECORD AREA clause.

Action: Correct your source code and recompile your program.

142-S Can only be used in nested program

Cause: The COMMON clause, for example, as defined by ANSI, is only in a nested program.

Action: Correct your source code and recompile your program.

- 143-S Unknown IDENTIFICATION DIVISION paragraph**
Cause: The paragraph label specified is invalid (or invalid for the selected COBOL dialect).
Action: Correct your program (or select another COBOL dialect) and recompile your program.
- 199-S LINE clause also specified in containing group**
Cause: The LINE clause cannot appear in an elementary data item when the containing group contains a LINE clause.
Action: Revise the declaration of the item to specify the LINE clause either in the group item or in the elementary data item, but not both, and recompile.
- 201-S Sort file cannot have ACCESS or ORGANIZATION clauses**
Cause: A screen description (SD) file description entry cannot have these clauses, because the access mode and organization of such files are fixed.
Action: Delete the offending clauses.
- 202-S Too many levels of OCCURS**
Cause: You have specified more than the permitted number of OCCURS clauses in the definition of a table item. See the *Language Reference* for details of how many levels of OCCURS you can specify.
Action: Delete the excess OCCURS clauses.
- 203-S CODE must be specified for all reports or no report**
Cause: You have specified the CODE clause in the definition of a report. If you are defining more than one report for the specified file, you must specify the CODE clause either for all reports or for none of them for that file.
Action: Add CODE clauses to the other reports you have defined, or ensure that no report definition contains a CODE clause for that file.
- 204-S REDEFINES on incorrect field**
Cause: A REDEFINES clause must have the same level number as the item it redefines.
Action: Change the level number.
- 205-S RECORD missing or has zero size**
Cause: This message is given if you declare a file using the SELECT statement within the Environment Division of your program, and then either fail to define it in a corresponding file description (FD) entry in the Data Division or use a different file name in the FD entry, perhaps the result of a spelling mistake. If a mismatch of the two occurs, then the record associated with the file does not exist, although the necessary space has been created for it.
Action: Add or correct the FD entry.

206-S 01 or 77 level required

Cause: This data item must have level number 01 or 77.

Action: Change the level number of the item to 01 or 77.

207-S FD, CD, or SD qualification syntax error

Cause: These entries describe the structure of a specified file and take the form FD, CD, or SD, followed by a file name and a description of the records within that file. The file name must be the name of a file defined in the FILE-CONTROL paragraph of your program. An FD, CD, or SD entry in your source code has an error in its syntax.

Action: Refer to your *Language Reference* for syntax details.

208-S WORKING-STORAGE missing

Cause: The words WORKING-STORAGE are missing from the relevant section heading within the Data Division, or if present, have not been recognized by the compiler, perhaps as a result of a spelling mistake or their position within your code.

Action: See the *Language Reference* for details of how the Data Division should be structured.

209-S Procedure Division missing or unknown statement

Cause: The Procedure Division header is missing from your program, or if present, the compiler has failed to recognize it. This could be due to a spelling error or incorrect placement of this header.

Alternatively, your code could contain a statement within the Procedure Division that the compiler has failed to recognize. This is probably because you have used a reserved word, but have spelled it incorrectly.

Action: Insert the Procedure Division header immediately after the last entry in the Data Division. Alternatively, if you have misspelled a reserved word, correct your spelling error.

210-S Unrecognized data description qualifier or '.' missing

Cause: A qualifier (for example, JUST or COMP) is missing or misspelled in the description of a data item.

This message is also given if you have not placed the necessary period at the end of a PICTURE clause.

Action: Add the necessary qualifier to your code. Alternatively, if you have omitted the necessary period, add the period.

211-S PICTURE clause not compatible with qualifiers

Cause: Your code contains a PICTURE clause that is qualified by the wrong type of qualifier. For example, a data item may be defined as PIC XX USAGE COMP. You will get this message if an alphanumeric PICTURE clause is specified with a numeric qualifier.

Action: Correct your code so that the PICTURE clause and its qualifiers are of the same type.

212-S Illegal data item used with BLANK clause

Cause: The BLANK WHEN ZERO clause can be used only as part of the data description entry for data items that are numeric (those that contain a 9 in the PICTURE clause) or numeric-edited (ordinary edited data items with special characters added to allow the data items to be presented in a form which can be easily understood). You have used this clause with a nonnumeric data item in your source code or with a numeric data item that contains a 'P', 'S', or 'V'.

Action: Delete this clause or redefine the data item to be either numeric or numeric-edited, or remove the 'P', 'S', or 'V' characters.

213-S Item is longer than this USAGE allows

Cause: A PICTURE clause within your Data Division exceeds the maximum permitted by the compiler. Numeric data items can be up to 18 characters in length, numeric-edited up to 512 characters, and alpha-numeric up to eight megabytes.

Action: Alter the clause in error to lie within the relevant range.

214-S VALUE too long for data item

Cause: A data item that you have used with a VALUE clause is bigger than its declared field. A declaration such as ...PIC 99 Value 123 would cause this message, because 123 is too large to fit into the receiving field without truncation.

Action: Alter your code so the data item is large enough to receive the value that is to be placed within it.

215-S VALUE in error or illegal for PICTURE type

Cause: You have made a mistake in the coding of a VALUE clause within your program. You may, for example, have used a numeric data item with a nonnumeric VALUE clause, or you may have used the VALUE clause in conjunction with a nonelementary data item. Alternatively, you may have used it in conjunction with a data item that has been redefined.

Action: Correct the code as necessary so that it does not violate the rules governing the use of the VALUE clause. See the *Language Reference* for a full description of these rules.

216-S Non-elementary item has JUSTIFIED or BLANK clause

Cause: JUSTIFIED or BLANK clauses can be used within the data description entry of the Data Division but must be used only with elementary data items. Elementary data items are the most basic subdivisions of a record, as they are not subdivided into higher level data items. You have used one of these clauses in conjunction with a nonelementary data item, that is, one that is redefined by higher level data items.

Action: Correct your program to ensure that these clauses appear only with elementary data items.

217-S Preceding item at this level has zero length

Cause: You have defined a data item but have failed to give it a size, either by using a PICTURE clause in conjunction with it, or if it is a group item, by failing to define any elementary items to go with it. For example, if the code is something similar to:

```
01 b.
01 m Pic X.
```

you will receive this message after m, because although it is b that is in error, the error is only detected once the next item at the same level as the offending data item is encountered.

Action: Add the necessary PICTURE clause or elementary items to the level hierarchy.

218-S Illegal arithmetic operator

Cause: You have specified an invalid operator in an arithmetic expression.

Action: See the *Language Reference* for details of valid arithmetic operators.

219-S Illegal level number

Cause: You have specified an illegal level number in a data description entry.

Action: See the *Language Reference* for details on how to specify level numbers.

220-S Literal type does not match data type

Cause: You have specified a literal value that is incompatible with the PICTURE clause of the associated data item, for example, PIC 999 VALUE "123".

Action: Make the literal value compatible with the data item.

221-S Data description qualifier inappropriate or repeated

Cause: You have used a data description qualifier that is incompatible with the associated PICTURE clause. For example, you may have used a BLANK WHEN ZERO clause for a PIC XX data item. This is illegal, because the data item is alphanumeric and the associated qualifier refers only to numeric items. Alternatively, you have used more than one PICTURE clause to define a single data item.

Action: Correct the qualifier or the data type to ensure that they are compatible. Alternatively, delete the superfluous PICTURE clauses to leave only one for that data item, since each data item should only have one PICTURE clause associated with it.

222-S REDEFINES data name not declared

Cause: You have not declared the data name that you have used in conjunction with a REDEFINES clause.

Action: Ensure that the data name is declared in your data description entry and that the relevant REDEFINES clause is placed immediately after the data name that it describes.

- 223-S Unknown USAGE**
Cause: You have specified an invalid USAGE qualifier in a data description entry.
Action: See the *Language Reference* for details of USAGE qualifiers.
- 224-S SIGN must be LEADING or TRAILING**
Cause: This optional clause can be specified only for numeric description entries that contain the character S in the PICTURE clause. The key word SIGN must be followed by either LEADING or TRAILING, depending on the position you wish the operational sign to take.
Action: Add either LEADING or TRAILING.
- 225-S Level hierarchy wrong**
Cause: The structure of level numbers in a group data item is incorrect.
Action: Check the hierarchy of the level numbers and correct it.
- 226-S Variable-length group not unitary**
Cause: You cannot declare a variable-length group item within an OCCURS clause.
Action: Give the group item a fixed length.
- 227-S ZERO missing**
Cause: The BLANK clause must be followed by the word ZERO.
Action: Delete any other word you may have used in place of ZERO. Ensure that the clause contains no spelling errors.
- 228-S Group VALUE truncated**
Cause: The value you have specified for a group item is longer than the defined length of the group item.
Action: Redefine the length of the group item or the value.
- 229-S Incompatible qualifiers**
Cause: You have specified qualifiers in the description of a data item that are not compatible with one another.
Action: See the *Language Reference* for the rules governing qualifiers in data descriptions. Correct the data description.
- 230-S PICTURE string has illegal precedence or illegal character**
Cause: You have used a character within a PICTURE clause that the compiler does not recognize or which is illegal for that particular type of PICTURE string.
Action: See the *Language Reference* for a full list of permitted characters. Alter the relevant PICTURE clause.

- 231-S INDEXED data-name missing or already declared**
- Cause:** The INDEXED phrase found within an OCCURS clause is followed by a data-name that you have used elsewhere in the program. This violates the rules of COBOL syntax, which state that each data-name must be uniquely identified. Alternatively, the clause is not followed by any data-name.
- Action:** Either alter or add a data-name, depending on the context in which this message was given.
- 232-S Numeric-edited PICTURE string is too large**
- Cause:** A PICTURE string that you have defined for a numeric-edited data-item (one that presents numeric data-items in a more readable form; for example, with leading zeros removed or with currency signs inserted) exceeds the maximum permitted for your compiler.
- Action:** Alter the string to be less than 512 characters in length.
- 233-S Unknown data description qualifier**
- Cause:** You used an invalid qualifier in a data description. You probably misspelled a valid qualifier.
- Action:** Correct the data description.
- 234-S DEPENDING missing**
- Cause:** You have defined a variable-length table without specifying the DEPENDING phrase that at run time allows the compiler to determine the actual table size.
- Action:** Specify the DEPENDING phrase.
- 237-S Cannot have more than one initial CD**
- Cause:** Only one of the communication description (CD) entries in the Communications section can have the INITIAL clause specified.
- Action:** Delete the INITIAL clauses from all but one of the CD entries.
- 238-S RENAMES missing**
- Cause:** You have omitted the word RENAMES in the definition of a level 66 data item.
- Action:** Add the word RENAMES.
- 239-S First data-name does not precede second**
- Cause:** You have included the syntax data-name RENAMES data-name-2 THRU data-name-3 in your program, but the data item you have specified for data-name-2 is declared after the data item for data-name-3. This is not valid COBOL syntax, and the first data item in a THRU clause must be declared before the second.
- Action:** Correct the THRU phrase.
- 240-S Only allowed at 01 level**
- Cause:** You have used the NEXT or TYPE clauses in a report description entry, or the GLOBAL or EXTERNAL clauses in a report file description entry that is not a 01 level item.
- Action:** Edit your source code to ensure that everywhere you use NEXT, TYPE, GLOBAL, or EXTERNAL, the clause applies to a 01 level item.

- 241-S Only allowed in WORKING-STORAGE SECTION**
- Cause:** You have used the EXTERNAL clause in a record description entry in a section other than the WORKING-STORAGE SECTION, which does not follow the rules of COBOL syntax.
- Action:** Edit your source code so that EXTERNAL is used as a clause within the record description only in the WORKING-STORAGE SECTION.
- 242-S Only allowed in WORKING-STORAGE and FILE SECTIONS**
- Cause:** You have used the GLOBAL clause in a file or report description entry that is not in the WORKING-STORAGE SECTION or the FILE SECTION. This does not follow the rules of COBOL syntax.
- Action:** Edit your source code to ensure that GLOBAL is used as a clause to the file or report description only within the WORKING-STORAGE SECTION and/or the FILE SECTION.
- 243-S VALUE of group item must be nonnumeric or figurative constant**
- Cause:** If you specify a VALUE clause for a group item, the value in the clause must be either a nonnumeric literal or a figurative constant.
- Action:** Correct the value in the VALUE clause.
- 244-S FD missing for file**
- Cause:** You have not made an FD entry for all the files named in the file control paragraph.
- Action:** Ensure that there is an FD entry for each file named in a SELECT clause in the file control paragraph.
- 245-S DEPENDING ON item missing or illegal**
- Cause:** You have not declared a data item named in the DEPENDING phrase in an OCCURS clause.
- Action:** Add a declaration of the missing data item.
- 246-S KEY missing or illegal**
- Cause:** You have not specified the RECORD KEY clause in the file description of an indexed sequential file.
- Action:** Add the RECORD KEY clause to the file description.
- 247-S Index-name has been declared explicitly**
- Cause:** You have declared an index data item explicitly. The item is declared implicitly by its appearance in an INDEX phrase, so you have in effect declared the same item twice.
- Action:** Delete the explicit declaration of the index data item.
- 248-S ISAM key too long**
- Cause:** A data item that is to be used as an indexed sequential file key is longer than the maximum length allowed for such keys.
- Action:** Redefine the key length.

- 249-S Alternate keys have same reference**
- Cause:** You have defined two alternate, overlapping keys for an indexed sequential file. Alternate keys must be completely distinct from one another.
- Action:** Redefine the alternate keys so that they do not overlap.
- 250-S STATUS field missing or illegal**
- Cause:** You have not declared the data item specified in the FILE STATUS clause of a file control entry, or you have declared it incorrectly.
- Action:** Refer to the *Language Reference* for details of the correct form of the FILE STATUS data item. Add or correct the definition of the FILE STATUS data item.
- 251-S CURSOR field missing or illegal**
- Cause:** You have not declared the data item specified in the CURSOR IS clause in the SPECIAL-NAMES paragraph. Alternatively, you have declared the item incorrectly.
- Action:** Refer to the *Language Reference* for the correct form of the cursor data item. Add or correct the declaration of the data item in the CURSOR IS clause.
- 252-S PASSWORD field missing or illegal**
- Cause:** You have not declared the data item specified in the PASSWORD clause in a file description entry. Alternatively, you have declared the item incorrectly.
- Action:** Refer to the *Language Reference* for the correct form of the password data item. Add or correct the declaration of the data item in the PASSWORD clause.
- 254-S 'VALUE OF' field missing or illegal**
- Cause:** You have not declared one or more of the data items named in the VALUE OF clause of an FD entry in the file section. Alternatively, you have declared such an item incorrectly.
- Action:** Refer to the *Language Reference* for the correct form of these data items. Add or correct the declaration of the data item(s) in the VALUE OF clause.
- 255-S User name same as special register**
- Cause:** You have declared a data item with the same name as one of the special registers, which are data items that are automatically declared by the compiler.
- Action:** Refer to the *Language Reference* for a list of the names of these special registers. Alter the name of the data item in error.
- 256-S Preceding record has zero length**
- Cause:** You have defined a data item with zero length. This is probably due to errors in the elementary item descriptions in the record definition.
- Action:** Correct the record description.

- 257-S KEY data-name missing or already declared**
Cause: You have not declared a data item specified in the KEY phrase of an OCCURS clause, or you have declared it twice.
Action: Declare the data item or remove the extra declaration.
- 258-S ASSIGN data-name illegal**
Cause: You have specified an ASSIGN data-name in a SELECT ... ASSIGN statement that is not unitary; that is, the data definition contains an OCCURS clause.
Action: Edit your source code to ensure that the ASSIGN data item does not include an OCCURS clause.
- 259-S Illegal report-name or bad RD clause**
Cause: You have defined a report name that is not unique or does not conform to the rules for user-defined words. Alternatively, you have specified a clause in a record description (RD) entry incorrectly.
Action: Refer to the *Language Reference* for the correct syntax of an RD entry, and correct your program accordingly.
- 260-S Inconsistent page specification**
Cause: The values you have specified in the PAGE LIMIT clause are not consistent. For example, the integer in the LAST DETAIL phrase is smaller than the integer in the FIRST DETAIL phrase.
Action: Refer to the *Language Reference* for the rules governing the PAGE LIMIT clause, and correct your program accordingly.
- 261-S Only allowed in REPORT SECTION**
Cause: You have tried to specify syntax for a report outside the REPORT SECTION of the Data Division.
Action: Refer to the *Language Reference* to see what syntax is allowed in the REPORT SECTION. Delete the syntax or relocate it in the REPORT SECTION.
- 262-S Not a CONTROL for this report**
Cause: The data-name in a TYPE CH or TYPE CF clause does not appear in the CONTROL clause of the associated RD entry. This may be due to misspelling the data-name.
Action: Ensure that the correct data-name appears in both the TYPE CF/CH and CONTROL clauses.
- 263-S Not allowed when PAGE not specified in RD**
Cause: You have specified a TYPE PH or TYPE PF clause without specifying a PAGE clause in the associated RD entry. Alternatively, you have specified an absolute line number in the LINE NUMBER clause without specifying a PAGE clause in the associated RD entry.
Action: Add a PAGE clause to the RD entry.

- 264-S Only one report group with this TYPE allowed per RD**
Cause: You have specified a duplicate of one of the TYPE clauses (for example, you have specified two TYPE PF clauses). At most, one clause of each type is allowed in a particular RD entry.
Action: Delete the duplicate TYPE clause.
- 265-S Not allowed with this TYPE**
Cause: You are using a qualifier in a statement that does not allow any qualifiers, or that does not allow this particular qualifier.
Action: Refer to the *Language Reference* for details of the qualifiers allowed with this statement.
- 266-S No TYPE specified**
Cause: All level 01 entries in the REPORT SECTION must have a TYPE clause.
Action: Add a TYPE clause.
- 267-S LINE specification missing or inconsistent**
Cause: There are three possible causes of this message:
- You specified a NEXT PAGE clause that was not in the first LINE clause of a report group description entry.
 - The absolute line numbers in the LINE clauses of a report group description entry are not in ascending order.
 - You have not specified a LINE clause in a particular report group description entry.
- Action:** To correct the fault:
- Delete the NEXT PAGE clause from all but the first LINE clause.
 - Rearrange the LINE clauses so that absolute line numbers are in ascending order.
 - Add a LINE clause.
- 268-S REPORT specified in more than one FD**
Cause: You have specified the same report name in more than one FD.
Action: Delete the duplicate report name.
- 269-S Duplicate CONTROL field**
Cause: You have specified the same CONTROL field value in more than one RD entry.
Action: Delete the duplicate CONTROL field value.
- 271-S Only allowed with DETAIL groups**
Cause: You can specify a GROUP INDICATE clause only with DETAIL report groups.
Action: Delete the GROUP INDICATE clause.

- 272-S Only allowed with CONTROL FOOTING groups**
Cause: You can specify a SUM clause only in a control footing report group.
Action: Delete the SUM clause.
- 273-S Non-elementary item has invalid qualifier, or PICTURE missing**
Cause: You have specified at a group level a qualifier that can only be used at an elementary level.
Action: Delete the clause that is in error and recompile your program.
- 274-S GROUP INDICATE without COLUMN**
Cause: You have specified the GROUP INDICATE clause, but no COLUMN clause is in the same report group description.
Action: Include the COLUMN qualifier.
- 275-S NEXT GROUP not allowed with this group TYPE**
Cause: You have specified NEXT GROUP in a group with TYPE RF or PH.
Action: Delete the NEXT GROUP clause.
- 276-S NEXT GROUP NEXT PAGE not allowed with this group TYPE**
Cause: You have specified NEXT GROUP or NEXT PAGE in a group with TYPE PF.
Action: Delete the clause.
- 277-S LINE NEXT PAGE not allowed with this group TYPE**
Cause: You may specify the LINE NEXT PAGE clause only with a group of TYPE CH, CF, or DE.
Action: Delete the LINE NEXT PAGE clause.
- 278-S RESET item is lower CONTROL level than group**
Cause: The control group on which a sum is reset must be at the same level as or a lower level than the sum.
Action: Refer to the *Language Reference* for the rules governing the resetting of sums, and change your code accordingly.
- 279-S Report line too long**
Cause: You have written a report line which exceeds the maximum permitted length.
Action: Correct your source code to ensure that the report line in question does not exceed the maximum permitted length.
- 282-S CICS facility not supported**
Cause: You have compiled your program with the CICS option, which is not supported by AIX VS COBOL.
Action: Do not use the CICS option when you compile. Also, remove any code that served as an interface to CICS.

- 283-S An EXTERNAL file cannot be subject of SAME RECORD AREA clause**
- Cause:** You have specified a file in a SAME RECORD AREA clause in your program that is specified as an EXTERNAL file in an FD entry.
- Action:** Delete the EXTERNAL file from the SAME RECORD AREA clause in your program.
- 284-S Not allowed in REPORT SECTION**
- Cause:** The syntax you have used should not appear within the REPORT SECTION.
- Action:** Remove the syntax in question from the REPORT SECTION.
- 301-S Unrecognized verb**
- Cause:** You have used a verb in the Procedure Division of your program that the compiler does not recognize as a valid COBOL verb. Alternatively, you may have misspelled a COBOL verb.
- Action:** Refer to the *Language Reference* to see which verbs are permitted within the COBOL language. Ensure that you have spelled them correctly.
- 302-S IF ... ELSE or scope-delimiter mismatch**
- Cause:** There is an error in your coding of the IF statement within the Procedure Division of your program. The two halves of one IF statement do not match. You may have more ELSE phrases than IF phrases.
- Alternatively, you have made an error in coding a construct that uses one of the scope delimiters (for example, END-ADD). There is a mismatch between the number of scope delimiters and the statement whose scope they delimit.
- Action:** Correct your program accordingly.
- 303-S Operand has wrong data-type**
- Cause:** You have used a data item with the wrong data-type in one of your statements. For example, this message would be displayed if you used a file name in a WRITE statement instead of a record name.
- Action:** Correct the data item.
- 304-S Procedure name not unique**
- Cause:** Two or more sections within the Procedure Division or two or more paragraphs within a section of your program have the same title. COBOL rules state that each section or paragraph name must be unique.
- Action:** Rename or qualify the sections or paragraphs to ensure uniqueness of reference.
- 305-S Procedure name same as data-name**
- Cause:** A paragraph within the Procedure Division of your program has the same title as a data item declared within the Data Division.
- Action:** Retitle either the paragraph or the relevant data item to ensure uniqueness of reference.

- 306-S Entry name not unique**
Cause: You have used the same entry-point name more than once in your program.
Action: Alter the entry name or qualify it.
- 307-S Wrong combination of data types**
Cause: You are trying to manipulate data items that are not compatible.
Action: Ensure that the data items are of the same type.
- 308-S Conditional statement not allowed in this context**
Cause: You have used a conditional statement (one which specifies that the truth value of a condition is to be determined at run time) in the Procedure Division of your program where an imperative statement is expected.
Action: Replace the statement with one that begins with an imperative verb and is followed by a specification of an unconditional action that is to be taken at run time.
- 309-S Malformed subscript**
Cause: The most common cause of this message is that you have specified two subscripts for a one-dimensional table item.
Action: Correct your code to ensure that you do not specify more than one subscript for a one-dimensional table item.
- 310-S ACCEPT/DISPLAY wrong or Communications syntax incorrect**
Cause: You have used an invalid piece of syntax with an ACCEPT or DISPLAY statement. Alternatively, you have coded the Communications syntax incorrectly. The most likely cause of this message is that you have a spelling error in your code.
Action: Correct the spelling error.
- 311-S Illegal syntax used with I-O verb**
Cause: The code following an I-O verb (for example, READ or WRITE) violates the rules of COBOL syntax.
Action: Refer to the *Language Reference* for details on the usage of the particular verb in your code that has caused this message.
- 312-S Invalid arithmetic statement**
Cause: An arithmetic statement used in the Procedure Division of your code does not conform to the rules of COBOL syntax. These statements must begin with an arithmetic verb (for example, SUBTRACT or DIVIDE) and should be followed by the relevant numeric literals or identifiers which the verb will act upon when the program is executed. The statement that you have specified is not a valid one.
Action: Refer to the *Language Reference* for details of arithmetic statements and ensure that the one you wish to use conforms to the relevant rules.

- 313-S Invalid arithmetic expression**
Cause: You have used an invalid arithmetic expression in the Procedure Division.
Action: Refer to the *Language Reference* for details on the more complicated coding that COBOL allows you to write to carry out complicated mathematical tasks.
- 314-S Illegal key**
Cause: The key value in a file operation or in a SEARCH statement is the wrong size.
Action: Check the key definition and correct the key value.
- 315-S Invalid conditional expression**
Cause: A conditional expression that you have specified in the Procedure Division of your program does not conform to the rules of COBOL syntax. These expressions, an example of which is the statement immediately following an IF, allow one of two following statements to be executed at run time depending on the truth value.
Action: Refer to the *Language Reference* for details on the coding of the particular statement which you have used.
- 316-S Too many AFTERS in PERFORM statement**
Cause: A PERFORM statement may only be followed by six AFTER phrases. Your code exceeds this limit.
Action: Rewrite your code ensuring that no PERFORM statement has more than six associated AFTER phrases.
- 317-S Incorrect structure of Procedure Division**
Cause: This message is displayed if you have made a mistake in the coding of the Procedure Division. For example, you will receive this message if you have forgotten one of the section headings in this division.
Action: Ensure that the Procedure Division follows a logical order and that each section within it has its own heading. You may also need to use the `nestcall` option if your program is nested.
- 318-S File must have ACCESS SEQUENTIAL**
Cause: The file named in the GIVING phrase of a SORT or MERGE statement must be sequential.
Action: Change the organization of the file to SEQUENTIAL.
- 319-S Only index names allowed with this format**
Cause: With the operation you are attempting, an index name is required at this point.
Action: Correct your source code to ensure that the name you have specified is an index name.

- 320-S Too many operands in one statement**
- Cause:** A statement in the Procedure Division of your program contains too many operands, or the individual operands are too long.
- Action:** Refer to the *Language Reference* for details of the correct number of operands for this statement, and amend it accordingly, or shorten those operands that are too long.
- 321-S Only one GIVING file allowed**
- Cause:** You have specified more than one file name in the GIVING phrase of a SORT or MERGE statement, where only one file name is allowed.
- Action:** Delete any additional file names from the GIVING phrase.
- 322-S Cannot reference DEBUG-ITEM outside declaratives**
- Cause:** You have referred to the compiler-generated data item DEBUG-ITEM in a procedure that is not in the declarative section of your program's Procedure Division. References to DEBUG-ITEM are only permitted in the declarative section.
- Action:** Delete the reference to DEBUG-ITEM, or relocate it within the declarative section.
- 323-S More than one USE procedure on same file**
- Cause:** You have associated two or more USE procedures with the same file. You can associate, at most, one USE procedure with a file.
- Action:** Delete the additional USE references.
- 324-S More than one USE procedure for same open mode**
- Cause:** You have associated two or more USE procedures with the same file that is in the open mode. You can associate, at most, one USE procedure with a file in the open mode.
- Action:** Delete the additional USE references.
- 325-S Illegal combination of debugging procedures**
- Cause:** You have specified an invalid combination of USE FOR DEBUGGING procedures in the declarative section.
- Action:** Refer to the *Language Reference* for details of debugging procedures. Correct the declarative section.
- 326-S Literal cannot be receiving field**
- Cause:** You have specified a literal value as the receiving field in an operation involving an implicit or explicit move. A receiving field must be a data item.
- Action:** Change the literal value of the receiving field to a data item.
- 327-S Index item not permitted**
- Cause:** You have named an index data item as the sending or receiving field in a MOVE statement. This is not permitted.
- Action:** Move the value into a nonindex data item, and use this data item in the MOVE statement.

- 329-S WHEN phrase missing from SEARCH statement**
Cause: You have specified a SEARCH statement with no WHEN phrase. You must specify at least one WHEN phrase in a SEARCH statement.
Action: Add a WHEN phrase to the SEARCH statement.
- 330-S Not a record name**
Cause: You must specify the name of a file record in this context (as defined in an FD entry in the file section). You have probably misspelled a valid record name.
Action: Correct the reference to the file record.
- 331-S Program is nested. Must compile with nestcall option**
Cause: You are trying to compile a program that has nested COBOL source code without using the **nestcall** option.
Action: Recompile using the **nestcall** compiler option.
- 332-S AFTER phrase not allowed with in-line perform**
Cause: You cannot specify an AFTER phrase in an in-line PERFORM statement.
Action: Delete the AFTER phrase.
- 333-S Not an ALTERable paragraph**
Cause: The paragraph you have named in an ALTER statement is not an ALTERable paragraph. An ALTERable paragraph must consist of a single sentence containing only a single GO TO statement without a DEPENDING phrase.
Action: Change the reference in the ALTER statement to refer to a paragraph that is ALTERable, or edit the named paragraph so that it is an ALTERable paragraph.
- 334-S Cannot follow WHEN OTHER**
Cause: The WHEN OTHER phrase, if specified, must be the last phrase in an EVALUATE statement.
Action: Move any WHEN phrases that come after the WHEN OTHER phrase so that they are then in front of the WHEN OTHER phrase.
- 335-S Selection object does not match selection subject**
Cause: There is a type mismatch between one of the selection subjects in an EVALUATE statement and the corresponding selection object.
Action: Refer to the *Language Reference* for the correct syntax of EVALUATE. Correct the EVALUATE statement.
- 336-S Variable-length group not allowed**
Cause: You cannot INITIALIZE a variable-length item.
Action: Delete the item name from the INITIALIZE statement.

- 337-S Cannot repeat same category**
Cause: You have repeated the same data class in the REPLACING phrase of an INITIALIZE statement.
Action: Delete the repeated class from the INITIALIZE statement.
- 338-S REPORT not specified in an FD**
Cause: You have referred to a report for which there is no FD entry in the REPORT SECTION. You may have forgotten to specify the report entry, or you may have misspelled a correct report name.
Action: Correct the report name, or add the necessary report definition to the REPORT SECTION.
- 339-S Not allowed with SEQUENTIAL files**
Cause: You have specified an operation that cannot be performed on a file opened in the sequential access mode.
Action: Refer to the *Language Reference* for the details of what operations you can perform on SEQUENTIAL files.
- 340-S Not allowed with RANDOM ACCESS files**
Cause: You have specified an operation that cannot be performed on a file opened in the RANDOM ACCESS mode.
Action: Refer to the *Language Reference* for the details of what operations you can perform on RANDOM ACCESS (relative or indexed sequential) files.
- 341-S Not allowed with LINE SEQUENTIAL files**
Cause: You have specified an operation that cannot be performed on a LINE SEQUENTIAL file.
Action: Refer to the *Language Reference* for the details of what operations you can perform on LINE SEQUENTIAL files.
- 342-S Not allowed with LINAGE files**
Cause: You have performed an illegal operation on a file defined with a LINAGE clause, for example, WRITE AFTER CO1.
Action: Delete the illegal operation, or redefine the file without a LINAGE clause.
- 343-S Should be declared in LINKAGE SECTION**
Cause: You have included a data reference that is not declared in the LINKAGE SECTION of the Data Division.
Action: Ensure that the item is declared in the LINKAGE SECTION.
- 344-S Should be level 01 or 77**
Cause: You have specified a data item that is not a level 01 or level 77 data item where the compiler requires a data item to be one of these levels.
Action: Redefine the data item to be level 01 or level 77.

- 345-S USING parameter used twice in parameter list**
Cause: You have specified the same data item name twice in the USING phrase of the Procedure Division header. All the names in the USING phrase must be different.
Action: Delete the extra reference.
- 346-S Only one WHEN phrase allowed with SEARCH ALL**
Cause: You have specified two WHEN phrases in a SEARCH statement using the ALL option. You can specify only one WHEN phrase in this context.
Action: Delete the extra WHEN phrase.
- 347-S MERGE needs at least two USING files**
Cause: You have specified less than two file names in the USING phrase of a MERGE statement. You must enter at least two files for a merge operation.
Action: Ensure that there are at least two file names in the USING phrase of the MERGE statement.
- 348-S Procedure name undeclared**
Cause: You have referred to a nonexistent paragraph or section. You have probably misspelled a valid procedure name.
Action: Ensure that the procedure to which you referred exists, and that you have spelled it correctly.
- 349-S 'LOCK' clause expected**
Cause: You have used a READ statement with no LOCK phrase on a file that requires you to specify one.
Action: Insert a LOCK phrase in the READ statement.
- 350-S Illegal use of 'NO LOCK'**
Cause: You have used a READ statement with the NO LOCK phrase on a file for which no record locking is required.
Action: Delete the NO LOCK phrase.
- 351-S 'LOCK' clause specified for 'EXCLUSIVE' file**
Cause: You have specified a LOCK clause in a READ statement for a file that you have already locked with an EXCLUSIVE lock, in the FILE CONTROL paragraph of your code.
Action: Record locking is impossible on a file that your run-unit has already locked with an EXCLUSIVE lock. To ensure that the two LOCK entries are compatible, you will have to either delete the LOCK entry in the relevant READ statement, or alter the LOCK MODE IS entry in the FILE-CONTROL paragraph.

- 352-S 'KEPT' specified for file with single record locking**
- Cause:** This message applies only to multiuser syntax. You have specified a WITH KEPT LOCK phrase in a READ statement which uses a file that does not support multiple record locking.
- Action:** Either change the file type or delete the WITH KEPT LOCK phrase to ensure compatibility between the two entries.
- 353-S 'KEPT' omitted for file with multiple record locking**
- Cause:** This message applies only to multiuser syntax. In the context of your program the READ statement for this specific file requires a WITH KEPT LOCK phrase to allow it to control access to that file.
- Action:** Insert the WITH KEPT LOCK phrase.
- 355-S Only '=' and 'NOT =' allowed for pointer data items**
- Cause:** You can only use the operators = and NOT = in comparisons involving pointer data items.
- Action:** Correct the comparison.
- 356-S Not allowed with REPORT files**
- Cause:** You have attempted to perform a file operation (READ, WRITE, REWRITE) on a file whose FD entry indicates that it is a report.
- Action:** Delete the input-output statement.
- 357-S Screen is display-only**
- Cause:** You have named a display screen in an ACCEPT statement that contains only display fields.
- Action:** Make sure that you have named the correct display screen.
- 358-S Missing comma**
- Cause:** AIX VS COBOL syntax requires a comma at this point.
- Action:** Refer to the *Language Reference* for the correct syntax. Add the missing comma.
- 359-S Mismatch of table dimensions**
- Cause:** There is a mismatch between the number of dimensions in a data item in an ACCEPT/DISPLAY statement and the corresponding screen section item.
- Action:** Make the two item definitions consistent.
- 360-S File must have ACCESS DYNAMIC**
- Cause:** You have specified an operation on a file that can be performed only if the file has access mode DYNAMIC.
- Action:** Change the file control entry for the file so that it has access mode DYNAMIC.

- 362-S Data name not declared for file or of wrong type for CODE-SET**
Cause: The data-names you have specified in the CODE-SET clause do not belong to a record in the file.
Action: Edit your source code so the data-names in the CODE-SET clause refer to a record in the file.
- 363-S Data name not in the same record as first item in CODE-SET clause**
Cause: You have specified data-names in the CODE-SET clause that belong to more than one record. All data-names specified in this clause must belong to a single record in the file.
Action: Edit your source code to delete one or more of the CODE-SET data-names to ensure that all data-names belong to only one record.
- 364-S Data name overlaps another item in CODE-SET clause**
Cause: One or more of the data-names specified in the CODE-SET clause is redefined, or you may have specified the whole record rather than individual data-names contained in that record.
Action: Edit your source code so that the data names within the record are specified only once.
- 365-S Variable size table not last in group or subsidiary to OCCURS**
Cause: OCCURS DEPENDING ON item must be the last item in a group.
Action: Revise your code to make the OCCURS DEPENDING ON item last in the group.
- 366-S Variable length delimiter not allowed**
Cause: You cannot use a reference modified item or a group containing an OCCURS...DEPENDING ON clause in this context.
Action: Change the indicated item to reference a fixed-length object and recompile.
- 367-S Description of operand does not contain the INDEXED BY clause**
Cause: A SEARCH is not possible because the description of the operand does not contain the INDEXED BY clause.
Action: Add the INDEXED BY clause to enable the SEARCH action.
- 368-S Exception phrase inappropriate**
Cause: The specified exception is not appropriate with this statement in the specified access mode. For example, an AT END has been specified with random access, or INVALID KEY with sequential access.
Action: Correct your code to use the correct exception syntax for the access mode you are using.
- 369-S OPEN EXTEND on file with LINAGE clause**
Cause: OPEN EXTEND is not allowed on a file defined with a LINAGE clause.
Action: Remove the LINAGE clause specification, or OPEN the file in another mode.

- 370-S Operand must be a table**
Cause: The operand used in this context can only be a table.
Action: Change your code to reference a table here.
- 384-S NEXT SENTENCE does not follow IF, ELSE, or SEARCH WHEN**
Cause: The NEXT SENTENCE syntax has been used incorrectly, or an IF or ELSE statement is missing.
Action: Refer to the *Language Reference* to correct your syntax.

Compiler Error Messages

- 001-E Character other than *, D, /, -, \$, or £ found in col 7. Blank assumed.**
You may have mistyped one of the characters allowed in column 7. The compiler cannot interpret the character in column 7 and has treated it as a space.
- 002-E Continuation character invalid at this point. Blank assumed.**
You have placed a hyphen in column 7, though the compiler is not expecting the syntax to be continued at this point. The continuation character is ignored.
- 003-E First character of a continued literal not a quote. Quote assumed.**
You have included a continuation character in column 7, but have forgotten to start the continuation of the literal with a quote. The compiler assumes that the quote is included.
- 004-E Continuation character expected. End of literal assumed.**
The literal in the previous line of source code is not delimited by quotes, so the compiler is expecting a continuation character in column 7, and a continuation of the literal. The compiler has assumed that you meant to end the literal on the previous line.
- 005-E Name ends in hyphen. Processed as written.**
You have used a hyphen as the last character in a user-defined name, which is against the rules of COBOL syntax. The compiler has accepted this as a valid name, however, and has not changed the name in any way.
- 006-E COBOL word contains more than 30 characters. Word truncated.**
The name you have specified is longer than 30 characters. The compiler treats this as a name consisting of the first 30 characters of the original name.
- 007-E VALUE literal too large. Literal truncated.**
The literal you have specified in the VALUE clause is too long to fit into the data item. The compiler inserts characters from the literal into the data item, until the data item is full.
- 008-E DBCS literal needs an even number of characters. Literal truncated.**
All DBCS (Double Byte Character Set) symbols are two bytes long. You have specified a literal that consists of an odd number of characters. The AIX VS COBOL system ignores the last, single character of the literal.

- 009-E Closing delimiter for DBCS literal not found. Delimiter assumed.**
 You have not included the quote to show the end of the DBCS (Double Byte Character Set) literal. The AIX VS COBOL system has assumed that you intended to end the literal at this point.
- 010-E Nonnumeric literal has length of zero. One SPACE assumed.**
 You have defined an alphabetic or alphanumeric literal in your source code that is empty; that is, you have a pair of quotes with no character between them. The compiler has assumed that the literal contains one space character.
- 011-E DBCS literal length zero. Length of one DBCS character assumed.**
 You have defined a DBCS (Double Byte Character Set) literal in your source code which is empty, that is, you have a pair of quotes with no DBCS character between them. The AIX VS COBOL system has assumed that the literal is two characters long, and that it has a value of spaces.
- 012-E DIVISION missing or misspelled. DIVISION assumed.**
 You have omitted the word DIVISION from a division header, or you have spelled it incorrectly. The compiler has assumed that DIVISION was intended.
- 013-E SECTION missing or misspelled. SECTION assumed.**
 You have omitted the word SECTION from a section header, or you have spelled it incorrectly. The compiler has assumed that SECTION was intended.
- 014-E Period missing. Period assumed.**
 You have omitted a period in a place where one was expected by the rules of COBOL syntax. The compiler has assumed the period is present.
- 015-E OCCURS integer-1 exceeds OCCURS integer-2. 0 assumed for integer-1.**
 You have included the OCCURS integer-1 TO integer-2 DEPENDING clause in your source code, but the value given for integer-1 is greater than that for integer-2, which does not follow the rules of COBOL syntax. The compiler has effectively changed the value of integer-1 to 0.
- 016-E Expected SEPARATE before CHARACTER in SIGN clause. SEPARATE assumed.**
 You have wrongly coded the SIGN clause because you have included the word CHARACTER, which is not required, but you have omitted the required word SEPARATE. The compiler has assumed that you intended the clause to be SIGN IS LEADING (or TRAILING) SEPARATE CHARACTER.
- 017-E REDEFINES ignored for 01 level item in FILE or COMMUNICATION SECTION.**
 You have tried to REDEFINE a data-item in the FILE SECTION, but the data-item is an 01 level item. This is not valid COBOL syntax. The REDEFINES clause is ignored.

- 018-E VALUE clause not allowed here. Clause processed as comment.**
You have tried to assign a value to a data item defined in the FILE SECTION or the LINKAGE SECTION. The VALUE clause is ignored for compilation purposes.
- 019-E Unsigned numeric literal expected. Sign ignored.**
You have specified a sign where one was not expected. For example, you may have used the AFTER ADVANCING +1 or BEFORE ADVANCING -2 clause to the WRITE statement. The compiler has ignored the sign, and has treated the digits as a positive value.
- 020-E Unsigned numeric field. Sign in VALUE clause ignored.**
You have defined a numeric data item, PIC 9, and have tried to assign a signed number as its value. The compiler ignored the sign you have specified. Where you are attempting to move a number with a negative sign into a numeric field, the result will not be as expected.
- 021-E Slack bytes added in conversion of COMP-6 to COMP.**
You are using a program converted from RM/COBOL to AIX VS COBOL that contains a COMP-6 numeric data item (which has been converted to a COMP numeric data item). As a result of this conversion, less data space may have been allocated to the numeric data item and, therefore, the AIX VS COBOL system adds leading binary zeros to pad the space if required.
- 023-E WORKING-STORAGE SECTION expected. Start of WORKING-STORAGE assumed.**
Your program begins with a Level 01 entry. It is assumed that this is the first item of WORKING-STORAGE.
- 024-E VALUE clause literal does not conform to PICTURE. Changed to blanks.**
A numeric value has been specified for a non-numeric data item. The data item will be filled with spaces.
- 025-E Move edited field to edited field - treated as alphanumeric move.**
A move from one edited field to another is treated as an alphanumeric move.
- 026-E Source literal is non-numeric — substituting zero.**
A MOVE statement is trying to MOVE a non-numeric literal to a numeric data item. This cannot be done. To avoid undefined results, zero will be moved to the data item.
- 027-E Literal is numeric — treated as non-numeric.**
A numeric literal is being used in relation to a non-numeric data item; for example, as the VALUE of a level 88 entry attached to a non-numeric data item. The literal will be converted to an alphanumeric literal.
- 028-E Statement should not reference an alphabetic data item.**
A statement would cause invalid data to be contained in an alphabetic data item. To prevent this situation change your source program to reference a numeric or alphanumeric data item and resubmit it to your COBOL system.

- 029-E A non-integer is being moved to an alphanumeric data item.**
 A MOVE statement is trying to MOVE a non-integer item to an alphanumeric data item. This cannot be done. To avoid undefined results, correct your source code and recompile your program.
- 030-E Cannot SORT or MERGE USING or GIVING two files with SAME AREA.**
 Two files specified in a SORT or MERGE statement are defined as sharing the SAME AREA. This is invalid. Change your source code so that the files use different AREAs and recompile your program.
- 031-E SORT file appears in more than one SAME SORT(-MERGE) AREA clause.**
 A SORT file has been defined in conjunction with more than one SAME SORT (-MERGE) AREA clause. This is invalid. Change your source code so that the files use different AREAs and recompile your program.
- 032-E File-names illegally specified in same SAME RECORD AREA clause.**
 Two files specified in a SORT or MERGE statement are defined as sharing the SAME AREA. This is invalid. Change your source code so that the files use different AREAs and recompile your program.
- 033-E File-names illegally specified in same SAME SORT(-MERGE) AREA clause.**
 A SORT file has been defined in conjunction with more than one SAME SORT(-MERGE) AREA clause. This is invalid. Change your source code so that the files use different AREAs and recompile your program.
- 034-E Source item is ALPHABETIC — treated as alphanumeric.**

Compiler Warning Messages

- 101-W No COBOL statement between periods.**
 You have placed one period immediately following another period. This is not against the rules of COBOL syntax, but it may indicate a fault in your program. For example, you may have wanted to include a line of source code here.
- 102-W Blank continuation source line. Line ignored.**
 You have placed a hyphen in column 7, but the rest of the line contains no other code. The next line should also contain a hyphen in column 7 to continue correctly.
- 103-W Sequence number out of order or missing.**
 You have compiled your program with the `seqchk` directive on, and the compiler is indicating an error in the sequence numbers.
- 104-W 77 level item in FILE SECTION processed as 01 level.**
 You have assigned a level of 77 to a data item in the FILE SECTION, which is against the rules of COBOL syntax. The compiler assumes that you had intended to code this as an 01 level item, and processes it as such.

- 105-W No CORRESPONDING items were found. Statement has no effect.**
The compiler found no matching data items for the CORRESPONDING clause, so no intermediate code was produced for this statement.
- 106-W ZERO value for BY operand. Statement processed as written.**
You have used the BY operand to the verb PERFORM, but the value you have specified for the increment is zero. The compiler can produce code to execute this statement, but the value is never incremented.
- 107-W Statement exceeds COMP subset.**
The COMP subset code you have written could be rewritten to execute more efficiently.
- 108-W Signed numeric compared with group. Processed as alphanumeric compare.**
You have written your code so that a signed numeric field is compared with a group item. The compiler has treated the signed numeric field as an alphanumeric field for the comparison. There are no problems with comparing a signed numeric field to an elementary item.
- 109-W WITH DEBUGGING MODE not specified. Section ignored.**
Your program includes the USE FOR DEBUGGING statement in the declarative section of your Procedure Division, but the WITH DEBUGGING MODE clause in the SOURCE-COMPUTER paragraph is omitted. The compiler ignores all code within this declarative section.
- 110-W First literal is greater than second. Processed as written.**
In the ALPHABET clause, the first literal specified has a value greater than that of the second; for example, P THRU D. The compiler accepts this as written; that is, the characters are processed in reverse order.
- 111-W Boundary violation. Processed as written.**
You have attempted to access an item beyond the end of a table. The compiler will generate code to access the appropriate line of code beyond the end of the table, but the result of this is undefined.
- 112-W Compatibility directive forcing non-standard behavior.**
When the RM option is set and an alphanumeric data item is MOVED to a numeric data item that is defined as larger, the compiler adds space characters to the front of the MOVED data item. However, under the RM/COBOL system, leading zeros are added to the front of the alphanumeric data item when it is MOVED to a numeric data item that is defined as larger.
- 113-W Imperative statement missing.**
Your program contains a conditional statement that has not been followed by an imperative statement. The compiler will execute the code, but the result of this may be undefined.
- 114-W Clause treated as documentary.**
You have assigned more than one external file reference to a SELECT ... ASSIGN clause. The compiler will accept the first external file reference, but will treat all remaining external file references in that clause as documentary.

- 115-W SAME AREA treated as SAME RECORD AREA.**
 You are using a program converted from DG Interactive COBOL to AIX VS COBOL that contains a SAME AREA clause. The compiler has assumed that the SAME RECORD AREA clause is what was intended.
- 116-W Accept qualifier used with display-only field - qualifier ignored.**
 The qualifier you used with a DISPLAY-only field, or in a DISPLAY statement, can only be used with an ACCEPT. It is ignored.
- 117-W Zero suppression after floating insertion - treated as floating ins.**
 As an example:
 PIC ++z.zz
 is treated as
 PIC +++.++.
- 118-W Indexed-name belongs to a different table.**
 The index used to subscript a table item is not one used in the INDEXED BY clause for this table. The results will be unpredictable.
- 119-W Record < minimum size given in FD statement.**
 The definition of a record following an FD clause is smaller than the minimum size specified in the RECORD CONTAINS phrase of that FD clause.
- 120-W Record > maximum size given in FD statement.**
 The definition of a record following an FD clause is larger than the maximum size specified in the RECORD CONTAINS phrase of that FD clause.
- 121-W VALUE clause in FILE or LINKAGE SECTION. Processed as comment.**
 A data item in the FILE or LINKAGE SECTION cannot be given a value. The VALUE clause specified will be ignored.
- 122-W Period must be followed by a space.**
 In all cases, a period must be followed by a space.
- 123-W Neither NAMED nor CHANGED specified. Treated as formatted DISPLAY.**
 An EXHIBIT statement has been used without the NAMED or CHANGED phrase. The resulting DISPLAY will be formatted by separating each item with a space.
- 124-W CALL parameter is literal (or LENGTH OF). BY CONTENT assumed.**
 You are using a program converted from RM/COBOL to AIX VS COBOL that contains a CALL...USING literal statement. The compiler has assumed that CALL...USING BY CONTENT literal is what was meant.
- 125-W Punctuation character not followed by a space. Assume space.**
 You have used a punctuation character that is not followed by a space. The compiler will assume that a space is present.

- 126-W Punctuation character not preceded by a space. Assume space.**
You have used a punctuation character that is not preceded by a space. The compiler will assume that a space is present.
- 127-W Double-Byte character(s) may be corrupted by use of this move.**
This message is issued when a MOVE is done from a PIC G to a PIC X data item, where the source is longer than the target, and the target has an odd number of bytes. You can prevent the possible corruption by making the PIC X data item an even number of bytes. Truncation will still occur.
- 128-W No STOP RUN, GOBACK, or EXIT PROGRAM statements found in source.**
No explicit syntax was found to terminate the execution of this code. The end of the source file will be used as the end of code to execute.
- 129-W Statement cannot be reached.**
The compiler has determined that the marked statement in your source file cannot be reached during execution due to the structure of your code or the flow of control in it.
- 130-W Prefix of filename treated as documentary.**
In an external assignment-name, only that portion of the name to the right of the right-most '-' character is treated as significant.
- 132-W Unable to validate contents of DBCS literals.**
AIX VS COBOL is unable to check that the contents of a DBCS literal are valid. If the contents are invalid the results will be unpredictable.
- 134-W Entry name would be changed by OS/VS COBOL and VS COBOL II.**
The name you specified for an ENTRY statement would be altered if your source program were processed by an OSVS or VSC2 compiler.

Compiler Information Messages

- 201-I Zero suppression PICTURE string overrides BLANK WHEN ZERO clause**
- 202-I Original item is larger than redefinition**
- 203-I LABEL clause processed as comment**
- 204-I BLOCK CONTAINS clause processed as comment**
- 205-I Previous paragraph or SECTION contains no statements**
- 206-I Procedure Division does not start with a SECTION**
- 207-I Original item is smaller than redefinition**
- 208-I USE clause omitted**
- 209-I COMP-5 is machine specific format. (Future occurrences not indicated)**
- 210-I COMP processed as DISPLAY (future occurrences not indicated)**
- 211-I COMP-6 processed as COMP (future occurrences not indicated)**
- 212-I COMP-1 processed as PIC S9(4) COMP (future occurrences not indicated)**

- 213-I **COMP-0 field exceeds S9(5), converted to USAGE DISPLAY**
- 214-I **COMP-0 processed as PIC S9(4) COMP (future occurrences not indicated)**
- 215-I **UNIT phrase processed as comment**
- 216-I **Literal exceeds 160 characters**
- 217-I **Procedure name same as data-name**
- 218-I **RERUN clause processed as comment**
- 219-I **No REPLACE currently in effect**
- 220-I **COMP-4 processed as PIC S9(9) COMP (future occurrences not indicated)**
- 221-I **COMP-4 field exceeds S9(10), converted to USAGE DISPLAY**
- 222-I **COMP-3 unsigned, converted to signed COMP-3**
- 223-I **BLANK WHEN ZERO clause overrides zero suppression PICTURE string**
- 224-I **MEMORY SIZE clause processed as comment**
- 225-I **MULTIPLE FILE TAPE clause processed as comment**
- 226-I **COMMON can only be used in nested program – processed as comment**

Compiler Flags

If the compiler produces any of the flags listed below, you need take no action unless you intend to compile and/or run your program in a different operating environment. If you want to change your source code, refer to *the Language Reference* for information on the various levels of COBOL.

- 005 **User name not unique**
- 009 **'.' missing**
- 010 **Word starts or is continued in wrong area of source line**
- 011 **Reserved word missing or incorrectly used**
- 026 **Literal too long**
- 042 **Must be non-zero**
- 055 **Word COPY may not be continued when in library text**
- 056 **COPY is recursive**
- 057 **Not a report group**
- 058 **Not a report name or a report group**
- 059 **Cannot GENERATE this report name**
- 060 **Not a detail group**
- 061 **Pseudo-text incorrectly specified**
- 094 **Name is not a COBOL word**
- 095 **Literal used as COPY name**
- 096 **Lowercase used (future occurrences not flagged)**

097	Both single and double quotes used (future occurrences not flagged)
098	Single quotes (apostrophe) used (future occurrences not flagged)
099	Nested COPY file
100	Sequence number out of order
101	Assignment-name is data-name
102	RELATIVE KEY clause should immediately follow ACCESS clause
103	RECORD missing
104	IDENTIFICATION missing
105	PROGRAM-ID missing
106	PROGRAM-ID has illegal format
107	Second status area
108	OPTIONAL not permitted on non-sequential file
109	Paragraphs or phrases in non-standard order or repeated
110	ENVIRONMENT missing
111	CONFIGURATION missing
112	SOURCE-COMPUTER missing
114	OBJECT-COMPUTER missing
123	I-O CONTROL missing
124	INPUT-OUTPUT missing
125	FILE-CONTROL missing
134	FILE SECTION missing
135	DATA DIVISION missing
200	Empty paragraph
202	Too many levels of OCCURS
204	REDEFINES on incorrect field
211	PICTURE clause not compatible with qualifiers
213	Item is longer than this USAGE allows
215	Value in error or illegal for PICTURE type
220	Literal type does not match data type
225	Level hierarchy wrong
230	PICTURE string has illegal precedence or illegal character
235	Record < minimum size given in FD statement
236	Record > maximum size given in FD statement
246	KEY missing or illegal
253	LABEL RECORD or DATA RECORD missing or illegal
265	Not allowed with this type

270	COLUMN specification overlapping or not left to right
280	THRU phrase not allowed with DBCS field (DBCS is Double Byte Character Set)
281	Illegal use of DBCS field (DBCS is Double Byte Character Set)
285	First LINE NUMBER clause in PAGE FOOTING group is relative
286	Not specified in CONTROL clause of RD
287	Data record specified for Report file
288	SYNC with USAGE IS INDEX
289	SYNC at group level
290	Index key not alphanumeric
291	Group FILLER
292	SIGN different from that at group level
293	REDEFINES of smaller item
294	REDEFINES of larger item
295	VALUE clause in FILE or LINKAGE SECTION, processed as comment
296	BLANK WHEN ZERO with zero suppression
297	OCCURS ... DEPENDING clause without "integer TO". "1 TO" assumed
298	OCCURS at level 01 or 77
299	FILLER omitted
300	LABEL RECORDS clause omitted
303	Operand has wrong data-type
315	Invalid conditional expression
328	Not allowed with OPTIONAL file
343	Should be declared in LINKAGE SECTION
344	Should be level 01 or 77
354	Multiple receiving fields in MOVE CORRESPONDING
361	Operation exceeds COMP subset
365	Variable size table not last in group-item
378	More than seven AFTER phrases
379	Non-DISPLAY numeric data cannot be compared with alphanumeric literal
380	Parameter count in CALL different from that in PROCEDURE DIVISION header
381	TALLYING option has ALL etc. distributed over multiple identifiers
382	Only one Procedure-name in GO TO ... DEPENDING
383	Missing ALSO
385	Order of initialization changed
386	No SECTION or paragraph at start of PROCEDURE DIVISION

387	AFTER and BEFORE options used together
388	Key is right-hand side of condition
389	EXIT or EXIT PROGRAM not in separate paragraph
390	OPEN EXTEND on non-sequential file
391	MF format ACCEPT/DISPLAY
392	More than two AFTER phrases
393	In-line PERFORM
394	No SECTION header after END DECLARATIVES
395	FROM literal
396	No suitable conditional phrase and no applicable declarative
397	No preceding SECTION
398	Offset only allowed with index-names
399	Index-name belongs to different table
400	Cannot use index data item as subscript
401	Limit exceeded - number of source statements
402	Limit exceeded - number of files > 25
404	Limit exceeded - number of pairs of REPLACING operands > 150
406	Limit exceeded - length of file/copy/library name
409	Limit exceeded - number of SELECT file names > 25
410	Limit exceeded - number of SAME RECORD AREA clauses
411	Limit exceeded - number of MULTIPLE FILE filenames
412	Limit exceeded - number of ALTERNATE RECORD KEY clauses in a file > 63
413	Limit exceeded - length of RECORD KEY > 120 characters
414	Limit exceeded - length of DATA DIVISION
415	Limit exceeded - length of data SECTION
419	Limit exceeded - BLOCK size > 32759 characters
420	Limit exceeded - RECORD length > 32759 characters
421	Limit exceeded - number of FD/SD filenames > 25
424	Limit exceeded - number of 01 & 77 in LINKAGE SECTION
425	Limit exceeded - length of FILE SECTION group item
426	Limit exceeded - length of group item exceeds 32768
427	Limit exceeded - length of data item exceeds 32767 characters
428	Limit exceeded - length of numeric edited item > 30
430	Limit exceeded - total length of VALUE literals
431	Limit exceeded - length of PICTURE string > 30 characters
432	Limit exceeded - length of PICTURE replication > 32768 characters

- 433 Limit exceeded - length of sort record > 32767 characters
- 434 Limit exceeded - length of table > 32767 characters
- 436 Limit exceeded - number of ASC/DESC KEY clauses > 12
- 437 Limit exceeded at data-name - length of ASC/DESC KEYs > 256 characters
- 438 Limit exceeded - number of INDEXED BY clauses > 12
- 440 Limit exceeded - number of paragraph labels
- 441 Limit exceeded - number of PERFORMs
- 443 Limit exceeded - number of GO TO DEPENDING ON names > 255
- 444 Limit exceeded - number of IF nesting levels > 30
- 445 Limit exceeded - number of CALL parameters > 30
- 446 Limit exceeded - number of SORT/MERGE input files > 8
- 447 Limit exceeded - number of SORT/MERGE keys > 12
- 448 Limit exceeded - number of conditions in a SEARCH ALL > 12
- 449 Limit exceeded - number of UNSTRING delimiters > 30
- 450 Limit exceeded - number of identifiers in INSPECT/
TALLYING/REPLACING > 15
- 451 Limit exceeded - length of SORT/MERGE keys > 256 characters
- 501 Feature is part of an optional module
- 502 This entire section is part of an optional module
- 503 USAGE COMP used at other than 01 level
- 504 ORGANIZATION clause in SELECT statement of sort file
- 505 VALUE OF clause in SD or CD
- 506 REDEFINES does not immediately follow data-name
- 507 Numeric literal VALUE on edited item
- 508 NEXT used in READ of sequential file
- 509 SET operation on non-index data item
- 510 ZEROS or ZEROES in BLANK WHEN clause. Treated as ZERO
- 511 FILE STATUS is not alphanumeric
- 512 data-name is qualified
- 513 Flag refers to entire section
- 514 "CHANGED" and/or "NAMED" missing
- 515 Phrases repeated
- 516 Only 1 file specified in SAME AREA clause
- 517 Jump out of inline PERFORM
- 518 > or < followed by THAN, or = followed by TO
- 519 More than 5 levels of qualification

- 520 "INVALID KEY" phrase used with sequential file
 - 521 USING literal/LENGTH OF identifier (BY CONTENT implied)
 - 522 BY CONTENT literal/LENGTH OF identifier
 - 523 DECLARATIVE SECTION without USE statement
 - 524 ALPHABET IS ASCII
 - 525 EOP or END-OF-PAGE used on file that has no LINAGE
 - 526 Phrases are not in the correct order
 - 527 REDEFINES at 01 in FILE SECTION
 - 528 Data-name in ASC/DESC key is not uniquely identified
 - 529 Clause treated as documentary
 - 530 Data item used as index file key is not alphanumeric
 - 531 Too many subscripts
 - 532 INTO data-name is group item
 - 533 START ... LESS/NOT GREATER THAN ...
 - 534 Entry treated as documentary
 - 535 ALPHABET IS EBCDIC
 - 536 CICS facility not supported
 - 537 BASIS mechanism not supported
 - 538 PICTURE string is continued
 - 539 This item is obsolete in 1985 standard
 - 540 Source field is edited
 - 541 Comparison between edited field and COMP field
 - 542 VALUE clause on group COMP
 - 543 VALUE clause on variable-length group
 - 544 VALUE clause with OCCURS or subsidiary to group OCCURS
 - 545 ALTERNATE RECORD KEY is not in SAA
 - 546 PICTURE symbol P not allowed in RELATIVE KEY
 - 547 END-IF used with NEXT SENTENCE
 - 548 Program is nested
 - 549 USING identifier must not be a redefinition
 - 550 Pseudo-text consists entirely of a separator comma or semicolon
 - 551 Second status field does not comply with VSC2 specifications
 - 552 Comment lines precede IDENTIFICATION DIVISION
 - 553 Syntax is non-conforming standard ANS85
- The language element is part of the ANSI 85 standard but above the flag-
ging level selected.

- 554 Syntax is non-conforming non-standard ANS85
The language element is not part of the ANSI 85 standard. It is an OSVS, VSC2, MF, and so on, extension.
- 555 Syntax is marked as obsolete in the ANS85 standard
The ANSI standard has defined this language element as obsolete. It will be removed from the next ANSI standard.
- 556 Multiple program source
The source file contains more than one source program. In this situation there is more than one separate, not nested, program.
- 557 Multiple GIVING files
A SORT or MERGE statement contains multiple files in the GIVING clause.
- 558 Comparison between index-name and arithmetic expression
- 559 Statement cannot be reached
- 560 Alphabet declared without ALPHABET keyword
- 561 A "NOT" phrase did not have a matching verb and was discarded.
- 562 An "ELSE" phrase did not have a matching IF and was discarded.
- 563 A "WHEN" phrase did not have a matching verb and was discarded.
- 564 A scope-delimiter did not have a matching verb and was discarded.
- 565 RECORDING MODE used with INDEXED or RELATIVE file
- 566 Floating point not supported
- 567 USAGE DISPLAY-1 missing.
- 568 Sign condition in EVALUATE statement.
- 569 Data-item does not have fixed location.
- 570 Insufficient space in area B for SO/SI insertion.
- 571 1103 Mixed literal is continued.
- 572 INITIALIZE operand does not have fixed location.
- 573 More than one REPLACING phrase.
- 574 Conditional statement not terminated by its scope-delimiter.
- 583 EXIT PROGRAM within GLOBAL declarative.

Errors Encountered During Code Generation

Error messages are output when you are compiling intermediate code (code produced by the compiler) into generated code and one of the following situations occur. After issuing any of these messages, the generation of native code stops.

Native Code Generator Messages

002 Open failure on intermediate code file

Cause: The specified intermediate code file is unavailable. This error may occur because you have specified the intermediate code filename incorrectly.

Action: Ensure that you have specified the intermediate code filename correctly. Resubmit your program to the AIX VS COBOL system to produce generated code.

003 Open failure on generated code file

Cause: The output file cannot be opened; the disk or directory may be full, or a read-only generated code file with the same name may already exist.

Action: If your disk or directory is full, you must delete any files you no longer need. If a read-only generated code file with the same name already exists, you must delete this file and resubmit your program to the AIX VS COBOL system to produce generated code.

004 Open failure on list file

Cause: The output file cannot be opened; the disk or directory may be full, or a read-only list file with the same name may already exist.

Action: If your disk or directory is full, you must delete any files you no longer need. If a read-only list file with the same name already exists, you must delete this file and resubmit your program to the AIX VS COBOL system to produce generated code.

006 Input file not intermediate code or wrong version

Cause: You may receive this error for any of the following:

- The input file you have specified is not an intermediate code file
- The intermediate code file you are using was produced under a version of the AIX VS COBOL system that is not compatible with the current version of the system which you are attempting to generate code
- Your input file is corrupt.

Action: If your input file is not an intermediate code file, you must ensure that an intermediate code file is specified. If your intermediate code file was produced under a previous version of the AIX VS COBOL system which is not compatible with your current version, or your input file is corrupt, you must resubmit your source code to the AIX VS COBOL system.

-
- 007 Write failure on generated code file**
Cause: An error occurred while writing the output file; the disk or directory may be full.
Action: If your disk or directory is full, you must delete any files you no longer need.
- 008 Write failure on list file**
Cause: An error occurred while writing the list file; the disk or directory may be full.
Action: If your disk or directory is full, you must delete any files you no longer need.
- 010 Read failure on generated code file**
Cause: The output file was successfully created but cannot be read. The AIX VS COBOL system must read this file to complete the code generation process.
Action: You must re-create the output file by restarting the production of the generated code.
- 012 Dynamic paging error**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 013 Illegal intermediate code (at *nnnnnn* in seg *mm*)**
Cause: You are attempting to create generated code from intermediate code that has been corrupted in some way.
Action: Recompile your source code to try to obtain valid intermediate code. If the same error occurs, follow your local procedures for reporting software problems.
Note: *nnnnnn* is an intermediate address; *mm* is an intermediate code segment number.
- 014 Too many IF levels**
Cause: The IF statements are nested too deeply in your COBOL program.
Action: You will have to recode your source program to ensure that you do not have a nest of more than 64 conditions.
- 018 n2 stack overflow - max size 10**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 019 Invalid condition code for branch**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.

-
- 020 cpst table overflow**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 037 Generated Code Module too large**
Cause: You have attempted to generate a program that is too large for the Native Code Generator to handle.
Action: This should not occur for this compiler. Follow your local procedures for reporting software problems.
- 038 Unprocessed transient code (Code *nnnnnnnn*)**
Cause: Either you are attempting to create generated code from a corrupt intermediate code file, or there is an internal error in the Native Code Generator.
Action: Recompile your source code, and try to create generated code again. If the same error occurs, follow your local procedures for reporting software problems.
Note: *nnnnnnnn* is an intermediate code address.
- 039 Errors detected during creation of intermediate code**
Cause: You have tried to generate native code from intermediate code that produced severe faults at compile time.
Action: Correct all of the severe faults in your source code, then recompile the code. Only when the code is compiled with no severe faults can you successfully generate native code from it.
- 040 NCG Error. Bad PROGRAM-ID or Entry name: *xxxxxxxx***
Cause: The name you have specified in the PROGRAM-ID clause or in the ENTRY USING phrase cannot be correctly handled by your system assembler.
Action: Edit your source code to ensure that the name you specify complies with the rules for a valid function name. Then recompile your program.
Note: *xxxxxxxx* is a PROGRAM-ID or Entry name.
- 041 Bad optimiser table code**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 042 Optimiser stack overflow**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.

-
- 043 Optimiser buffer overflow**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 044 Optimiser temp overflow**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 045 Optimiser hold overflow**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 046 Instruction disassembly failure**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 047 Action routine failure**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 048 E4 sequence error**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 049 Expected Symbol Not Found (Key nnnnnnnn)**
Cause: A directory (dynamic memory) search has failed to find an item that should be present.
Action: Follow your local procedures for reporting software problems.
Note: *nnnnnnnn is an intermediate address.*
- 050 Pass 2 address mismatch**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.
- 053 Object module creation failure**
Cause: Indicates an internal error while executing the Native Code Generator.
Action: Follow your local procedure for reporting software problems.

-
- 058** **NCG Error: Error opening object file in *function-name***
- Cause:** There has been an error in opening the object file when initializing the native code object module.
- Action:** Follow your local procedures for reporting software problems.
- 059** **NCG Error: Write to object file failed; out of space? Function *function-name***
- Cause:** An error occurred while writing some information to the native code object module. The error may have occurred if there is no more space in the filesystem.
- Action:** If there is no space for the file to be written, you must clear some space or enlarge the filesystem. If that does not seem to be the problem, follow your local procedures for reporting software problems.
- 060** **NCG Error: Memory allocation failed in *function-name***
- Cause:** The compiler tried to allocate more space for the native code object module, and the allocation failed.
- Action:** Follow your local procedures for reporting software problems.
- 061** **NCG Error: Memory reallocation failed in *function-name***
- Cause:** The compiler tried to reallocate space for the native code object module, and the reallocation failed.
- Action:** Follow your local procedures for reporting software problems.
- 062** **NCG Error: Error accessing Animator (.idy) file in *function-name***
- Cause:** The compilation is attempting to provide debugging information by reading and processing the .idy file. There has been an error in opening or reading that .idy file.
- Action:** Make sure that an .idy file exists. For example, if you compiled using **cob -i** and then tried to complete the compilation using **cob -gx**, then only an .int file will exist from the first **cob** processing, and there will be no .idy file for the **cob -g** to use.
- If the .idy file does exist, you may be able to get around the problem by not requesting debugging as a compilation option (that is, omit the **-g** option on the **cob** command.) To solve the problem, however, follow your local procedures for reporting software problems.
- 063** **NCG Error: CDI routine error in *function-name***
- Cause:** There has been an error detected in one of the Symbolic Debugging Interface routines. These routines are used to read the .idy file in order to create debugging information in the native object code modules.
- Action:** You may be able to get around the problem by not requesting debugging as a compilation option (that is, omit the **-g** option on the **cob** command). To solve the problem, however, follow your local procedures for reporting software problems.

-
- 064** **NCG Error: Duplicate relocation address in *function-name***
Cause: An error has occurred in the generation of the native object code module.
Action: Follow your local procedures for reporting software problems.
- 065** **NCG Error: CDI call failed — Is the ‘.idy’ file current? Function *function-name***
Cause: There has been an error detected in one of the Symbolic Debugging Interface routines. These routines are used to read the **.idy** file in order to create debugging information in the native object code modules. The **.int** code file and the **.idy** file must have been produced by the same compilation. Otherwise, they may be mismatched.
Action: Recompile the COBOL source with the **-g** option to make sure that the **.idy** and the **.int** files are produced together and are therefore a matched set.
If that does not solve the problem, you may be able to get around the problem by not requesting debugging as a compilation option (that is, omit the **-g** option on the **cob** command). To solve the problem, however, follow your local procedures for reporting software problems.
- 066** **NCG Error: Invalid object file name in *function-name***
Cause: An error has occurred while processing the name to be used for the native object code file.
Action: Follow your local procedures for reporting software problems.
- 067** **Error *number*: Index into raw wrong for format *format-code*.
Value is *number* should be *number*
First 3 elements of raw are: *data data data***
Cause: There is a compiler error in generating native object code. The number of operands provided for this native code instruction is not appropriate for this instruction format.
Action: Follow your local procedures for reporting software problems.
- 068** **Error *number*: Unknown format in object code instruction processing.
Format is *format***
Cause: This is a compiler error in generating native object code. The instruction format indicated for this object code instruction is unknown.
Action: Follow your local procedures for reporting software problems.
- 069** **NCG Error: Error opening assembler file in *function-name***
Cause: An error has occurred while processing the name to be used for the assembler source file.
Action: Follow your local procedures for reporting software problems.
- 086** **File open failure**
Cause: Indicates that the intermediate code file cannot be opened for some reason.
Action: Check that you have specified the file name correctly.

087 File I-O error

Cause: Indicates an error while reading the intermediate code file.

Action: This may be due to a fixed-disk fault, otherwise follow your local procedure for reporting software problems.

088 Internal error

Cause: Indicates an internal error while executing the Native Code Generator.

Action: Follow your local procedure for reporting software problems.

Run Time Environment Errors

Errors reported by the Run Time Environment (RTE) may occur when you are running the compiler, ANIMATOR, the Native Code Generator, or one of your own COBOL programs.

An RTE error is returned on a program that is syntactically correct and occurs when problems are encountered during the actual running of the code. You could receive such an error if you attempt to access a file in the wrong mode, or if you use a corrupt file. RTE errors are thus environment dependent, and their handling is very much dependent upon the situation in which they occur.

Types of Errors

There are two types of run-time errors:

- **Recoverable errors** are reported by the operating system so that you can trap them and take steps to recover from them if at all possible.
- **Fatal errors** are not reported and so cannot be trapped.

Recoverable Errors

Recoverable errors can be caught by your program, but the responsibility to take action when one of these errors is received is yours.

File Operation Errors: When an RTE error occurs during a file operation, one of the following occurs:

- If you did not specify a STATUS clause for the file on which the error occurred, the error is treated as though it were a fatal one. That is, the program terminates immediately with the RTE displaying its message on the console.
- If you specified a STATUS clause for the relevant file, the value 9 (which indicates that an operating system error message has been received) is placed into status key 1, and the operating system or RTE error number is placed into status key 2. You must examine status key 1 after each file operation to ensure that the operation has been carried out successfully. A value other than 0 in status key 1 indicates an error condition of some type. If an error condition is reported, the action that your program then takes is entirely your responsibility. If you do not include the STATUS clause and fail to check the condition of the STATUS byte after each file operation, and an error is reported, the program will not necessarily terminate, but its action will almost certainly not be that which you expect.

Having received a file error, you may choose to handle it in any way you like. You may want your program to display its own general error message before closing any open files (if this is possible) and then terminating. This should enable you to save any data which you have already written to the files. This data could be lost if you do not trap the error. Should you wish to recover from an error and continue the program run, you can code your program in such a way as to take certain actions should a particular error be reported. For example, if you receive a "File not found error", your program could prompt you to insert a diskette containing the required file into a specified drive. Hints on how to recover from specific errors are given later, but you will be able to follow these hints only if you have coded your program in such a way as to be able to take advantage of them. In some cases, you may need to recode your program.

The following four points suggest ways in which you may wish to code your program to handle the possible occurrence of recoverable errors:

- Use `AT END` (which checks for a value of 1 in status key 1), or `INVALID KEY` (which checks for a value of 2 in status key 1) where appropriate. You do not need to declare `STATUS` items in this case.
- Use declaratives that check that status key 1 is not equal to 0 (that is, the operation was not completed successfully) for all file operations that have no `AT END` or `INVALID KEY` clauses. You will need to declare `STATUS` items in this case.
- Declare `STATUS` items and either check for a value of 9 in status key 1, which indicates an RTE error, or check that status key 1 is not equal to 0, which shows that the file operation has not been successfully completed. You should explicitly check the `STATUS` byte after each file operation.
- Do not use the `STATUS` clause or `AT END` or `INVALID`. The RTE will terminate immediately if a file error is received. In this situation all error messages are treated as though they were fatal and the relevant RTE error message is output to the console. Your program will terminate immediately.

Fatal Errors

Fatal errors output an error message to the console; once this error message has been displayed, your program terminates immediately.

Although you will not be able to recover from such an error during the run of your program, once it has terminated you may be able to take steps to rectify the conditions that caused the error to occur. The list of Run Time Environment errors given later in this section indicates how this may be achieved for individual errors. There are two types of fatal errors: exceptions, and input-output errors.

Exceptions: These errors cover conditions such as arithmetic overflow, too many levels of `PERFORM` nesting, subscript out of range, and similar errors.

Input-Output Errors: These are recoverable errors occurring on files that have no `STATUS` items set.

Run Time Environment Error Messages

The following is a complete list of RTE error messages. Each error is followed by an explanation of the most likely cause of that error and, where possible, by hints on how you can recover. You should be able to eliminate the cause of fatal errors if you follow the recovery hints once the program has terminated. Sometimes this will mean you will have to recode your program. For recoverable errors, although recovery hints are given, your actual handling of the error will depend upon the coding of each individual program.

All of the following assume that there are FILE STATUS items set for each file. If these are not set, all errors are treated as though they were fatal. Following is a list of RTE messages marked as fatal or recoverable. Those shown as fatal cannot be trapped; those shown as recoverable can be trapped if your program includes FILE STATUS items.

001 Insufficient buffer space (recoverable)

Cause: You have tried to OPEN a file and, while you have not exceeded your system's file limit, something within your system is unable to allocate sufficient memory space for this operation to be carried out successfully.

Action: Although you can trap this error you must issue a STOP RUN as soon as it is reported.

002 File not open when access attempted (recoverable)

Cause: You have tried to access a file without OPENing it first.

Action: OPEN the file with the open mode that you require and try the file operation again. Since this error implies that there is an error in your program's logic, you may decide to terminate the run and recode your program.

003 Serial mode error (recoverable)

Cause: The program that you are trying to execute is a device, not a program.

Action: OPEN the device in the correct mode, or CLOSE any files which you have OPEN, issue a STOP RUN, and recode your program.

004 Illegal file name (recoverable)

Cause: A file name that you have supplied contains an illegal character. This could be any character that is not part of the permitted character set, or a system-dependent delimiter.

Action: Attempt the file operation again, ensuring that you use the correct file name.

005 Illegal device specification (recoverable)

Cause: Devices to which your COBOL program can WRITE are defined by the operating system. You have attempted to access a device that is not defined by your system.

Action: Attempt the operation again using a device name that your system recognizes.

- 006 Attempt to WRITE to a file opened for INPUT (recoverable)**
Cause: You have tried to WRITE to a file that is opened for INPUT only.
Action: CLOSE the file and reOPEN it with a mode such as I-O, which will allow you to WRITE to the file. Since this error implies that your program contains a mistake in its logic, you may want to CLOSE any OPEN file(s) and issue a STOP RUN. Recode your program to eliminate the logic error.
- 007 Disk space exhausted (fatal)**
Cause: There is no room available on your current disk for file operations. This error can be trapped, but once it has been reported you must issue a STOP RUN immediately to terminate your program's run.
Action: When your program has terminated, delete any files that you no longer need on your current disk, or insert a new disk in one of your disk drives and redirect your program's file operations to this disk.
- 008 Attempt to READ from a file opened for OUTPUT (recoverable)**
Cause: You have attempted to READ from a file that you opened for OUTPUT only. This violates one of the general rules of COBOL programming.
Action: CLOSE and re-OPEN the file with a mode that permits you to carry out input operations such as I-O. You should then be able to carry out the READ successfully, although this error suggests that your program contains an error in its logic, so you may want to CLOSE all of the OPEN files and issue a STOP RUN. You can then recode your program to eliminate the logic error.
- 009 No room in directory (recoverable)**
Cause: There is no room available for further file operations in the directory that you have specified, or the specified directory cannot be found by your program.
Action: Insert a new diskette and redirect your program's output to this. Alternatively, you could specify a different directory for your file operations.
- 012 Attempt to open a file which is already open (recoverable)**
Cause: You have tried to OPEN a file that is already OPEN and so cannot be OPENed again.
Action: Cancel your second attempt to OPEN the file. If the fact that the file is already OPEN is acceptable to you, continue to run your program.
- 013 File not found (recoverable)**
Cause: The operating system has been unable to find a file that you have attempted to access in your program.
Action: Insert the correct diskette (the one that contains the required file), provided that no files are currently OPEN on the present diskette. If the error is the result of a spelling mistake, then ask for the correct file. Alternatively, the file could be on a different directory.

- 014 Too many files open simultaneously (recoverable)**
Cause: You have exceeded the maximum number of files that can be OPEN at any one time. You must not violate this restraint.
Action: CLOSE some of the OPEN files that you are not currently accessing, and then try to OPEN the relevant file again.
- 015 Too many indexed files open (recoverable)**
Cause: You have tried to exceed the maximum number of ISAM files that can be OPEN at any one time. You must not violate this restraint.
Action: CLOSE some of the OPEN ISAM files that you are not currently accessing, and then try to OPEN the relevant file again. You should be able to continue to run your program. Note that ISAM files count as two files; one for data and one for the index.
- 016 Too many device files open (recoverable)**
Cause: You have tried to exceed the maximum number of device files which you can have OPEN at any one time. You must not violate this restraint.
Action: CLOSE some of the OPEN device files that you are not currently accessing, and then try to OPEN the relevant file again.
- 017 Record error: probably zero length (recoverable)**
Cause: You have probably tried to access a record that has had no value moved into it.
Action: Although this error is recoverable in the sense that it can be caught, once it has been reported you must issue a STOP RUN immediately, and then recode your program to ensure that the COBOL record length is not zero.
- 018 Read part record error: EOF before EOR or file open in wrong mode (recoverable)**
Cause: A part record has been found at the end of a file. Consequently your RTE treats the data file as a record, and not finding a full record, reports this error.
Action: Ensure that the record size you give when you READ from or WRITE to a file is consistent.
- 019 Rewrite error: open mode or access mode wrong (recoverable)**
Cause: You are attempting to do a REWRITE to a file that has not been opened with the correct access mode for this operation.
Action: CLOSE the file and reOPEN it in a mode such as I-O that allows you to do REWRITE operations on that file. Because this error implies that there is a mistake in your code logic, you may decide to recode your program, after CLOSEing any OPEN files and then issuing a STOP RUN.
- 020 Device or resource busy (recoverable)**
Cause: You have attempted to OPEN a file that is assigned to a device or resource (for example, a line printer) that is not available at this time.
Action: You can trap the error status returned by OPEN and retry the OPEN at regular intervals until it succeeds.

- 021 File is a directory (fatal)**
Cause: You have tried to WRITE to a directory instead of to a file.
Action: You will have to recode your program so that it WRITES to a file and not to a directory.
- 022 Illegal or impossible access mode for OPEN (recoverable)**
Cause: The mode in which you are attempting to OPEN a file violates the general rule of COBOL programming for that type of file. For example, you may have OPENed a line-sequential file in the I-O mode.
Action: OPEN the file with a mode that is compatible with that type of file.
- 023 Illegal or impossible access mode for CLOSE (recoverable)**
Cause: The mode in which you are attempting to CLOSE a file is not possible for that type of file.
Action: CLOSE the file with a new access mode that is compatible with the type of file, or issue a STOP RUN and recode your program.
- 024 Disk input-output error (recoverable)**
Cause: This error would be given if you do a READ after a WRITE, or if there is a verification error or a parity error.
Action: In some circumstances this error will be fatal, but if it occurs during a READ you can trap it and then issue a CLOSE on the file before issuing a STOP RUN.
- 025 Operating system data error (fatal)**
Cause: You are trying to set up terminal characteristics for a device that is not a terminal.
Action: Recode your program.
- 026 Block I-O error (fatal)**
Cause: An error occurred while you were attempting to access a disk. This could be the result of a corrupt disk.
Action: If you have a corrupt disk, try to run your program again using your backup copy of that disk.
- 027 Device not available (recoverable)**
Cause: You are attempting to access a device that is not attached to your machine or, if attached, is not online.
Action: Attach the device to your machine, and ensure that it is online.
- 028 No space on device (fatal)**
Cause: You have attempted to do a file operation such as WRITE for which there is not sufficient space available.
Action: You will have to delete some of the files or directories to make enough room for file operations.

- 029 Attempt to delete open file (recoverable)**
Cause: You have attempted to perform a DELETE FD operation on an open file.
Action: Close the file before performing the DELETE FD operation.
- 030 File system is read-only (recoverable)**
Cause: The file system that you are using is READ only, which effectively means that it is WRITE protected. You have tried to update the information found within a file in some way. For example, you may have tried to WRITE to a file or to DELETE information found within it. Since the file system you are using is READ only, you can only READ the contents of its files; you cannot alter them in any way.
Action: Abandon your attempt to alter the information within the file unless you can take another copy of that file. In this case you should be able to alter the contents of your copy, but not of the original source.
- 031 Not owner of file (recoverable)**
Cause: You are attempting to do an operation on a file, but the file's owner has not given you the necessary permission for that operation. You could, for example, be attempting to alter the access mode for a file, which only the file's owner can do.
Action: Abandon your attempted file operation or have the file's owner alter the file's permission attributes to allow you to perform the intended operation.
- 032 Too many indexed files, or no such process (recoverable)**
Cause: You have tried to OPEN an indexed file, but the number that you currently have open is the system limit. Alternatively, you could be trying to use a process ID that does not exist, or which the operating system does not recognize.
Action: CLOSE some of the indexed files that you are no longer accessing. You should then be able to OPEN the file you require.
If you are trying to use a nonexistent process ID, rewrite your code so it uses a valid process ID.
- 033 Physical I-O error (fatal)**
Cause: You have a hardware error of some kind. Perhaps you have failed to place a disk in the relevant drive, or you may have tried to WRITE to a disk but the hardware interface has failed.
Action: You will have to try to correct the fault. For example, try placing a disk in the necessary drive.
- 034 Incorrect mode or file descriptor (recoverable)**
Cause: You are either trying to WRITE to a file that is open for READ purposes only, or READ to a file that is open for WRITE purposes only.
Action: CLOSE the file and reopen using the correct access mode. Because this error implies that there is a mistake in the logic of your program, you may want to CLOSE any OPEN files, issue a STOP RUN, and then recode your program to eliminate the logic error.

- 035 Attempt to access a file with incorrect permission (recoverable)**
Cause: You are attempting to do a file operation that you do not have sufficient permission to achieve. For example, you could be trying to WRITE data to a file that has been set up with READ permission only.
Action: If you are the owner of the file, you can alter the attributes of the file so you have the permission needed to effect the particular file operation. If you are not the owner of the file, you cannot carry out that operation successfully unless you copy the file and make the changes to the copy only. You cannot alter the original source file.
- 036 File already exists (recoverable)**
Cause: You are attempting an inappropriate operation on an already existing file.
Action: This error implies a fault in your program's logic. You may want to recode your program to eliminate this.
- 037 File access denied (fatal)**
Cause: Your attempt to access a file has been denied by the operating system. You may have tried to WRITE to a WRITE protected file, or you could have attempted to READ from an OUTPUT device.
Action: Alter the access permission on the relevant file. Access can be READ only if you just want to read the contents of the file without making any changes, or it can be READ and WRITE, in which case you can alter its contents.
- 038 Disk not compatible (fatal)**
Cause: You have tried to load a disk that is incompatible with the current version of your operating system. This could be because it was created under a completely different operating system. You would also receive this error if you tried to load a disk with the same name as a disk that was already loaded.
Action: If the error is a result of duplicate disk names, you can rename one of the disks. Then you can load both disks together.
- 039 File not compatible (fatal)**
Cause: You are trying to load a file that is not compatible with the structure of AIX VS COBOL files. This could be because the file was created under a different operating system or under a different COBOL product.
Action: Create a new copy of the file that has the correct structure.
- 040 Language initialization not set up correctly (fatal)**
Cause: You have attempted to use the additional language variants at run time, but the environment or side file that is required to set up the language has not been set up correctly, does not exist, or is invalid.
Action: Set up the required environment or side file.

- 041 Corrupt indexed file (recoverable)**
- Cause:** Your Run Time Environment does not recognize the control information for an indexed file. Since the index has been corrupted in some way, the data in the file is no longer accessible by your system. This error is recoverable in the sense that it can be trapped, but should you receive it, there is little you can do except to CLOSE any OPEN files and STOP your program's RUN.
- Action:** Rerun your program using the backup copy of that file. If you have added a great deal of information to the file since you last made a backup, you may want to rebuild the file using a utility that is capable of reading the data (if this has not been corrupted) and build a new index for it (**bcheck**).
- 042 Attempt to write on broken pipe (recoverable)**
- Cause:** Your program has created a process as a result of a dd_xxx logical file name mapping assignment (for example, the process may be a line printer spooler). The process was not created properly, or has died prematurely. This error occurs when your program attempts to write to the process.
- Action:** You can trap the error status returned by the write operation, then open the file again.
- 043 File information missing for indexed file (fatal)**
- Cause:** You normally receive this message if the system crashed on the program's previous run while the file was OPEN. Information was probably added to the end of the file, but the directory information was not updated and so that data cannot be accessed by your system. You can also receive this message if you copied the ISAM file from one disk to another but only copied either the data part of the file or the index.
- Action:** If the error is the result of a crash, whether you can access the necessary data or not is entirely dependent on the state of the system at the time of the crash. However, if it is the result of a faulty copy, you should be able to restore the missing part of the file from the **.dat** or **.idx** file.
- 044 Attempt to OPEN an NLS file in a non-NLS program**
- Cause:** You have attempted to open a file created as an NLS file from a program that was not compiled with the NLS option.
- Action:** You must either not access NLS files from this program, or compile it with the **nls** option.
- 045 Attempt to OPEN a file using incompatible NLS attribute**
- Cause:** All operations on NLS files must be done using the same language selection as the one used when the NLS file was created. You are attempting to OPEN a file using a different language setting than the one used to create it.
- Action:** You must create and use NLS files using the same language selection. Change the language selection to the one used to create the file and re-run your program.

- 047 Indexed structure overflow (fatal)**
- Cause:** There is some fault in the structure of your ISAM file. You have probably tried to put another entry in the index when there is no room for it. This error could also be given if you have tried to access an old ISAM file, perhaps created using another COBOL version.
- Action:** If there is no room in your index for further entries, reorganize your file. If you have attempted to access an old ISAM file, you can run the **bcheck** utility to check the consistency of this ISAM file and to construct a new ISAM file if the old one was found to be corrupt.
- 065 File locked (recoverable)**
- Cause:** You have tried to OPEN a file that has already been locked or opened for output by another user. Alternatively you have tried to OPEN a file for output that another user already has open.
- Action:** Your program can inform the system operator that it is unable to access this file and should wait until the other user finishes using the file and closes it.
- 066 Attempt to add duplicate record key to indexed file (fatal)**
- Cause:** You have tried to add a duplicate key for a key that you have not defined as being able to have duplicates.
- Action:** This error implies that there is a fault in your program logic which you need to correct.
- 067 Indexed file not open (recoverable)**
- Cause:** You are attempting to access an indexed file that you have not OPENed.
- Action:** OPEN the file in the relevant access mode, and then retry the unsuccessful file operation.
- 068 Record locked (recoverable)**
- Cause:** You have tried to access a record that is currently locked by another user.
- Action:** Your program should display a message to the system operator that the record is currently locked. Wait until the other user has released the lock on that record. Then you can access the relevant record. Do not continually retry to gain access to the record without operator intervention. This could result in your application hanging up.
- 069 Illegal argument to ISAM module (fatal)**
- Cause:** This is the result of an internal system error.
- Action:** Follow your local procedures for reporting software problems.
- 070 Too many indexed files open (recoverable)**
- Cause:** You are attempting to OPEN an indexed file but you have already exhausted the system limit that specifies how many of these files can be OPENed at any one time.
- Action:** CLOSE some of the open indexed files that you are not currently accessing. Then you can OPEN the indexed file that you require.

- 071 Bad indexed file format (fatal)**
Cause: This error could be given if you are using a file that has been corrupted. Otherwise, it is the result of an internal system error.
Action: If the file you are using is corrupt, rerun your program using your backup copy of the file. If a corrupt file is not the cause of the error, follow your local procedures for reporting software problems.
- 072 End of indexed file (fatal)**
Cause: This is the result of an internal system error.
Action: Follow your local procedures for reporting software problems.
- 073 No record found in indexed file (fatal)**
Cause: This is the result of an internal system error.
Action: Follow your local procedures for reporting software problems.
- 074 No current record in indexed file (fatal)**
Cause: This is the result of an internal system error.
Action: Follow your local procedures for reporting software problems.
- 075 Indexed data file name too long (fatal)**
Cause: The maximum number of characters that the AIX VS COBOL system allows a file name to have is 14. However, when using ISAM the extension `.idx` is added to the end of the user-defined file name. Therefore, if your file name exceeds 10 characters in length, you will receive this error message.
Action: Rename the file with a file name that is less than 10 characters in length.
- 076 Cannot create lock file in /isam directory (fatal)**
Cause: Your system is unable to create a lock file in the ISAM directory. It could be that in a previous run your program terminated abnormally, leaving some files locked. If you try to run a program following an abnormal termination, you will receive this error.
Action: Manually remove from the ISAM directory all files still locked.
- 077 Internal ISAM module error (fatal)**
Cause: This is the result of an internal system error.
Action: Follow your local procedures for reporting software problems.
- 078 Illegal key description in indexed file (fatal)**
Cause: This is the result of an internal system error.
Action: Follow your local procedures for reporting software problems.
- 081 Key already exists in indexed file (fatal)**
Cause: This is the result of an internal system error.
Action: Follow your local procedures for reporting software problems.

- 100 Invalid file operation (fatal)**
- Cause:** You have attempted a file operation that violates a general rule of COBOL in some way. You may have attempted a REWRITE on a sequential file opened I-O, or on a relative file with access mode sequential also opened I-O, without preceding it with a successful READ NEXT.
- Action:** Recode your program to ensure that the REWRITE statement is preceded by a READ NEXT.
- 101 Illegal operation on indexed file (fatal)**
- Cause:** This is the result of an internal system error.
- Action:** Follow your local procedures for reporting software problems.
- 102 Sequential file with non-integral number of records (fatal)**
- Cause:** This error could be given if you have specified an incorrect record length for a sequential file, if the sequential file you are attempting to access is corrupt in some way, or if the file that you have specified is not a sequential file.
- Action:** Recode your program to specify the correct type of file, or if the error is a result of a corrupt file, attempt to run the program again using a backup copy of that file.
- 104 Null file name used in file operation (fatal)**
- Cause:** You have specified a variable name for a file name instead of a literal, so on attempting to OPEN that file, only spaces are found.
- Action:** Recode your program specifying the correct file name.
- 105 Memory allocation error (fatal)**
- Cause:** The Run Time Environment is unable to allocate sufficient memory space to carry out the attempted operation successfully. This error implies that there is no memory space left on your system.
- Action:** You need to increase the size of the data area allowed for your process. In the **sh** shell, give the “ulimit -b *newsiz*” command. In the **csh** shell, give the “limit datasize *newsiz*” command. See the AIX Operating System commands documentation for the **sh** and **csh** commands for a description of the **ulimit** and **limit** features, and the units used for the *newsiz* values. If this does not solve the memory problem, you may need to acquire more memory on your system.
- 106 Dictionary error (fatal)**
- Cause:** This could be the result of a READ or WRITE error to file or disk, but it is more likely to be the result of an internal system error.
- Action:** Follow your local procedures for reporting software problems.
- 107 Operation not implemented in this Run Time Environment (fatal)**
- Cause:** You are attempting to do a file operation that the RTE does not support.
- Action:** Recode your program so it does not attempt such operations, or supply a user-written program to support this facility.

- 108 Failure to initialize Data Division (fatal)**
Cause: The RTE cannot load your program correctly because the data needed to initialize the Data Division correctly has become corrupted.
Action: Recompile your program to obtain a good piece of intermediate code.
- 109 Invalid checksum in Run Time Environment (recoverable)**
Cause: The internal information within the RTE has been altered. This error may be caused by a corrupted RTE, or you may have illegally attempted to change the internal RTE information.
Action: Follow your local procedures for reporting software problems.
- 116 Cannot allocate memory (CISAM) (fatal)**
Cause: A part of your RTE is unable to allocate sufficient memory to enable the execution of your code.
Action: Reduce memory usage by cancelling programs not in use, then attempt again the operation that caused this message.
- 117 Bad collating sequence (CISAM) (recoverable)**
Cause: This is an internal system error.
Action: Follow your local procedures for reporting software problems.
- 118 Symbol not found (fatal)**
Cause: You are unable to load your object file. You receive this message if you attempt to call a program that has not been specified in the COBPATH environment variable.
Action: Check that your COBPATH has been set up correctly. If not, correct your COBPATH to include the program being called.
- 119 Symbol redefined (fatal)**
Cause: You are unable to load your object file because it has an entry point with the same name as a module already loaded.
Action: Once your program has ended, recode it to remove the naming duplication, then recompile the code.
- 120 Symbol string table of zero size (fatal)**
Cause: You probably have a malformed object file.
Action: Once your program has ended, correct the object file. If this does not work, follow your local procedures for reporting software problems.
- 121 Symbol is not in text section (fatal)**
Cause: You have attempted to CALL a subprogram that is not executable. Alternatively, you have used the same name for a called program as a previously defined data item.
Action: Check that the subprogram being called is executable. Correct the subprogram's name in the CALLing program or recode it to remove the naming duplication, then recompile the code.

- 122 Coblongjmp() called below level of cobsetjmp (fatal)**
Cause: You have returned control to a higher level in the CALL/PERFORM hierarchy than the level at which **cobsetjmp** was called. **coblongjmp** must only be called from the same or from a lower level in the CALL/PERFORM hierarchy, as **cobsetjmp** was.
Action: Check and correct the logic of your program.
- 123 Unknown relocation type (fatal)**
Cause: You are using incompatible versions of the object file and the COBOL Run Time Environment.
Action: Once the program has ended, recompile your object file with the AIX VS COBOL Run Time Environment/6000.
- 129 Attempt to access record zero of relative file (recoverable)**
Cause: The value specified in the RELATIVE KEY data item contains the value zero.
Action: Ensure that the value in the RELATIVE KEY data item is greater than zero.
- 135 File must not exist (recoverable)**
Cause: You have attempted to OPEN for OUTPUT a file that already exists.
Action: Either open the existing file for I-O, or supply a unique file name for the file to be opened for OUTPUT. This error implies that an error exists in your program logic. CLOSE any OPEN files, issue a STOP RUN, and edit your code to correct the logic error.
- 138 File closed with lock -- cannot be opened (recoverable)**
Cause: You are attempting to OPEN a file that you previously CLOSED with a lock. Because such an operation violates COBOL programming rules, you are given this error message.
Action: You cannot OPEN the relevant file. This message indicates an error in the logic of your program. CLOSE any OPEN files, issue a STOP RUN, and correct the logic error in your code.
- 139 Record length or key data inconsistency (recoverable)**
Cause: There is a discrepancy between the length of a record, or the keys that you specified in your current program and their definition in the program in which they were first OPENed.
Action: This message indicates an error in your program, so you need to correct your code and recompile.
- 141 File already open -- cannot be opened (recoverable)**
Cause: You have tried to OPEN a file that is already OPEN and so cannot be OPENed again.
Action: Cancel your second attempt to open the file and continue to run your program if it is acceptable to you that the file is OPEN. This message implies that an error exists in your program logic. CLOSE any OPEN files, issue a STOP RUN, and edit your code to correct the logic error.

- 142 File not open -- cannot be closed (recoverable)**
- Cause:** You have tried to CLOSE a file that is not OPEN, which is impossible.
- Action:** Abandon the attempt to CLOSE the file. Do not continue to run your program. This message implies that an error exists in your program logic. CLOSE any OPEN files, issue a STOP RUN, and edit your code to correct the logic error.
- 143 Rewrite/delete in sequential mode not preceded by successful read (recoverable)**
- Cause:** You have violated one of the COBOL programming rules. You failed to do a successful READ on a sequential file before attempting a REWRITE or DELETE on some of the information contained within that file.
- Action:** If the previous READ was successful, then perform a READ on the relevant file before you retry the unsuccessful REWRITE or DELETE operation. If the previous READ was also unsuccessful, CLOSE the file, issue a STOP RUN, and edit your code to correct the logic error.
- 146 No current record defined for sequential read (recoverable)**
- Cause:** The file position indicator in your file is undefined due to a failed READ/START or INVALID KEY condition. You have tried to read another record in the file but because the current record is undefined the system cannot find the start of the record for which you have asked.
- Action:** Attempt a START operation, and continue to do so until the file position indicator is updated successfully.
- 147 Wrong open mode or access mode for read/start (recoverable)**
- Cause:** You violated a COBOL general rule for programming when you tried to carry out a READ or START operation on a file that has not been OPENed for INPUT or I-O, or is not OPEN at all.
- Action:** OPEN the file for I-O or for INPUT. You should then be able to continue running your program. This error implies that an error exists in your program logic. CLOSE any OPEN files, issue a STOP RUN, and edit your code to correct the logic error.
- 148 Wrong open mode or access mode for write (recoverable)**
- Cause:** You tried to WRITE to a file in sequential access mode that you have not OPENed for OUTPUT or EXTEND, or you tried to WRITE to a file in random or dynamic access mode that has not been OPENed for INPUT or I-O.
- Action:** CLOSE the file and reOPEN it with the correct open mode for the file type. This error implies that an error exists in your program logic. CLOSE any OPEN files, issue a STOP RUN, and edit your code to correct the logic error.

- 149 Wrong open mode or access mode for rewrite/delete (recoverable)**
Cause: You violated a COBOL general rule for syntax, when you tried to do a REWRITE or a DELETE on a file that you had not OPENed for I-O.
Action: CLOSE the file and reOPEN it for I-O. This error message implies that an error exists in your program logic. CLOSE any OPEN files, issue a STOP RUN, and edit your code to correct the logic error.
- 150 Program abandoned at user request (fatal)**
Cause: You have interrupted the program by means of a keyboard interrupt. Any open files are closed by the RTE.
- 151 Random read on sequential file (recoverable)**
Cause: You violated a COBOL syntax rule by trying to do a random READ on a file that has sequential organization.
Action: READ the file with the correct access mode. This error implies that an error exists in your program logic. CLOSE any OPEN files, issue a STOP RUN, and edit your code to correct the logic error.
- 152 REWRITE on file not opened for I-O (recoverable)**
Cause: You violated a COBOL syntax rule by attempting a REWRITE on a file that has been OPENed for I-O.
Action: CLOSE the relevant file and reOPEN it for I-O operations. You should then be able to carry out the REWRITE operation successfully. However, this implies that an error message exists in your program logic. CLOSE any OPEN files, issue a STOP RUN, and edit your code to correct the logic error.
- 153 Subscript out of range (fatal)**
Cause: A subscript that you have used in your program is out of the defined range; that is, it is less than one or it is greater than the number of occurrences of the item.
Action: Recode your program.
- 154 PERFORM nested too deeply (fatal)**
Cause: This error usually results if you have used GO TO to jump out of the range of a PERFORM rather than to jump to an EXIT statement at the end of its range.
Action: When your program has ended, recode your program to ensure that the GO TO in question jumps to an EXIT statement at the end of the PERFORM's range.
- 155 Illegal command line (fatal)**
Cause: The general command line interpreter cannot be found. It must be present if your program is to be run successfully.
Action: Ensure that the interpreter is present to enable your system to pick up the commands correctly.

- 156 Too many parentheses in compute statement (fatal)**
Cause: You have coded a compute statement that is too complex for your system to handle successfully.
Action: Recode your program. You are advised to break the relevant compute statement into a number of simpler statements.
- 157 Not enough program memory: object file too large to load (recoverable)**
Cause: Your program is too large for the available memory space.
Action: If you have specified the ON OVERFLOW clause in the relevant CALL statement, the error is recoverable. The associated imperative statement will be executed before the next instruction.
- 158 Attempt to REWRITE to a line-sequential file (recoverable)**
Cause: You have used the REWRITE statement in conjunction with a file whose organization is line-sequential. The REWRITE statement cannot be used with line-sequential files.
Action: Close the offending file before issuing STOP RUN to ensure that you do not lose any data from it. Recode your program to make the organization of the file to which you want to do a REWRITE either sequential, indexed sequential, or relative.
- 159 Malformed line-sequential file (recoverable)**
Cause: A line-sequential file that you are trying to access is corrupt in some way.
Action: Rerun your program using the backup copy of that file.
- 160 Overlay loading error (recoverable)**
Cause: An error occurred while the system was loading the intermediate code for an independent segment. The segment is either missing or corrupted in some way.
Action: If the segment is missing, locate it. If you cannot find it, or if it is present and corrupt, recompile your program.
- 161 Illegal intermediate code (fatal)**
Cause: The code that is currently being processed is not valid. You are probably trying to execute a corrupted file or one that has not been compiled successfully.
Action: You will have to recompile your source program to obtain uncorrupted code.
- 162 Arithmetic overflow or underflow (fatal)**
Cause: You have attempted to divide a data item by zero.
Action: Recode your source program to avoid this illegal operation.

- 163 Illegal character in numeric field (fatal)**
- Cause:** The value that you enter into a numeric field is checked by default to ensure that it is numeric. If any of the characters are found to be nonnumeric, you get this message. The message is also given if you have entered numerics that are not initialized into numeric fields, because these are automatically space-filled and are thus classified as nonnumeric items.
- Action:** If you unset the numeric field check switch (-F) on the run command line, the RTE will not check that all values in a numeric field are numeric, and you should be able to run your program successfully. Alternatively, you can make sure that you initialize numeric items with numeric values. This should enable your program to run successfully regardless of the numeric field check-switch setting.
- 164 Run Time subprogram not found (fatal)**
- Cause:** You have attempted to call a subroutine whose entry address has not been set up in your RTE.
- Action:** Check to see that you used a valid call number in the unsuccessful subroutine call. If not, correct your code to contain a call number that your system recognizes.
- 165 Version number incompatibility (fatal)**
- Cause:** You are using intermediate code which has been produced on a version of the AIX VS COBOL system that is incompatible with the Run Time Environment you are currently using. The RTE will not, therefore, be able to execute correctly any generated code you are producing or have already produced from this intermediate code. Alternatively, you may have attempted to execute a file which is not AIX VS COBOL intermediate or generated code.
- Action:** Recompile your source programs using the current version of the AIX VS COBOL Compiler.
- 166 Recursive COBOL CALL is illegal (fatal)**
- Cause:** You have tried to CALL a COBOL module that is already active.
- Action:** You will need to recode your program.
- 167 Too many USING items (fatal)**
- Cause:** The list of items that you have supplied in a CALL ... USING statement is longer than the RTE can handle.
- Action:** Once your program has ended, recode it with group items rather than elementary items before rerunning it.
- 168 Stack overflow (fatal)**
- Cause:** You have nested a PERFORM statement or a series of CALL statements too deeply.
- Action:** Edit your program to reduce the number of levels within a nested PERFORM or CALL statement, then recompile.

- 169 Illegal configuration information (fatal)**
Cause: You have attempted an operation for which your machine is not configured; the most likely cause is that ADIS is not configured correctly.
Action: Check that ADIS is configured correctly. See Chapter 10, "Configuring Your AIX VS COBOL System."
- 170 System program not found (fatal)**
Cause: A system program (for example, ADIS) is not found.
Action: Ensure that all the system programs are available and COBDIR is set to the correct path. Copy those that are not currently present using your backup system disk.
- 171 Japanese operations illegal with this RTE (fatal)**
Cause: You are attempting to do Japanese operations with a non-Japanese RTE, or you have used a Japanese compiler to compile your program and now are trying to run your code using a non-Japanese RTE.
Action: Recompile your program using the Japanese version of the AIX VS COBOL Compiler. Also check your environment to make sure that you are using the Japanese RTE (that is, the DBCS-variety of the compiler and environment).
- 172 Recursive OS/VIS PERFORM is illegal (fatal)**
Cause: You have tried full recursion of a PERFORM statement in a program that was compiled with the **perform-type = osvs** option set. That is, you attempted to end two PERFORMs with the same return address.
Action: Recompile your program with the **perform-type = osvs** option off, or recode your program so each PERFORM has its own unique return address before you recompile it with the **perform-type = osvs** option on.
- 173 Called program file not found in drive/directory (fatal)**
Cause: You have tried to call a file that is not present on your current directory, drive, or in a directory pointed to by the COBPATH environment variable.
Action: Once your program ends, copy the relevant file into your current drive or directory. Then you will have to set the COBPATH environment variable to search the directory or drive on which the file is present when your program calls it.
- 174 I-O Error: .idy file (fatal)**
Cause: The system is unable to write to the .idy file.
Action: Correct the permissions on the file or directory to allow write access, then recompile your program.

- 175** **Attempt to run intermediate code program which had severe errors (fatal)**
Cause: You are attempting to run a program that produced severe errors during compilation with the run-time switch E turned off.
Action: Edit your source code to correct all the severe errors, recompile, then run the intermediate code that is produced. Alternatively, you could attempt to run the program with the E run-time switch set, though this may not give the desired results. Refer to Chapter 7, "Running an AIX VS COBOL Program."
- 176** **Illegal intersegment reference (fatal)**
Cause: Your code contains a segment reference for the Forward Reference Table. This is illegal.
Action: Follow your local procedures for reporting software problems.
- 177** **Attempt to cancel active program (fatal)**
Cause: You have tried to remove a currently executing program, or its next levels, from memory.
Action: Once your program has ended, recode your program to ensure that you do not attempt to cancel a program or its next levels while it is still being executed.
- 178** **Error during save (fatal)**
Cause: You cannot save the information that your program has generated. One of the most common reasons for this is the attempt to build a module that is too large for the available memory space.
Action: If the error is caused by a lack of space, you can either delete some of the files that you no longer need on your current disk, or insert a new diskette to take the output from your program. Rerun your program, and save the information given by it.
- 179** **Error during chain (program not found) (fatal)**
Cause: You have tried to chain to another program that your system is unable to find.
Action: Ensure that your spelling is correct, and if not, correct it before rerunning your program. If you have used the correct spelling, then check that the program is available on your disk. If necessary, copy the backup copy of your program onto your disk. When your program is being animated, ANIMATOR will report this error and will allow you to continue to run the program. If there is not sufficient space available to allow you to do this, then you will have to set the COBPATH environment variable to search the directory or drive on which the file is present when your program calls it.
- 180** **End-of-file marker error (fatal)**
Cause: A file marker used to indicate that the end-of-file has been reached is missing from one of your files.
Action: Recompile your code, or use a debugger to place the end-of-file marker at the end of the file.

- 181 Invalid parameter error (fatal)**
Cause: You used a parameter that is not recognized by your system.
Action: Correct your code to contain a parameter that is known by your system.
- 182 Console Input or Console Output open with incorrect mode (fatal)**
Cause: You are either trying to READ input from the display screen or WRITE to the keyboard.
Action: Recode your program.
- 183 Attempt to open line-sequential file for I-O (fatal)**
Cause: You tried to open a line-sequential file in the input-output open mode, but this mode is not supported for files with this organization.
Action: When your program has ended, recode your program to ensure that the file with organization line-sequential is opened for input, output, or extend. Rerun your code.
- 184 ACCEPT/DISPLAY I-O error (fatal)**
Cause: You have either tried to READ input from the display screen or WRITE to the keyboard, or the ADIS module has not been able to open your terminal's channels for I-O.
Action: Recode your program to correct the error.
- 185 Fatal File Malformation (fatal)**
Cause: The RTE detects that a file is corrupt when opening the file. For example, only part of a record is there.
Action: Use a backup version of that file if one is available, or open the original with the correct record size.
- 186 Attempt to open stdin, stdout or stderr with incorrect mode (fatal)**
Cause: You have tried to open `stdin` for output, or `stdout` or `stderr` for input.
Action: Open these files with the correct mode.
- 187 Run Time Environment not found on \$COBDIR path (fatal)**
Cause: The RTE cannot be found on the path you have set up in the COBDIR environment variable.
Action: Correct your \$COBDIR environment variable to include the path that your RTE is on.
- 188 File name too large (fatal)**
Cause: A file name that you have used has more characters than the maximum number allowed by the AIX VS COBOL system (fourteen).
Action: Once your program has ended, recode your program, renaming the offending file with a file name of acceptable length.

- 189 Intermediate code load error (fatal)**
- Cause:** You are unable to load your intermediate code. You could receive this error if you attempt to load a piece of code that has not been successfully compiled, or if you try to load a piece of intermediate code that has been corrupted in some way.
- Action:** Obtain uncorrupted intermediate code by recompiling your source code.
- 190 Too many arguments to CALL (fatal)**
- Cause:** A CALL within your program cannot be successfully executed because of the number of arguments used with it.
- Action:** When your program has ended, recode it using group items rather than elementary ones.
- 191 Terminal type not defined (fatal)**
- Cause:** Your terminal is not defined in the terminfo database.
- The operating system is unable to drive your terminal because it has no environment specification for it.
- Action:** Set up the necessary environment for your terminal. Until this is done you cannot run your programs.
- 192 Required terminal capability description missing (fatal)**
- Cause:** A compulsory entry (for example, cursor movement or clear display screen) is missing from your **termdesc** file.
- Action:** Add the missing entry to your **termdesc** file.
- 193 Error in variable length count (fatal)**
- Cause:** The piece of intermediate code that is currently being processed is not a valid operation. You are probably trying to execute a corrupt file or one that has not been compiled.
- Action:** Recompile your source program.
- 194 File size too large (fatal)**
- Cause:** A file that your program is accessing is too large for successful execution to continue.
- Action:** When your program has ended, recode your program spreading the data over more than one file to ensure that no file becomes too large for your operating system to handle. You can also try to increase your **ulimit**.
- 195 DELETE/REWRITE not preceded by a read (fatal)**
- Cause:** Before a DELETE or a REWRITE statement can be successfully executed in sequential access mode, the last input-output statement executed for the associated file must have been a successful READ. In your code no READ statement precedes your attempted DELETE or REWRITE statements.
- Action:** When your program has ended, recode your program, making sure the last input-output statement to be executed before the DELETE or REWRITE is a READ statement.

- 196 Record number too large in relative or indexed file (fatal)**
Cause: The relative record key has exceeded the system limit; that is, the file is too large for the system to handle.
Action: The record key that you have specified is too large for the system to deal with, or the pointer to the record has been corrupted in some way so that it is either too large, or it is not a multiple of the record length.
- 197 Screen-handling system initialization error (fatal)**
Cause: The display screen-handling interface has not been correctly initialized because your terminal does not have the required capabilities, or your **terminfo** file is corrupted, or memory has been incorrectly allocated.
Action: Check that the **terminfo** file contains the correct entry for your terminal.
- 198 Load failure (fatal)**
Cause: You have not been able to load the file that you requested because it is corrupt.
Action: If the file failed to load because it is corrupt, then rerun your program, loading your backup copy of the file.
- 199 Operating system error code lies outside defined range (fatal)**
Cause: A system call has returned an unexpected error number that is not documented.
Action: Follow your local procedures for reporting software problems.
- 200 Run Time Environment internal logic error (fatal)**
Cause: You can receive this message if the amount of memory available on your machine is so low that not even the RTE can be loaded correctly.
Action: Free some memory. Then you should be able to run your program. If the RTE has halted as a result of an internal error from which you cannot recover, follow your local procedure for reporting software problems.
- 201 I-O error in paging system (fatal)**
Cause: There is no room available in your current directory or on the directory you are using for the paging file.
Action: When your program ends, increase the size of your page space minidisk.
- 202 Exported functionality error (fatal)**
Cause: You have either caused an internal RTE error by invalid use of an exported function, or the code produced by a preprocessor within the compiler contains errors.
Action: Ensure that all your external assembler applications call and use RTE functions correctly before you attempt to run your program again. If you are using a preprocessor as part of the compiler, use the software as a stand-alone preprocessor to isolate the problem areas.

- 203 CALL parameter not supplied (fatal)**
Cause: You have not supplied your currently executing program with all of the parameters mentioned in the LINKAGE SECTION of your main program.
Action: Recode your program to ensure that it contains all of the necessary parameters, or check that it is a valid caller.
- 206 Reading unwritten data from memory (fatal)**
Cause: You are attempting to read data that has not been written.
- 207 Machine does not exist (recoverable)**
Cause: You have tried to access a machine that is not connected to your network, or if the machine is part of your network, it is not online.
Action: Ensure that the machine is connected to the network and online.
- 208 Error in multi-user system (recoverable)**
Cause: This is normally caused by an unexpected error occurring within the network or file-sharing facilities. A corrupted network message will also return this error.
Action: You may be able to recover from this error by executing a COMMIT statement.
- 209 Network communication error (recoverable)**
Cause: This message is normally given if an incorrect checksum has been received in a communications packet.
Action: Your program should continue to execute after you have received this error, but the effect of the error is undefined.
- 210 File is closed with lock (fatal)**
Cause: You have tried to open a file that you previously closed with a lock.
Action: Recode your program to avoid opening a file previously closed with a lock.
- 211 Program linked with wrong library (fatal)**
Cause: You have tried to link a program that is incompatible with the current version of your RTE, your object file, or your COBOL run-time library. For example, your RTE will not run a program linked using a different object file format or COBOL run-time library.
Action: If your object file is incompatible with the current version of either your COBOL run-time library or your RTE, recompile your object file and then relink with the current version of your COBOL run-time library. Otherwise, relink your program.
- 212 Malformed assembler subroutine file (fatal)**
Cause: You are attempting to access an assembler routine that is not in the specific format for such a file.
Action: Examine the assembler routine and alter the structure of your routine if necessary. Ensure that it is linked correctly, that it has the correct structure, and that you used the right assembler.

- 213 Too many locks (recoverable)**
Cause: You have either tried to exceed the maximum number of record locks allowed per file, or you have exhausted an operating system or network resource (for example, dynamic memory).
Action: Execute a COMMIT or an UNLOCK operation on the relevant file. You should then be able to continue running your program. Do not try to retain a record lock for longer than is necessary. This should prevent the occurrence of this error.
- 214 GO TO has not been altered (fatal)**
Cause: You violated a COBOL programming rule.
Action: CLOSE any files that may be OPEN, and issue a STOP RUN. Edit your program to avoid such illegal operations.
- 215 Cannot animate a program running COMMUNICATIONS (fatal)**
Cause: The communications module is not supported at run time.
Action: In order to run your program on AIX VS COBOL, you must recode your program so that it does not use communications.
- 216 Cannot initialize named communications device (fatal)**
Cause: The communications module is not supported at run time.
Action: In order to run your program on AIX VS COBOL, you must recode your program so that it does not use communications.
- 217 Incompatible host for generated code file (fatal)**
Cause: The .gnt file is not valid for the host processor.
Action: Recompile your program.
- 218 Malformed MULTIPLE REEL/UNIT file (fatal)**
Cause: Your file header is not correctly formatted, or you are not using a MULTIPLE REEL/UNIT file.
Action: Attempt to run your program again using a backup copy of the relevant file.
- 219 Operating system shared file limit exceeded (recoverable)**
Cause: You have tried to exceed your operating system limit on the number of shared files that you can have OPEN simultaneously.
Action: CLOSE some of the open shared files you are no longer accessing and retry the file operation.
- 220 Attempt to execute more than one SORT or MERGE simultaneously (fatal)**
Cause: You have coded your program in such a way that it is attempting to execute more than one SORT or MERGE at the same time. For example, you may have coded a SORT statement in the input or output procedure of another SORT statement, an operation that is prohibited under ANSI COBOL rules.
Action: Recode your program to ensure that it does not execute more than one SORT or MERGE at any one time.

221, 222, 223 SORT/MERGE error: see status key (fatal)

Cause: You receive one of these three messages if you attempt to do a SORT/MERGE operation that is unsuccessful. These errors can result from a number of causes. For example, you may have too many files OPEN when you attempt a SORT/MERGE, or the file that you are trying to access may be locked.

Action: The action you take is situation-dependent.

254 Keyboard interrupt to ANIMATOR during ACCEPT (fatal)

Cause: While using ANIMATOR, you have ended your program with a keyboard interrupt.

cob Command Errors

017 COBOL compiler argument exceeds 128 byte limit

You must shorten the long argument to fall within this limit.

020 I see no use for ...

cob is unable to process the files you have specified because they have already been processed as far as possible for the **cob** command, or the specified options are not recognized.

022 I see no work

You have specified inappropriate parameters which do not match the options you have used.

034 Argument expected: *optionname*

The specified option requires an argument.

052 cobol version number invalid — Call Technical Support

The contents of the cobver file are incorrect.

053 Entry defined: *entryname-1* conflicting “main” found in *entrypoint-2*

You have defined the symbol “main” in a module other than the specified entry-point module.

055 Entry: *entryname* does not have “main” defined

The specified entry-point module (which is a non-COBOL module) does not define the symbol “main”. The entry-point “main” must exist for any non-COBOL modules that will be the main program for your code.

056 Entry: *entryname* not found

You need to specify an entry-point in at least one module.

071 Invalid entry-point name: *entryname*

The specified entry-point symbol is not a valid symbol name. Valid symbol names may consist of any of the characters a through z, A through Z, 0 through 9, the underscore (), or the period (.). A name cannot begin with a digit.

Appendix A. Environment Variables

Introduction	A-3
COBATTR	A-3
COBCPY	A-4
COBCTRLCHAR	A-4
COBDIR	A-4
COBHELP	A-5
COBIDY	A-5
COBLPFORM	A-5
COBOPT	A-6
COBPATH	A-6
COBPRINTER	A-7
COBSW	A-7
TMPDIR	A-8

Introduction

The following environment variables can all be set using the AIX VS COBOL system:

- COBATTR
- COBCPY
- COBCTRLCHAR
- COBDIR
- COBHELP
- COBIDY
- COBLPFORM
- COBOPT
- COBPATH
- COBPRINTER
- COBSW
- TMPDIR

Note: If you are using the *ksh* shell, you must export any environment variables you set before you can use them.

Full details on all these environment variables can be found throughout this manual. However, the following sections give a brief summary of each environment variable.

COBATTR

COBATTR changes the behavior of the high- and low-intensity attributes to determine whether text appears highlighted on your screen.

Format

The format of COBATTR is as follows:

```
COBATTR=n
```

where *n* can be any of the following values:

- 0** 0 is the default. Highlighted text appears in the high-intensity mode for terminals that support a high-intensity attribute but no low-intensity attribute. Highlighted text appears in normal mode for terminals that support low-intensity mode as the default mode for text that is not highlighted.
- 1** Highlighted text always appears in high-intensity mode. Low-intensity mode is never used.
- 2** High- and low-intensity space characters are not assumed to be the same as normal mode space characters.
- 3** Same as for 1 and 2 above.

High- and low-intensity modes are as described in terminfo. See Chapter 10, “Configuring Your AIX VS COBOL System,” for details.

Example

The following is an example of the COBATTR variable:

```
COBATTR=1
```

COBCPY

COBCPY specifies the directory that the compiler and ANIMATOR are to use to search for COPY files.

Format

The following example shows the format of the COBCPY variable:

```
COBCPY=path-name
```

where *path-name* specifies the path that the compiler and ANIMATOR are to search.

Example

The following is an example of the COBCPY variable:

```
COBCPY=:src/lib/mylib:$HOME/mylib
```

If multiple directories are specified, the first character must be a colon(:).

COBCTRLCHAR

When COBCTRLCHAR is set, the AIX VS COBOL system will allow the use of raw escape sequences. See the full description of this functionality in “Using Escape Sequences to Send Attribute Information to the Screen” on page 9-19.

Support for this feature is provided for compatibility with older code. It is not recommended for use in new code.

Example

To set the COBCTRLCHAR variable to allow the use of raw escape sequences, use the format:

```
COBCTRLCHAR=y
```

COBDIR

By default, the AIX VS COBOL system software is located in the file /usr/lpp/COBOL/lib (for the non-DBCS variety of COBOL) or in /usr/lpp/COBOL/dblib (for the DBCS variety of the system). That is where **cob** normally expects to find it. If you install this system in a different directory, COBDIR allows you to specify the name of the directory in which you installed it. Then, **cob** will search the specified directory for the system software instead of the default directory.

The maximum length of the COBDIR string is 40 characters.

Format

The format of the COBDIR variable is as follows:

```
COBDIR=directory
```

where *directory* is the directory that contains the AIX VS COBOL system software.

Example

The following is an example of the COBDIR variable:

```
COBDIR=/cob32u1
```

This causes **cob** to search directory **/cob32u1** for the AIX VS COBOL system software.

COBHELP

COBHELP specifies the path to search to find help files to be displayed to the user upon request.

Format

The following example shows the format of the COBHELP variable:

```
COBHELP=path-name
```

Example

The following is an example of the COBHELP variable:

```
COBHELP="/usr/LPP/COBOL/lib/help"
```

COBIDY

COBIDY specifies the path(s) that ANIMATOR will search for **.idy** files if the required files are not found in the current directory nor in the directory specified on the command line.

Format

The following example shows the format of the COBIDY variable:

```
COBIDY=path-name
```

where *path-name* specifies the path that the ANIMATOR is to search.

Example

The following is an example of the COBIDY variable:

```
COBIDY="/usr/LPP/COBOL/lib/usr/myfile"
```

COBLPFORM

The AIX VS COBOL system emulates printer channels C01 through C12 by line feeds and form feeds. COBLPFORM allows you to define the line numbers on the form so you can write to these printer channels.

Format

The following example shows the format of the COBLPFORM variable:

```
COBLPFORM="n:::n:::n:::n"
```

where *n* are digits that specify the line number you require for the relevant channel.

Note: Any channels that have line number 0; mnemonics S01, S02, or CSP; or are undefined are set to line 1, which is the beginning of the page.

Example

The following is an example of the COBLPFORM variable:

```
COBLPFORM="1:::60"
```

This sets channel 1 to line 1 and channel 12 to line 60.

COBOPT

COBOPT can do either of the following:

- Contain options that supplement or override the system-wide default compiler and Native Code Generator options as defined in the file **\$COBDIR/cobopt**
- Specify the path of a file which contains such options.

If you use COBOPT to point to such a file, this file must be in the same format as **\$COBDIR/cobopt**. See Chapter 5, "Compiler Options," for details.

Format

The following example shows the format of the COBOPT variable:

```
COBOPT="compiler:[COBOL-COMPILER-OPTION]... ]  
        ncg:[NCG-OPTION]... ]  
        [SET environment-variable=value... ]  
        [; comment-entry... ]"
```

where:

COBOL-COMPILER-OPTION is one or more of the compiler options listed in Chapter 5, "Compiler Options."

NCG-OPTION is one or more of the native code generator options listed in Chapter 6, "Native Code Generator Options."

environment-variable can be any environment variable supported by the AIX VS COBOL system.

value is the value to which you want to set the specified environment variable.

comment-entry can be any statements you wish **cob** to treat as comment lines.

Example

The following is an example of the COBOPT variable:

```
COBOPT="compiler:ANS85  
        SET COBCPY=$COBDIR/src/lib:$HOME/mylib::"
```

This enables ANS85 standard COBOL syntax and sets COBCPY to the specified path.

COBPATH

COBPATH specifies which directories the Run Time Environment (RTE) is to search for a specified file. This variable is also used to specify CALL paths. (It is not used to locate data files.)

Format

The following example shows the format of the COBPATH variable:

```
COBPATH=[:] path-name [:path-name]...
```

where:

the initial ":" indicates that the current directory is to be searched first.

path-name specifies the directories and their paths that the RTE is to search.

Example

The following is an example of the COBPATH variable:

```
COBPATH=/u:/v:/qa/src/lib:qa/otherlib
```

COBPRINTER

COBPRINTER specifies the name of the printer to which the output from a DISPLAY UPON PRINTER statement is directed.

Format

The following example shows the format of the COBPRINTER variable:

```
COBPRINTER=print-command
```

where *print-command* can be the name of any print command supported by your system, for example, /bin/print.

Example

```
COBPRINTER="/bin/print rp2"  
export COBPRINTER
```

This example shows that you can name alternate printers on your system. If no printer is named, the default printer on the system will be used.

COBSW

COBSW sets various switches when you execute files output by **cob**.

Format

The following example shows the format of the COBSW variable:

```
COBSW=[ $\pm$ switch]
```

where:

\pm enables or disables the specified switches. Switches can have one of the following values:

0 to 8 run-time switches

- A** ANIMATOR switch
- B** Skip locked record switch
- D** ANSI COBOL debug switch
- e** COBOL symbol switch
- E** Error switch

F	Compatibility check switch
i	Keyboard interrupt switch
K	ISAM files sequence check switch
l	Memory switch
N	Null switch
p	Dynamic linkage setup switch
Q	Ryan - McFarland (RM) file status error switch
R	Reread locked record switch
S	SORT switch
T	TAB switch

All of these switches are described in detail in Chapter 7, "Running an AIX VS COBOL Program."

Example

The following is an example of the COBSW variable:

```
COBSW=+0+D
```

This starts run-time switch 0 and the ANSI COBOL debug switch.

TMPDIR

TMPDIR specifies an alternate directory to use for temporary files, rather than the default **/tmp** directory.

Format

The following example shows the format of the TMPDIR variable:

```
TMPDIR=path-name
```

where the *path-name* specifies the directory name for temporary files.

Example

The following is an example of the TMPDIR variable:

```
TMPDIR=/usr/alternate/tmp
```

Appendix B. National Language Support

Introduction	B-3
Features Provided by National Language Support	B-3
Compiling Programs with National Language Support	B-4
Running Programs with National Language Support	B-5
Running Your Program	B-5
RTE NLS Initialization	B-6
String Comparisons	B-6
Class Condition Tests	B-6
Indexed Sequential File Operations	B-7
Comparisons Performed as Part of SORT or MERGE Statements	B-7
The NLS Support Routines	B-7
Mixing Programs with and without National Language Support	B-8

Introduction

AIX VS COBOL has two facilities that allow an application to take advantage of many of the features provided by the NLS component of the AIX operating system. West European languages are supported by the NLS facility of AIX VS COBOL while Japanese and Asian languages are supported by the DBCS facility of AIX VS COBOL. An application cannot be compiled to support both NLS and DBCS at the same time. This section describes the NLS facility of AIX VS COBOL.

To handle 8-bit character sets, you must configure your terminal to support this. See your operating system documentation for details on how to configure your terminal (for example, the *stty* command).

Features Provided by National Language Support

The NLS facility makes it unnecessary for you to specify the following clauses in your program:

- ALPHABET
- PROGRAM COLLATING SEQUENCE
- COLLATING SEQUENCE for **sort-merge** files
- CURRENCY SIGN
- DECIMAL POINT IS COMMA

In addition, new facilities are provided as follows:

- Class condition tests (ALPHABETIC and NUMERIC) appropriate to national and multinational character sets.
- Linguistic collating sequences appropriate for West European languages as provided by the AIX system.
- Linguistic collating sequences applied to **indexed** files.
- Linguistic collating sequences applied to **sort-merge** files.

You can use the NLS facility by specifying the compiler option **nls**. See “Compiling Programs with National Language Support” on page B-4 for information on using the **nls** option.

Compiling Programs with National Language Support

To use the NLS facility in your program, you must compile the program with the compiler option **nls** on the command line. The option has the following two forms:

- **nonls**

This is the default value of the directive. If you compile a program with **nonls**, the program will run exactly as described in the *Language Reference*.

- **nls**

Your program will use the NLS facility for the following:

- Explicit string comparisons, class condition tests, and numeric editing
- Key comparisons in indexed sequential file operations
- Comparisons performed as part of a SORT or MERGE operation.

When you compile a program with the **nls** option, the compiler will not accept the following elements of syntax in your program:

- The following phrase in the OBJECT-COMPUTER paragraph:

PROGRAM COLLATING SEQUENCE IS *alphabet-name*

- The following phrases in the SPECIAL-NAMES paragraph:

- alphabet-name IS {
 STANDARD-1
 NATIVE
 literal-1 THRU literal-2 etc. }
- CURRENCY SIGN IS literal
- DECIMAL-POINT IS COMMA

- The following phrase in MERGE or SORT statements:

- COLLATING SEQUENCE IS *alphabet-name*

If your program contains any of this syntax, the program will fail compilation with the following error:

```
136 Illegal use of phrase for National Language operation
```

You must remove this syntax (or mark it as a comment line) and recompile your program before you can successfully run your program.

Running Programs with National Language Support

Before you run a program with National Language Support (NLS), you must set up the environment. Use the LANG environment variable to do this. For example:

```
LANG=Sp_SP
export LANG
```

AIX uses the value of LANG to specify the language environment to be used by a process. The collating table for a particular language is located in the file: `/usr/lib/nls/$LANG`. The file `/usr/lib/nls/$LANG.en` contains a number of “name = value” pairs that specify characteristics of a language environment. In particular, AIX VS COBOL uses the value of NLCURSYM for the value of the currency symbol, while the comma and decimal point become the values of the triad and decimal punctuation symbols contained in NLNUMSEP. Because COBOL requires the values for all of these symbols to be a single character, only the first character is used. Further, the currency symbol is always printed to the left of a numeric value even if the language specifies that it should be printed on the right. For more information about National Language Support provided by the AIX system, see the AIX system documentation.

If the language specified in the LANG environment variable is not available, the RTE gives the following error message:

```
40 Language Initialization not set up correctly
```

and the program will terminate.

Running Your Program

You can run programs with National Language Support the same way that you run programs without this facility. See Chapter 7, “Running an AIX VS COBOL Program” for more information.

When you run a program compiled with the `nls` option, the following operations will use the facilities provided by the operating system for the language specified in the LANG environment variable:

- String comparisons
- Class condition tests
- Key comparisons in indexed sequential files
- Comparisons performed as part of a SORT or MERGE statement.

If you have specified the use of decimal points, commas, or currency symbols in your program, these constants contain the symbols required for the language specified. However, although it is possible to specify the currency symbol as trailing to the AIX operating system, AIX VS COBOL always displays the currency symbol as leading.

Certain NLS definitions have characters other than the ASCII characters 0-9 defined as numerics. Although such characters are identified as numeric by the operating system, they cannot be ACCEPTed into numeric picture strings, nor can they be used in numeric operations. In all NLS operations in the COBOL environment, a numeric item must be formed only from the ASCII digits 0-9, with or without the ASCII operational signs “+” or “-”. There is no means of automatically converting the NLS representation to the ASCII equivalent.

It is possible to enter European modifying characters into numeric ACCEPT fields. These are accepted as zero.

The values assigned to figurative constants, for example LOW-VALUES, are not changed by using NLS features.

You can also use the ADIS Flip Case Control key when using NLS characters. However, if you attempt to convert a European character to uppercase using this key, the character will not be replaced if it has no uppercase equivalent.

RTE NLS Initialization

The RTE initializes the NLS facility only once during an application's run. It does this when it encounters the first program within the suite which was compiled with the `nls` option set. It uses the `LANG` environment variable to determine which language environment to set up for the application. The RTE uses the same language environment for any subsequent programs within the suite which are compiled for NLS. See “Mixing Programs with and without National Language Support” on page B-8 for details. If an error occurs during this initialization process, for example the language specified in the `LANG` environment variable is not supported, the RTE returns the error:

```
40 Language Initialization not set up correctly
```

and terminates its run.

String Comparisons

For programs that have been compiled with the `nls` option, the RTE invokes one of the routines: `nl_cobcmp()` (for comparisons of two regular strings) or `nl_coball()` (for comparisons that use the ALL phrase). For details of how these two routines operate, see “The NLS Support Routines” on page B-7.

During a MOVE operation of one alphanumeric item to another which is longer, padding with spaces occurs. Similar padding is also implied before the comparison of two such items. In both cases, an ASCII space is assumed.

The logical length of a string may be different from its physical length. For example, the spanish “LL” character is considered a single character during comparisons. Also, the german “ß” character is considered equivalent to the two characters “ss.” So in german, “ß” = “ss” even though their physical lengths are different.

Class Condition Tests

For programs which have been compiled with the `nls` option set, the RTE invokes operating system routines when it needs to carry out class condition tests. These tests determine if a string of information is in ALPHABETIC, ALPHABETIC-UPPER, ALPHABETIC-LOWER, NUMERIC, or user-defined CLASS condition format. The class conditions are performed on a character by

character basis on a string so that the results may sometimes be unexpected. For example, in spanish the string "Ch" is considered uppercase, but AIX VS COBOL considers it mixed case. The numeric test always tests that all characters are in the range of ASCII 0-9.

Indexed Sequential File Operations

If the logical filename of an indexed sequential file is preceded by the five characters %NLS%, the file is treated as an NLS file. This is true either before or after it has been resolved by use of environment variables, as described in Chapter 3, "Device- and File-Handling." In the following example, *file1* is the name of the NLS file:

```
SELECT filename ASSIGN TO "%NLS%file1"
```

When redirecting *file1* using environment variables, use the name *dd_file1*. Do not include the %NLS% in the environment variable's name.

A program compiled with the **nls** option may open both NLS and non-NLS indexed files, but a program that has not been compiled with the **nls** option may not open an NLS file. If a non-NLS program attempts to open an NLS file and there is no file-status specified for the file the program terminates with the following error:

```
44 Attempt to OPEN an NLS file in a non-NLS program.
```

A file created as an NLS file must be opened as an NLS file and vice-versa; if not, the OPEN fails and the RTE returns an error. If there is no file status specified for the file, the program terminates with the following error:

```
45 Attempt to OPEN a file with an incompatible NLS attribute.
```

It is an error to take an indexed file created for use with one language and then use it with another language. The RTE cannot detect when a file has been created with one language and then used in a different language. If an application attempts to do this, unexpected results occur and the file may become unusable.

The indexes of an NLS file are compared using the function **nl_cobsort()**.

See "The NLS Support Routines" for more details.

Comparisons Performed as Part of SORT or MERGE Statements

All files sorted in a program compiled with the **nls** option are sorted according to the collating sequence for the language specified. The RTE routine **nl_cobsort()** is used for the comparisons. See "The NLS Support Routines" for more details.

The NLS Support Routines

The NLS string comparison routines provided by AIX follow the C programming language rules. This means that strings are terminated by a null (LOW-VALUE) character. In COBOL, a null character is not a terminator, it is a valid character. Further, COBOL requires that when two strings of different length are compared, the shorter one is to be considered extended with blanks so that it matches the length of the other string. Finally, COBOL allows a string to be compared with another using the ALL phrase. For example, the following condition is TRUE in AIX VS COBOL:

```
if "ababab" = all "ab"
```

Because of these requirements, the NLS routines in the C library were modified to fulfill the requirements of COBOL. The following routines are written in C. To access them you need to follow the guidelines for calling a C program from COBOL outlined in Chapter 2, "Advice on Writing COBOL Programs."

The definition of `nl_cobcmp()` is:

```
nl_cobcmp (string1, string2, length1, length2)
char * string1, string2;
size_t length1, length2;
```

This function takes two strings of given lengths and compares them using NLS collation rules. The shorter string is logically extended by blanks and nulls are allowed.

Return Value: Returns a negative value if `string1 < string2`
Returns a zero value if `string1 = string2`
Returns a positive value if `string1 > string2`

The definition of `nl_coball()` is:

```
nl_coball (string, all_string, length, all_length)
char * string, all_string;
size_t length, all_length;
```

Return Value: Returns a negative value if `string < all_string`
Returns a zero value if `string = all_string`
Returns a positive value if `string > all_string`

This function is similar to `nl_cobcmp`, except that the *all_string* is repeated until the comparison with *string* is done.

```
nl_cobsort (string1, string2, length1, length2)
char * string1, string2;
size_t length1, length2;
```

This function is identical to `nl_cobcmp` except that ties are broken. Since comparisons are based upon the collating sequence of a character set it is possible for two different characters to collate the same. For example, "A" may collate the same as "a". In regular COBOL comparisons, only the result from the collation is used ("A" = "a" in this example). But for a sort it is desirable for strings that are equal in collation but different in actual characters to be grouped in a predictable manner. For example, consider the strings:

```
"A" "a" "A" "a"
```

The function `nl_cobcmp` would consider all of these characters equal in value for languages where "A" is equivalent to "a". Therefore, a sort based on `nl_cobcmp` would be inadequate. `nl_cobsort()` would not consider "A" as equal to "a".

Mixing Programs with and without National Language Support

You can call a program with National Language Support from a program without NLS and can also call programs without NLS from a program with it. When you enter a program with NLS from a program without NLS, or vice versa, the language environment will automatically change.

Data may be shared between the NLS and non-NLS programs, however, one must be aware that the non-NLS program obeys the standard AIX VS COBOL rules

when operating on such data. For example, a string may sort differently in the NLS routine than it would in the non-NLS routine.

Also note the restrictions placed on NLS indexed files. See “Indexed Sequential File Operations” on page B-7.

Appendix C. Character Sets and Collating Sequence

The following table shows the COBOL collating sequence. An X in the third column indicates that that symbol is not part of COBOL syntax.

ASCII	HEX	COBOL
NUL	00	X
SOH	01	X
STX	02	X
ETX	03	X
EOT	04	X
ENQ	05	X
ACK	06	X
BEL	07	X
BS	08	X
HT	09	X
LF	0A	X
VT	0B	X
FF	0C	X
CR	0D	X
SO	0E	X
SI	0F	X
DLE	10	X
DC1	11	X
DC2	12	X
DC3	13	X
DC4	14	X
NAK	15	X
SYN	16	X
ETB	17	X
CAN	18	X
EM	19	X
SUB	1A	X
ESC	1B	X
FS	1C	X
GS	1D	X

Table C-1 (Page 2 of 4). Character Set and Collating Sequence		
ASCII	HEX	COBOL
RS	1E	X
US	1F	X
space	20	
!	21	X
"	22	
#	23	X
\$	24	
%	25	X
&	26	X
'	27	
(28	
)	29	
*	2A	
+	2B	
,	2C	
-	2D	
.	2E	
/	2F	
0	30	
1	31	
2	32	
3	33	
4	34	
5	34	
6	36	
7	37	
8	38	
9	39	
:	3A	
;	3B	
<	3C	
=	3D	
>	3E	
?	3F	X

Table C-1 (Page 3 of 4). Character Set and Collating Sequence		
ASCII	HEX	COBOL
@	40	X
A	41	
B	42	
C	43	
D	44	
E	45	
F	46	
G	47	
H	48	
I	49	
J	4A	
K	4B	
L	4C	
M	4D	
N	4E	
O	4F	
P	50	
Q	51	
R	52	
S	53	
T	54	
U	55	
V	56	
W	57	
X	58	
Y	59	
Z	5A	
[5B	X
\	5C	X
]	5D	X
^	5E	X
_	5F	X
'	60	X
a	61	

Table C-1 (Page 4 of 4). Character Set and Collating Sequence

ASCII	HEX	COBOL
b	62	
c	63	
d	64	
e	65	
f	66	
g	67	
h	68	
i	69	
j	6A	
k	6B	
l	6C	
m	6D	
n	6E	
o	6F	
p	70	
q	71	
r	72	
s	73	
t	74	
u	75	
v	76	
w	77	
x	78	
y	79	
z	7A	
{	7B	X
	7C	X
}	7D	X
~	7E	X
DEL	7F	X

Appendix D. Packaging Application Programs

Introduction	D-3
The Run Time Package	D-3
Preparing Application Packages	D-3
Statically Linkable Native Code (.o)	D-3
Dynamically Loadable Native Code (.gnt)	D-4
Intermediate Code (.int)	D-4

Introduction

There are several ways an application program can be packaged for distribution to an end user. The run-time routines are available as an IBM product for an application customer to purchase as a prerequisite for the application package. All code developed with the AIX VS COBOL Compiler/6000 will need the AIX VS COBOL Run Time Environment/6000 Version 1.1 in order to execute.

This chapter explains how to compile application programs that will be distributed and explains the procedures that the end customer must follow to run the application package. Application developers should document the final installation procedures for their end customers based on these guidelines.

The Run Time Package

The Run Time package, available through your IBM representative, your IBM Authorized Dealer, or your IBM Authorized Remarketer will contain the functions needed to run a COBOL program developed with AIX VS COBOL (libraries, file handler routines, screen input and output functions, error messages). The **cob** and **cobrun** commands will also be included, to allow the application customer to bind the application programs with the RTE, if necessary.

Preparing Application Packages

The programs that comprise the application package can be distributed when compiled to .int, .o, or .gnt code. The procedures that an application customer needs to execute vary according to the type of code used. These procedures can be explained as part of the installation process or provided in a shell script which would facilitate the end user's install procedures. Compiler options may be included on the command lines, although these options are omitted from the examples here. The application program will be referred to as *application* in the following examples.

Statically Linkable Native Code (.o)

The application programs can be compiled to object code, using the following command:

```
cob -x application.cbl
```

This produces an executable module, *application*, that the application customer would enter to invoke the application. This will be the preferred packaging for most application programs.

Dynamically Loadable Native Code (.gnt)

The application programs can be compiled to dynamically linked code using the following command:

```
cob -u application.cbl
```

The application buyer could be instructed to run the application by entering the following command:

```
cobrun application.gnt
```

A shell script containing this command could be provided, allowing the application buyer access to the application by using a single executable module name.

Intermediate Code (.int)

Intermediate code can be run on any system that supports any of the following:

- IBM COBOL/2
- AIX VS COBOL

The application programs can be compiled to intermediate code using the following command:

```
cob -i application.cbl
```

The application customer would invoke the application using the following command:

```
cobrun application.int
```

A shell script containing this command could be provided allowing the application customer to access the application by using a single executable module name.

Glossary

adiscf. A utility for setting up the ADISCTRL database.

ADISCTRL database. The configuration file for the AIX VS COBOL ADIS Accept/Display module. Use the adiscf utility to alter entries in this database.

ANIMATOR. An interactive program debugging tool for use with AIX VS COBOL programs.

automatic locking. A type of data locking, in which a single record or multiple records are locked by accessing them.

batch mode. A method for passing parameters to a module by specifying them on the command line or in a file instead of giving them interactively.

binary sequential file. A sequential file that contains binary records.

cob command. The command used to access the various components of the AIX VS COBOL system: the compiler, code generator, and linker.

cobkeymp database. A file that contains a mapping of control characters and terminfo codes onto a standard set of function keys that the ADIS ACCEPT/DISPLAY module can recognize. Set up this file with the keybcf utility.

cobrun. The command used to execute an intermediate code or dynamically loadable native code program.

COBSW. The environment variable that controls the settings of the switch parameters.

collating sequence. The sequence in which characters are ordered within the computer for sorting, combining, or comparing.

compiler. A program that converts a high-level language (such as COBOL) source program to a module that can be executed on some machine (such as the RT).

computational data. A type of COBOL data designed specifically for numeric computations.

convert3. A program to convert RM/COBOL data files to AIX VS COBOL data files.

convert5. A program to convert DGCBOBOL data files to AIX VS COBOL data files.

data locking. A mechanism by which a program may lock certain file records, preventing other programs from modifying the file records.

DGCBOBOL. Data General Interactive COBOL Rev 1.30.

dynamically loaded executable module. An executable AIX VS COBOL module that loads its overlays or other procedures as needed.

environment. The settings for shell variables and paths associated with each process.

environment variable. A variable defined in the AIX shell that, if exported, is global to all processes run under that shell.

exclusive locking. A type of data locking in which entire files are locked as soon as they are opened by a program.

file conversion program. A program created by convert3 that will convert RM/COBOL type files to AIX VS COBOL type files.

flag. A modifier preceded by a dash that appears on a command line as an option to a command.

FORMS-2. A tool that can be used to interactively create and edit display screens for use in AIX VS COBOL programs.

hexadecimal value. The base 16 representation of a number.

index program. A program generated by the FORMS-2 package for creating and maintaining an indexed sequential file.

indexed sequential files. A type of file organization implemented as a pair of files, with a key or index file that controls access to data records in the data file.

information line. A line at the bottom of the ANIMATOR screen that describes the current state of the program being animated.

interactive mode. A method for giving parameters to a module by responding to prompting from the module.

intermediate code. Code that is in a format acceptable for running using the cobrun command.

ISAM. Indexed Sequential Access Method.

keybcf. A utility for setting up the cobkeymp database.

line-sequential files. Files that consist of a series of variable-length records, each of which is terminated by the character hex 0A.

linker. A program that takes unlinked a.out format files and resolves all external references to produce an executable module.

literal. A group of characters enclosed in quotation marks.

manual locking. A type of data locking in which a single record or multiple records are locked by issuing statements that explicitly lock them.

National Language Support (NLS). A facility that provides a means of adapting your program to the character set used by a non-English-speaking country.

native code. Code that is in a form that can be executed directly on the machine hardware. (On AIX this is in the a.out object module format.)

Native Code Generator. The pass of the COBOL compiler that takes an input intermediate code file and generates an AIX a.out format file.

numeric editing. A COBOL data field description for representing numeric data that contains a decimal point.

octal value. The base 8 representation of a number.

paging. The action of transferring instructions, data, or both between real memory and external page storage.

parameter. Information that a user supplies to a command.

profiler. A tool that can be used to obtain detailed statistics on the run-time performance of your program.

random access mode. A mode for accessing data in a nonsequential manner.

record-sequential files. Files consisting of a series of fixed-length records. The length of a record is the length of the longest FD entry for the file in the FILE section of the program.

relative files. Files that allow you to access data randomly by specifying its position within the file. These files consist of fixed-length records, each of which is uniquely identified by a record number.

RM/COBOL. The Ryan McFarland 2.0 dialect of COBOL.

Run-Time Environment (RTE). A module that interprets intermediate code and provides various support services to native code.

Run-Time System (RTS). See *Run-Time Environment*

statement. An instruction in a program or procedure.

statically linked executable module. A module in the form of a standard AIX a.out executable object module that has all of its overlays and procedures linked into memory.

switch parameters. The switches that affect the way your AIX VS COBOL program is run.

terminfo database. A database containing terminal descriptions.

Index

- A flag 4-12
- C option flag 4-12
- CC option flag 4-12
- D option flag 4-12
- d *symp* flag 4-10
- e *epsym* flag 4-10
- F option flag 4-13
- g flag 4-10
- i flag 4-10
- k *ext* flag 4-10
- L *dir* flag 4-13
- l *key* flag 4-11
- m *symp* = *newsym* flag 4-11
- N option flag 4-13
- o filename flag 4-11
- O flag 4-13
- p flag 4-11, 4-13
- pg flag 4-11
- S flag 4-14
- T option flag 4-14
- u flag 4-12, 4-14
- v flag 4-12, 4-14
- W *err-lever* flag 4-14
- x flag 4-12
- X *symp* flag 4-15

A

- A flag 4-12
- ACCEPT statement 7-4
- ACCEPT/DISPLAY 9-9, 9-14, 10-16
- ADIS key control 10-25
- adiscf utility 10-14
- ADISCTRL 10-5
- AIX VS COBOL
 - calling non-COBOL programs 2-15
 - calling subprograms 2-11
 - compiler flags 15-47
 - converting applications 13-6
 - converting DG Interactive COBOL applications 14-4
 - data descriptions 12-29
 - debug switch (D) 7-8
 - development cycle 4-5
 - devices 3-4
 - DG data types 14-10
 - disk file structure under AIX 3-12
 - enhancing converted applications 14-4
 - facilities within multi-user 8-4

AIX VS COBOL (*continued*)

- file assignment 3-4
- file details 14-13
- installation 1-8
- interface command 4-4
- introduction 1-4
- migrating 13-7
- optimizing programs 2-4
- option specification 4-7
- permitted options 6-4
- program development cycle 1-6
- program source conventions 1-7
- programming restrictions 2-5
- search sequence 2-11
- source compatibility 13-9
- submitting an application 13-7
- submitting source programs 13-6
- symbol switch (-e) 7-8
- syntax checking errors 15-4
- system components
 - ANIMATOR 1-5
 - compiler 1-5
 - dynamically loaded code 1-6
 - FORMS-2 1-5
 - linker 1-5
 - Native Code Generator 1-5
 - Run Time Environment 1-5
 - static linking and dynamic loading 1-6
 - statically linked code 1-6
- work phase 12-18
- alter option 10-16
- altering
 - ACCEPT-DISPLAY options 10-16
 - ADIS key control 10-25
 - CRT-UNDER-HIGHLIGHTING
 - options 10-16
 - indicators 10-23
 - message and indicator positions 10-24
 - messages 10-23
 - tab stop options 10-22
- alternate file status table 3-18
- ANIMATOR
 - break points
 - setting 11-11
 - unsetting 11-12
 - using 11-11
- CALLed programs 11-17
- command line switches 11-16

ANIMATOR (continued)

- commands 11-8
 - align 11-20
- ANIMATOR 11-19
- eXchange 11-20
- go 11-10
- help 11-19
- lookup 11-21
- step 11-9
- summary 11-41
- view 11-20
- where 11-20
- word-left 11-21
- word-right 11-21
- zoom 11-11

- cursor control keys 11-18
- data item commands
 - add 11-36
 - before 11-36
 - brother 11-35
 - clear 11-34
 - cursor-name 11-38
 - delete 11-36
 - do 11-40
 - down-table 11-35
 - enter-name 11-39
 - find 11-37
 - following 11-37
 - hex and text 11-34
 - heX/ASCII 11-35
 - join 11-40
 - locate 11-37, 11-38
 - monitor 11-34
 - next 11-36
 - other menu 11-36
 - parent 11-35
 - previous 11-36
 - refresh 11-40
 - son 11-35
 - split 11-39
 - text 11-39
 - up-table 11-35
 - update 11-36
- description 1-5
- directives 11-6
- display screen 11-7
- ending 11-14
- escape key 11-21
- facilities not supported 11-5
- file searches 11-16
- getting started 11-5
- help display screens 11-9

ANIMATOR (continued)

- letter commands
 - animation speed 11-22
 - back track 11-31
 - break 11-24
 - cancel-all 11-25
 - cursor name 11-32
 - cursor position 11-24
 - do 11-26
 - dump-list 11-33
 - enter-name 11-32
 - env 11-27
 - examine 11-25, 11-31
 - exit 11-23
 - go 11-22
 - if 11-25
 - monitor-off 11-33
 - next 11-24
 - next-if 11-22
 - on-count 11-27
 - perform 11-23
 - program-break 11-27
 - query 11-32
 - quit-perform 11-24
 - repeat 11-33
 - reset 11-23
 - set 11-24
 - start 11-24
 - step 11-21, 11-23
 - threshold-level 11-28
 - unset 11-25
 - until 11-30
 - zoom 11-22
- programs 11-15
- running 11-6
- stockI demo program 11-9
- switch 7-7, 11-15
- animator switch A-7
- ANSI COBOL debug switch 7-8
- ANS85 options 5-24
- arithmetic of group level items 14-6
- ASCII 5-6, 5-14, 5-21
- assigning files 3-4
- audible alarm 9-13
- automatic locking 8-5

B

- background/foreground 12-19
- bcheck utility 3-14
- binary sequential files 13-28

break points 11-11

C

C option flag 4-12
CALLED programs 11-17
CANCEL statement 2-9
cancelling non-COBOL subprograms 2-18
CC option flag 4-12
character sets B-3, C-1
choose option 10-29
CISAM file handler 9-26
cob command 4-4
COBATTR A-3
COBCPY 3-16, A-4
COBCTRLCHAR A-4
COBDIR 2-12, A-4
COBHELP A-5
COBIDY A-5
cobkeymp 10-5
coblongjmp library routine 9-4
COBLPFORM 3-4, A-5
COBOL interface command 4-4
COBOL profiler
 description 7-14
 directives 7-14
 output 7-15
COBOL symbol switch 7-8
COBOPT 4-7, A-6
COBPATH 2-12, A-6
COBPRINTER A-7
cobprof 7-14
cobrun 7-4
cobsetjmp library routine 9-4
COBSW 7-6, A-7
cobtidy 2-17, 9-5
code problems, executable 13-14
collating sequence C-1
column number specification 13-13
command line
 conventions 4-16
 examples 4-17
 options 4-8
 passing 2-21
 switches 11-16
 syntax 7-4
command mode 12-21
commands
 align 11-20
 ANIMATOR 11-19
 eXchange 11-20
 go 11-22

commands (*continued*)

 help 11-19
 letter
 See letter commands
 lookup 11-21
 selected data item
 See selected data item commands
COMP-3/COMPUTATIONAL-3 data 13-21
COMP-6/COMPUTATIONAL-6 data 13-22
compatibility check switch 7-8
compile for animation flag (-a) 4-9
compile to statically linkable object module flag
 (-c) 4-9
compiler flags 15-47
compiler messages 5-29
component definitions 9-26
COMPUTATIONAL data types 13-10
COMPUTATIONAL-1 data types 13-10
COMPUTATIONAL-6 data types 13-10
COMP/COMPUTATIONAL data 13-20
console display screen 12-16
conversion problem solving 13-11
converting data files 13-6
convert3
 converting data files 13-6, 13-20
 error messages 13-35
 escape 13-29
 generate program 13-28
 parameter file 13-29
 program modifications 13-23
 running 13-24
 running in batch mode 13-29
 running with a parameter file 13-33
 tabx program 13-8
convert5
 error messages 14-22
 escape 14-17
 FD parameter 14-18
 file conversion process 14-12
 file conversion program 14-20
 generate program 14-16
 help 14-13
 IDENTIFIER parameter 14-18
 indexed files 14-10
 LISTFILE parameter 14-17
 PARAMETER file 14-17
 printfile name 14-14
 PROGRAM parameter 14-18
 record number parameters 14-18
 reformatting a DG source file 14-6
 relative files 14-8
 running 14-12

convert5 (*continued*)

- sequential files 14-8
- source file restrictions 14-11
- SOURCEFILE parameter 14-17
- SUBROUTINE parameter 14-18
- supported data file types 14-8
- COPY files 3-16, 5-16, A-4
- CRT Screen handling 9-14
- currency sign 12-16
- currency symbol B-3
- cursor control keys 11-18

D

- D option flag 4-12
- d symb flag 4-10
- data
 - descriptions 12-29
 - locking 8-5
 - name base 12-16
 - names and program identification 13-13
 - naming 12-29
- data item commands for ANIMATOR
 - add 11-36
 - before 11-36
 - brother 11-35
 - clear 11-34
 - cursor-name 11-38
 - delete 11-36
 - do 11-40
 - down-table 11-35
 - enter-name 11-39
 - find 11-37
 - following 11-37
 - hex and text 11-34
 - heX/ASCII 11-35
 - join 11-40
 - locate 11-37, 11-38
 - monitor 11-34
 - next 11-36
 - other menu 11-36
 - parent 11-35
 - previous 11-36
 - refresh 11-40
 - son 11-35
 - split 11-39
 - text 11-39
 - up-table 11-35
 - update 11-36
- DBCS support 5-8, 10-22
- dbx 4-10

- DDS file 12-30
- debug switch 7-8
- debugging 4-10, 11-4
- decimal point 12-16
- DECIMAL-POINT IS COMMA clause 13-24
- default options 5-24, 6-5
- delete option 10-28
- demonstration programs 8-13
- DEPENDING names 13-24
- devices 3-4
- DGCOBOL

- arithmetic of group level items 14-6
- calls 14-6
- converting applications to AIX VS COBOL 14-4
- converting data files 14-7
- convert5 14-7
- data types 14-10
- directive 14-5
- enhancing converted applications 14-4
- exception values 14-6
- file status 14-6
- indexed files 14-10
- international character set 14-5
- linkage section access 14-6
- program identification and data-names 14-6
- reformatting a source file 14-6
- reform5 14-7
- reserved words 14-5
- run-time switches 14-6
- source compatibility 14-5
- directives
 - See options*
- display attributes 9-15
- display data 13-22
- display raw data to screen 9-11, 9-19
- display screen 12-18
- display screen image file 12-33
- display screen input and output 9-11
- dynamic linkage setup switch 7-10
- dynamically loadable native code D-4
- dynamically loaded code 1-6
- dynamically loaded programs 2-13

E

- e epsym flag 4-10
- EBCDIC 5-6, 5-14, 5-21
- edit mode 12-20
- embedded source file options 4-15
- end-of-file notification 13-13

environment variables

- COBATTR A-3
- COBCPY A-4
- COBCTRLCHAR A-4
- COBDIR 2-12, A-4
- COBHELP A-5
- COBIDY A-5
- COBLPFORM 3-4, A-5
- COBOPT A-6
- COBPATH 2-12, A-6
- COBPRINTER A-7
- COBSW 7-7, 7-8, 7-9, 7-10, 7-11, 7-12, A-7
- TMPDIR 4-4, A-8
- ulimit 3-17

error messages

- compiler flags 15-47
- convert5 14-22
- during code generation 15-54
- Native Code Generator 15-54
- Run Time Environment 15-60
- severe compiler 15-7
- syntax checking 15-4
- types
 - exceptions 15-61
 - fatal 15-61
 - file operation 15-60
 - input-output 15-61
 - recoverable 15-60

error switch 7-8

error-handling 3-17

escape 14-17

escape key 11-21

exclusive locking 8-5

F

- F option flag 4-13
- fatal errors 15-61
- FCD information format 9-23
- FD parameter 13-30, 14-18
- field wrap-around 13-17
- file and record locking 13-19
- file assignment
 - dynamic 3-5
 - file name mapping 3-7
 - fixed 3-5
- file conversion program
 - creating 13-34
 - error messages 13-36
 - running 13-34
 - using 13-33
- file handler
 - CISAM 9-26
 - description 9-20
- file name base 12-16
- file name mapping 3-7
- file operation errors 15-60
- file searches 11-16
- file status 3-17, 8-11
- file-related operations 9-8
- files
 - indexed sequential 3-13
 - library 3-16
 - line-sequential 3-12
 - record-sequential 3-12
 - relative 3-12
 - restrictions 3-17
- fixed text 12-30
- fixed text display screens 12-32
- flagging COBOL dialects 2-24, 5-32, 15-5
- flags
 - a 4-9, 4-12
 - c 4-9
 - C option 4-12
 - CC option 4-12
 - D option 4-12
 - d *symb* 4-10
 - e *epsym* 4-10
 - F option 4-13
 - g 4-10
 - i 4-10
 - k *ext* 4-10
 - L *dir* 4-13
 - l *key* 4-11
 - m *symb* = *newsym* 4-11
 - N option 4-13
 - O 4-13
 - o filename 4-11
 - p 4-11, 4-13
 - pg 4-11
 - S 4-14
 - T option 4-14
 - u 4-12, 4-14
 - v 4-12, 4-14
 - W err-level 4-14
 - x 4-12
 - X *symb* 4-15
 - +F option 4-13
- format of compiler options 5-4
- FORMS-2 1-5
 - checkout program 12-31
 - display screen generation example 12-35
 - display screen image file 12-33

FORMS-2 (continued)

- index program 12-39
 - index program example 12-43
 - initialization phase 12-5
 - maintenance 12-33
 - operator interface 12-6
 - outputs 12-5
 - validation 12-7
 - work phase 12-5
 - work phase completion 12-28
- function key definition
- alter 10-10
 - review existing 10-9
 - save 10-13

G

- g flag 4-10
- global information 9-25
- go command 11-10

I

- i flag 4-10
- IDENTIFIER parameter 13-30, 14-18
 - run parameter 14-19
- index program 12-39
- indexed sequential file format 3-14
- indexed sequential files 3-13, 8-9
- information format, FCD 9-23
- initialization display screen 12-16, 12-17
- initialization files
- initialization of WORKING-STORAGE 13-19
- initialization phase 12-16
- input-output error-handling 3-17
- input-output errors 15-61
- installation 1-8
- intermediate code 2-7, D-4
- interprogram communication 2-9
- ISAM files sequence check switch 7-10

J

- join a file name 9-7

K

- k ext flag 4-10
- key definitions for indexed files 9-25
- keybcf utility
 - description 10-6
 - invoking 10-8
 - using 10-9

- keyboard conversion process 10-5
- keyboard interrupt switch 7-9
- keyboard status 9-13

L

- L dir flag 4-13
- l flag 4-11
- l key flag 4-11
- letter commands
 - animation speed 11-22
 - back track 11-31
 - break 11-24
 - cancel-all 11-25
 - cursor name 11-32
 - cursor position 11-24
 - do 11-26
 - dump-list 11-33
 - enter-name 11-32
 - env 11-27
 - examine 11-25, 11-31
 - exit 11-23
 - go 11-22
 - if 11-25
 - monitor-off 11-33
 - next 11-24
 - next-if 11-22
 - on-count 11-27
 - perform 11-23
 - program-break 11-27
 - query 11-32
 - quit-perform 11-24
 - repeat 11-33
 - reset 11-23
 - set 11-24
 - start 11-24
 - step 11-21, 11-23
 - threshold-level 11-28
 - unset 11-25
 - until 11-30
 - zoom 11-22
- library
 - files 3-16
 - subroutines
 - coblongjmp 9-4
 - cobsetjmp 9-4
 - cobtidy 9-5
- limits in AIX VS COBOL system 2-5
- line-sequential files 3-12, 8-6
- LINKAGE SECTION access 14-6
- linker 1-5

LISTFILE parameter 13-29, 14-17
listing format 5-30
load option 10-27

M

m symb = newsymb flag 4-11
mainframe options 5-27
manual locking 8-5
MCS
 See message control system
memory switch 7-10
move the cursor 9-13
mudemo 8-13
multi-user environment 8-4
multi-user syntax 8-4

N

N option flag 4-13
National Language Support (NLS)
 compiling programs B-4
 features provided B-3
 introduction B-3
 mixing programs B-8
 running programs B-5
 specifying 5-14
native code
 generator 1-5
 generator messages 6-6, 15-54
 using 2-7
non-COBOL subprograms 2-15
null switch 7-10
number specification 13-13
numbering segments 13-12

O

o file flag 4-11
O flag 4-13
ON SIZE ERROR clause 13-17
operating system functions 2-23
operation codes 9-21
optimizing native code 2-8
options
 alter 10-16
 altering ACCEPT-DISPLAY 10-16
 altering
 CRT-UNDER-HIGHLIGHTING 10-16
 altering tab stop 10-22
 ANS85 5-24
 choose 10-29
 command line 4-8

options (*continued*)
 default 5-24, 6-5
 delete 10-28
 embedded source file 4-15, 5-28
 excluded combinations 5-23
 format of compiler 5-4
 load 10-27
 mainframe 5-27
 optional user default options 4-7
 permitted 5-5, 6-4
 save 10-28
 specifications 4-7
 system-wide default options 4-7
osex compiler option 3-16
overlying 2-9

P

P flag 4-13
pack byte 9-14
packaging application programs D-3
parameter file 14-17
parameters
 example file 13-32
 FD 13-30, 14-18
 IDENTIFIER 13-30, 14-18
 LISTFILE 13-29, 14-17
 PROGRAM 13-30, 14-18
 record number 13-31, 14-18
 RUN 13-31
 SIGN 13-30
 SOURCEFILE 13-29, 14-17
 SUBROUTINE 13-30, 14-18
 switch 7-6
PERFORM statement behavior 5-16, 13-9
picture generation 12-30
PICTURE strings 13-24
pipes 3-10
pi.cbl 1-9
portability of COBOL syntax 5-32
preparing application packages D-3
printing file using print spooler 3-20
procedure division 8-11
profiler 7-14
program identification and data-names 13-13
PROGRAM parameter 13-30, 14-18
programming restrictions 2-5
programs
 call and cancel 2-9
 cancelling non-COBOL subprograms 2-18
 demonstration 8-13
 development cycle 1-6

programs (*continued*)
dynamically loaded 2-13
large 2-8
mixing C and COBOL 2-18
mudemo.cbl 1-8
optimizing COBOL 2-4
pi.cbl 1-8
source conventions 1-7
stockin.cbl 1-8
stockioa.cbl 1-8
stockiom.cbl 1-8
stockout.cbl 1-8
stock1.cbl 1-8
tabx 13-8
put a character to the screen 9-6

R

read a character from the keyboard 9-7
READ statement 7-4
record
naming 12-29
number parameters 13-31, 14-18
sequential files 3-12, 8-6
REDEFINES clause 13-23
reformatting a DG source file 14-6
reform5 14-7
relative files 3-12, 8-7
reread locked record switch 7-11
reserved words 13-12, 14-5
restrictions
ANSI COBOL standard X3.23 3-17
programming 2-5
SAA 2-6
RM directive 13-9
RM file status error switch 7-11
RTE subprograms 9-5
RUN parameter 13-31
run time
environment 1-5
switches 7-7, 14-6
system error messages 7-13
system errors 15-60
Run Time package D-3

S

S flag 4-14
SAA COBOL compatibility 5-27
SAA compatibility 2-6
save option 10-28

screen handling from C 9-16
segmentation 2-9
segments 13-12
sharing files on multi-user systems 8-12
SIGN parameter 13-30
skip locked record switch 7-7
Sort memory switch 7-6
sort switch 7-12
SOURCEFILE parameter 13-29, 14-17
split/join a file name 9-7
static linking and dynamic loading 1-6
statically linkable native code D-3
statically linked code 1-6
stdin 7-4
step command 11-9
stock1 demo program 11-9
stock.cbl 1-10
STOCK.IT 1-11
STOCK.IT.idx 1-11
SUBROUTINE parameter 13-30, 14-18
switch parameters
ANIMATOR (A) 7-7
ANSI COBOL debug 7-8
COBOL symbol 7-8
compatibility check (F) 7-8
description 7-6
dynamic linkage setup (p) 7-10
error (E) 7-8
examples 7-13
ISAM files sequence check (K) 7-10
keyboard interrupt (i) 7-9
memory (l) 7-10
null (N) 7-10
reread locked record (R) 7-11
RM file status error (Q) 7-11
run-time 7-7
skip locked record (B) 7-7
sort (S) 7-12
tab (T) 7-12
symbol switch 7-8
system-wide default options 4-7

T

T option flag 4-14
tab switch 7-12
tabx program 13-8
terminfo 10-5
test keyboard status 9-13
TMPDIR 4-4, A-8
trailing blanks 13-15

U

u flag 4-12, 4-14
ulimit 3-17
unpack byte 9-14
USAGE IS INDEX clause 13-23
USE procedures 3-18
user attributes 9-9
user default options 4-7

V

v flag 4-12
variable data display screens 12-32
variable data fields 12-30
verbose flag 5-29

W

W err-level flag 4-14
work display screen 12-19
work phase 12-18
work phase completion 12-28
WRITE output directly to a printer 3-20

X

x flag 4-12
X symb flag 4-15

Z

zoom command 11-11

Numerics

8-bit character sets B-3

Special Characters

+F option flag 4-13
\$SET option specification 4-15, 5-28



Reader's Comment Form

User's Guide for IBM AIX VS COBOL Compiler/6000

SC23-2178-00

Please use this form only to identify publication errors or to request changes in publications. Your comments assist us in improving our publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your IBM-approved remarketer. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

- If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.
- If you would like a reply, check this box. Be sure to print your name and address below.

Page	Comments

Please contact your IBM representative or your IBM-approved remarketer to request additional publications.

Please print

Date _____

Your Name _____

Company Name _____

Mailing Address _____

Phone No. () _____
Area Code

No postage necessary if mailed in the U.S.A

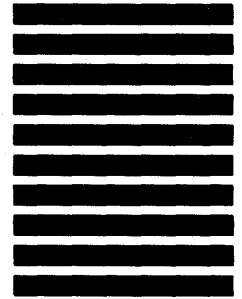


NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 997
11400 Burnet Rd.
Austin, Texas 78758-3493



Fold

Fold

Cut or Fold Along Line

Fold and Tape

Please Do Not Staple

Fold and Tape



© IBM Corp. 1990

International Business Machines
Corporation
11400 Burnet Road
Austin, Texas 78758-3493

Printed in the
United States of America
All Rights Reserved

SC23-2178-00

SC23-2178-00

