


# SYSTEM 2000® PLEX Manual

## Version 12
## First Edition

*The Power to Know®*

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 1995. *SYSTEM 2000® PLEX Manual, Version 12, First Edition*. Cary, NC: SAS Institute Inc.

**SYSTEM 2000® PLEX Manual, Version 12, First Edition**

Copyright © 1995, SAS Institute Inc., Cary, NC, USA

ISBN 1-55544-184-X

All rights reserved. Produced in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

U.S. Government Restricted Rights Notice: Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19, Commercial Computer Software-Restricted Rights (June 1987).

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, June 1995

SAS Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at support.sas.com/pubs or call 1-800-727-3228.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

# Contents

List of Illustrations . . . . . ix
Using This Book . . . . . xi
Changes and Enhancements . . . . . xvii

## USAGE CONCEPTS

### USAGE CONCEPTS

1 OVERVIEW . . . . . 1

SYSTEM 2000 Facilities . . . . . 1
PLEX Capabilities . . . . . 2
The PLEX Processor . . . . . 3
PLEX Commands . . . . . 3
Program Structure . . . . . 3
The Scope of PLEX and This Manual . . . . . 4

2 GETTING STARTED . . . . . 5

Program-Database Communication . . . . . 6
Initial Steps . . . . . 9
Introduction to Retrieval . . . . . 11
Retrieval in Logical Order . . . . . 25
GET1 . . . . . 28
Subset Retrieval . . . . . 36
Linked Retrieval . . . . . 48
Updating a Database . . . . . 52
Saving, Restoring, and Recovering Databases . . . . . 67
Security and Integrity . . . . . 77
Multiple Databases . . . . . 86
Termination . . . . . 88
Program Structure and Processing . . . . . 88

## REFERENCE

### PLEX COMMAND REFERENCE

3 INTRODUCTION . . . . . 3-1

Format of COBOL PLEX Commands . . . . . 3-1

4 SYSTEM 2000 ITEM TYPES AND COBOL DATA DESCRIPTIONS . . . . . 4-1

Introduction . . . . . 4-1
Item Type TEXT . . . . . 4-2
Item Type CHARACTER . . . . . 4-2
Item Type DATE . . . . . 4-3
Item Type INTEGER . . . . . 4-3

Item Types DECIMAL and MONEY . . . . . 4-5
Item Types REAL and DOUBLE . . . . . 4-7
Item Type UNDEFINED . . . . . 4-7
Item Types and COBOL Data Descriptions of Items in Where-clauses . . . . . 4-7
Considerations . . . . . 4-9
Summary Chart . . . . . 4-10

5 COMMBLOCK AND SUBSCHEMA RECORD DESCRIPTIONS . . . . . 5-1

COMMBLOCK for COBOL Programs . . . . . 5-1
Subschema Records for COBOL Programs . . . . . 5-22

6 INITIALIZATION . . . . . 6-1

START S2K . . . . . 6-1
OPEN . . . . . 6-7

7 WHERE-CLAUSES . . . . . 7-1

Fixed Where-Clauses . . . . . 7-2
DYNAMIC Where-Clauses . . . . . 7-35

8 RETRIEVALS . . . . . 8-1

GET1 . . . . . 8-5
GETA . . . . . 8-47
GETD . . . . . 8-54
LOCATE . . . . . 8-81
ORDER BY . . . . . 8-92
GET . . . . . 8-100
Inclusion List . . . . . 8-121
USING 0 Option . . . . . 8-131
HOLD Option . . . . . 8-131
DROP HOLD . . . . . 8-144
COPYS and COPYL . . . . . 8-147
GETP . . . . . 8-151
The LINK Command . . . . . 8-163
LINK Processing . . . . . 8-166

9 EXECUTION MODES . . . . . 9-1

Immediate Mode . . . . . 9-1
QUEUE and TERMINATE . . . . . 9-2
LOAD and TERMINATE . . . . . 9-5
CANCEL QUEUE . . . . . 9-7

10 UPDATES . . . . . 10-1

Introduction . . . . . 10-2
REMOVE . . . . . 10-2
MODIFY . . . . . 10-11
REMOVE TREE . . . . . 10-21
INSERT: General Information for All Modes . . . . . 10-28
INSERT in Immediate Mode . . . . . 10-31
INSERT in Queue Mode . . . . . 10-52
INSERT in Load Mode . . . . . 10-66
MOVE TREE . . . . . 10-75
CLEAR Commands: Moving Updated Pages to Disk . . . . . 10-81

11 SAVING AND RESTORING . . . . . 11-1

SAVE . . . . . 11-3
RESTORE . . . . . 11-8
KEEP . . . . . 11-12
APPLY . . . . . 11-16
SUSPEND . . . . . 11-22
RELEASE . . . . . 11-25
ENABLE ROLLBACK and DISABLE ROLLBACK . . . . . 11-27
FRAME and END FRAME . . . . . 11-29
COMMIT . . . . . 11-42
ROLLBACK . . . . . 11-45
RESET ROLLBACK . . . . . 11-48

12 FOR, $FOR, AND $CFOR . . . . . 12-1

Introduction . . . . . 12-1
Converting from the FOR Command to the $FOR and $CFOR Directives . . . . 12-2
FOR Processing . . . . . 12-5
The $FOR Directive . . . . . 12-11
The $CFOR directive . . . . . 12-17

13 ENABLE/DISABLE VALUES PADDING . . . . . 13-1

Introduction . . . . . 13-1

14 FULL PASSES . . . . . 14-1

Introduction . . . . . 14-1

15 $TIME . . . . . 15-1

16 ENABLE ROUTINE AND DISABLE ROUTINE . . . . . 16-1

Introduction . . . . . 16-1

17 TERMINATION . . . . . 17-1

CLOSE . . . . . 17-1
STOP S2K . . . . . 17-4

18 YEARCUTOFF . . . . . 18-1

Introduction . . . . . 18-1

19 COBOL PLEX PROCESSING . . . . . 19-1

COBOL PLEX Processor Directives . . . . . 19-2
COBOL PLEX Processor Execution Options . . . . . 19-10

20 SAMPLE COBOL PLEX PROGRAMS . . . . . 20-1

Generalized COBOL PLEX Program . . . . . 20-2
Sample Program FINDORG . . . . . 20-3
Job Control Language (JCL) . . . . . 20-25

21 POPULATING A DATABASE . . . . . 21-1

Procedures for Loading . . . . . 21-1
Loading Techniques . . . . . 21-6

Error Handling During Loading . . . . . 21-6
Recovery During the Loading Process . . . . . 21-8
Testing the Load Program . . . . . 21-8

22 USAGE AND GUIDELINES . . . . . 22-1

Efficiency Considerations in PLEX . . . . . 22-2
Execution-Time PLEX Exits . . . . . 22-7
CICS COBOL II Command-Level Linkage Requirements . . . . . 22-7
Pseudo-Conversational CICS Transactions . . . . . 22-9
Programming Facility Messages . . . . . 22-13

## APPENDICES

APPENDIX A: THE NO-SORT CAPABILITY . . . . . A-1

APPENDIX B: PL/I PLEX . . . . . B-1

Format of PLEX Commands . . . . . B-3
SYSTEM 2000 Item Types and PL/I Attributes . . . . . B-7
COMMBLOCK Declarations . . . . . B-19
SUBSCHEMA RECORD Declarations . . . . . B-31
The START S2K Command . . . . . B-43
Inclusion Lists for Retrievals and Updates . . . . . B-45
Inclusion List in LINK Commands . . . . . B-55
The FOR Command . . . . . B-55
PL/I PLEX Processor Directives . . . . . B-57
PL/I PLEX Processor Execution Options . . . . . B-71
Sample PL/I PLEX Program . . . . . B-83
Job Control Language (JCL) . . . . . B-91

APPENDIX C: FORTRAN PLEX . . . . . C-1

Format of PLEX Commands . . . . . C-3
SYSTEM 2000 Item Types and FORTRAN Variable Types . . . . . C-7
COMMBLOCK Declarations . . . . . C-11
Subschema Record Declarations . . . . . C-22
FORTRAN PLEX Processor Directives . . . . . C-34
FORTRAN PLEX Processor Execution Options . . . . . C-39
Sample FORTRAN PLEX Program . . . . . C-51
Job Control Language (JCL) . . . . . C-63

APPENDIX D: ASSEMBLER PLEX . . . . . D-1

Format of PLEX Commands . . . . . D-2
SYSTEM 2000 Item Types and Assembler Data Forms . . . . . D-3
COMMBLOCK Declarations . . . . . D-7
SUBSCHEMA RECORD Declarations . . . . . D-9
Inclusion List . . . . . D-13
USING 0 Option . . . . . D-24
PLEX Processor Directives . . . . . D-25
PLEX Processor Execution Options . . . . . D-32
Sample Assembler PLEX Programs and JCL . . . . . D-32

Index . . . . . X-1

# Illustrations


1.1 SYSTEM 2000 Facilities . . . . . 2
1.2 COBOL PLEX Program Structure . . . . . 4

2.1 The COMMBLOCK, a Subschema Record Area, and the Data Transfer Process . 8
2.2 Diagram of the Hierarchy of Clipping File Items . . . . . 11
2.3 Hierarchical Relationships in Data Trees . . . . . 12
2.4 Path and Family . . . . . 13
2.5 Data Records in a Logical Entry . . . . . 13
2.6 A Schema Tree . . . . . 14
2.7 The Hierarchical Structure of a SYSTEM 2000 Database . . . . . 15
2.8 Hierarchical Table with Pointers . . . . . 18
2.9 The Parts of a Program's Logical View . . . . . 21
2.10 Specifying a Program's Logical View . . . . . 22
2.11 An Inclusion List for Retrieval . . . . . 24
2.12 Retrieval Into the Stack and the Subschema Record Area . . . . . 25
2.13 Address Retrieval with Stack . . . . . 27
2.14 Stack Configurations with GET1 NEXT . . . . . 28
2.15 Retrieval with GETD NEXT . . . . . 30
2.16 Stack Configurations with GETD . . . . . 31
2.17 The Domain of GETD . . . . . 34
2.18 Stack Configurations with GETA . . . . . 35
2.19 Subset Selection Using AND, OR, and NOT . . . . . 37
2.20 Normalization of the First Kind . . . . . 39
2.21 Normalization of the Second Kind . . . . . 40
2.22 Normalization of the Third Kind . . . . . 41
2.23 Execution of a LOCATE Command . . . . . 42
2.24 Retrieval with GET . . . . . 44
2.25 Stack with GET NEXT . . . . . 45
2.26 Stack with GET PREVIOUS . . . . . 46
2.27 Possible Combination of Retrieval Commands . . . . . 47
2.28 Establishing Symbolic Relations with LINK Commands . . . . . 50
2.29 Linked Retrieval . . . . . 51
2.30 Preparation for an Update . . . . . 53
2.31 Using REMOVE to Delete Record Values . . . . . 55
2.32 Using MODIFY to Replace Record Values . . . . . 56
2.33 Using REMOVE TREE to Delete Parts of the Data Structure . . . . . 58
2.34 Possibilities with REMOVE TREE . . . . . 59
2.35 INSERT Placement with Respect to Parent . . . . . 60
2.36 INSERT Placement with Respect to Sibling . . . . . 60
2.37 Update Processing in the Execution Modes . . . . . 63
2.38 Clearing Updated Pages . . . . . 65
2.39 Using CLEAR and CLEAR AUTOMATICALLY . . . . . 66
2.40 Restoring and Saving a Database . . . . . 68
2.41 Database Recovery with the Update Log . . . . . 70
2.42 Two Ways to Frame a Logical Unit of Work (LUW) . . . . . 73
2.43 Example of Coordinated Recovery Processing . . . . . 75
2.44 Exclusive Use . . . . . 79
2.45 Use of a Global Hold . . . . . 82

2.46 Use of a Local Hold . . . . . 84
2.47 Job Steps Before Execution . . . . . 90
2.48 PLEX Program in a Single-User Environment . . . . . 91
2.49 PLEX Program in a Multi-User Environment . . . . . 92

4.1 Correspondence between SYSTEM 2000 Item Types and COBOL Data Description . . . . . 4-10

7.1 Key Conditions and Non-key Conditions . . . . . 7-3
7.2 Schema Record of a List of Qualified Data Records . . . . . 7-4
7.3 The NOT Operation . . . . . 7-18
7.4 Relative Position Under a Parent . . . . . 7-23
7.5 Placing of Conditions in a Where-clause . . . . . 7-33

8.1 Retrieval with an Inclusion List . . . . . 8-121
8.2 Disposition of Request for Retrievals, Holds, and Updates in Immediate Mode . 8-135
8.3 Detailed View of Multi-User . . . . . 8-136
8.4 Resolving a Local Hold Deadlock . . . . . 8-139

10.1 Effects of CLEAR Commands . . . . . 10-82

13.1 Adding a Value in the Distinct Values Table . . . . . 13-2
13.2 Distinct Values Table Padding . . . . . 13-3

15.1 SYSTEM 2000 Modules . . . . . 15-2
15.2 Opcodes for PLEX Commands . . . . . 15-4
15.3 Sample PLEX Timing Output . . . . . 15-6

19.1 Sample S2KDUM Generated by the PLEX Processor for $BLOCKS . . . . 19-3
19.2 Description of S2KDUM Control Block Items . . . . . 19-4
19.3 Sample PLXCOMM Generated by the PLEX Processor for $BLOCKS . . . 19-6

20.1 Skeleton of a PLEX Program . . . . . 20-2
20.2 Flowchart of FINDORG . . . . . 20-4
20.3 Source Statements for COBOL PLEX FINDORG Program . . . . . 20-14
20.4 Output from FINDORG Program . . . . . 20-24
20.5 Sample JCL for a Single-User COBOL PLEX Program . . . . . 20-26

B.1 SYSTEM 2000 Item Types and PL/I Attributes . . . . . B-16
B.2 Retrieval with an Inclusion List . . . . . B-46
B.3 Correspondence between the Schema Record and the SSR Description . . . B-47
B.4 Processing GET1 with an Item List . . . . . B-47
B.5 Processing REMOVE with an Item List . . . . . B-48
B.6 Processing GETD with a Designation Bit String . . . . . B-50
B.7 Processing MODIFY with a Designation Bit String . . . . . B-51
B.8 Retrieval Processing with a Designation Vector . . . . . B-53
B.9 Processing INSERT with a Designation Vector . . . . . B-54
B.10 Code Generated by the PLEX Processor for $BLOCKS . . . . . B-58
B.11 Description of S2KDUM Control Block Items . . . . . B-59
B.12 Skeleton of a PLEX Program . . . . . B-83
B.13 Source Statements for the FINDORG Program . . . . . B-85
B.14 Output from the FINDORG Program . . . . . B-90
B.15 Sample JCL for a Single-User PL/I PLEX Program . . . . . B-92

C.1 SYSTEM 2000 Item Types and FORTRAN Variable Types . . . . . C-7
C.2 Sample S2KDUM Generated by the PLEX Processor for $BLOCKS . . . . . C-34
C.3 Description of S2KDUM Control Block Items . . . . . C-35
C.4 Skeleton of a PLEX Program . . . . . C-51
C.5 Source Statements for the FINDORG Program . . . . . C-53
C.6 Output from the FINDORG Program . . . . . C-62
C.7 Sample JCL for a Single-User FORTRAN PLEX Program . . . . . C-64
D.1 Correspondence Between SYSTEM 2000 Item Types and Assembler Data Forms . . . . . D-3
D.2 Retrieval with an Inclusion List . . . . . D-14
D.3 Correspondence Between the Schema Record and the Subschema Record Description . . . . . D-16
D.4 Processing GET1 with an Item List . . . . . D-16
D.5 Processing REMOVE with an Item List . . . . . D-17
D.6 Processing GETD with a Designation Bit String . . . . . D-19
D.7 Processing MODIFY with a Designation Bit String . . . . . D-20
D.8 Retrieval Processing with a Designation Vector . . . . . D-22
D.9 Processing INSERT with a Designation Vector . . . . . D-23
D.10 Description of S2KDUM Control Block Items . . . . . D-30

# Using This Book

## Purpose

This book provides concepts, usage and reference information for the PLEX facility of SYSTEM 2000 software, Version 12. SYSTEM 2000 software is SAS Institute's data management system for mainframe computer systems running under MVS and CMS operating systems.

This book is for SYSTEM 2000 users who retrieve and update databases with COBOL, PL/I, FORTRAN, or Assembler language programs. It assumes you have Version 12 or later of SYSTEM 2000 software.

## Organization of This Book

This book has three parts: Usage Concepts, Reference, and Appendices. The following sections describe the parts and list the chapters they contain.

**Usage Concepts** The Usage Concepts part discusses program communication areas, stacks, Locate Files, and navigating through a database. It contains an introduction to retrieving and updating data using PLEX commands in a user-coded program. It also discusses administrative functions, such as saving, restoring, and recovering databases within a PLEX program.

Chapter 1, "Overview"

Chapter 2, "Getting Started"

**Reference** The Reference part describes the syntax of PLEX commands in detail and provides examples. It also discusses COBOL communications areas, subschema records, PLEX processor directives, and execution options.

Chapters 3 and 4 discuss the format of COBOL PLEX commands, SYSTEM 2000 item types, and COBOL data descriptions.

Chapters 5 and 6 describe the COBOL COMMBLOCK, subschema record areas, and PLEX initiation commands.

Chapter 7 describes the fixed PLEX where-clause and DYNAMIC where-clauses.

Chapter 8 describes PLEX retrieval commands.

Chapter 9 describes execution modes.

Chapter 10 describes PLEX update commands.

Chapters 11 through 17 describe the administrative operations, such as saving, restoring, and recovering databases, along with PLEX commands that control a PLEX environment. FOR processing of return codes is also discussed, as well as PLEX job termination commands.

Chapter 18 describes the new YEARCUTOFF command.

Chapter 19 describes the COBOL PLEX processor directives and execution options.

Chapters 20 through 22 present usage guidelines and a sample COBOL PLEX program, along with an overview of PLEX messages and return codes that can be sent to your program.

**Appendices** The appendices provide special information on various topics.

Appendix A, "The No-Sort Capability"

Appendix B, "PL/I PLEX"

Appendix C, "FORTRAN PLEX"

Appendix D, "Assembler PLEX"

## Reference Aids

The following reference aids are located at the end of the book:

| | |
|---|---|
| Index | identifies pages where specific topics and terms are discussed. |
| Foldout | provides the definition of the EMPLOYEE database and four sample logical entries. Most of the examples use the EMPLOYEE database. |

## Typographical Conventions

You will notice several type styles throughout the book. Their different purposes are summarized here.

| | |
|---|---|
| roman | is the basic type style used for most text. |
| UPPERCASE ROMAN | is used for references in the text to SYSTEM 2000 command keywords and database component names. |
| *italic* | is used for terms that are defined in the text, to emphasize important information, and for comments in sample code. |
| `MONOSPACE` | is used to show examples of commands. This book uses uppercase type for SYSTEM 2000 commands. You can enter your own commands in lowercase, uppercase, or a mixture of the two. |

## Syntax Conventions

SYSTEM 2000 command syntax uses the following syntax notation:

| | |
|---|---|
| UPPERCASE ROMAN | indicates keywords. They must be spelled exactly as given in the syntax shown. Abbreviations for keywords are also shown in uppercase. If you have more than one choice, the choices are stacked vertically with a bar to the left. Select only one. A keyword without brackets is a required keyword. |
| *italic* | indicates generic terms representing words or symbols that you supply. If the term is singular, supply one instance of the term. If the term is plural, you can supply a list of terms, separated by commas. If you have more than one choice, the choices are stacked vertically with a bar to the left. Select only one. A term without brackets is a required term. |
| [ ] | enclose an optional portion of syntax. The brackets are not part of the command syntax. Multiple choices are stacked vertically with a bar to the left. Select only one. |
| \| (vertical bar) | in syntax, means to select one of the vertically stacked choices. If the choices are in brackets, the choice is optional; if not, a choice is required. Then continue to the next portion of syntax, shown on the top line of the command syntax.<br><br>In margins, indicates a technical change in the text for Release 12.1 of the software. |
| . . . (ellipsis)<br>.<br>. | indicates that the previous syntax can be repeated. In examples, either vertical or horizontal ellipses indicates an omitted portion of output or a command. |
| *b* | indicates a significant blank in syntax or output. Generally, spaces indicate blanks, and the *b* is used only for emphasis. |
| * (asterisk) | is the default system separator throughout this book. |
| END | is the default entry terminator word throughout this book. |

Note that symbols within syntax (such as parentheses, asterisks, or commas) are required unless enclosed in brackets or specifically noted as optional.

## Example Conventions

The examples show you how to use the commands. Most of the examples use the EMPLOYEE database. Comments appear in italics and within parentheses. The manual also contains sample skeleton PLEX programs and a complete COBOL, PL/I, and FORTRAN program. The delivery tape contains sample Assembler code.

## Page Numbering Conventions

Page numbering in the Usage Concepts part is different from that in the Reference part.

- Pages in the Usage Concepts part are numbered sequentially, that is, 1, 2, 3, and so on.

- Pages in the Reference part are numbered sequentially within the chapter number, that is, 8-1, 8-2, and so on.

Both page numbering conventions are reflected in the index. For example, the index entry for the GET command might list page numbers 43 and 8-100. This means that usage information for the GET command is located on page 43 of the Usage Concepts part and reference information starts in the Reference part on the page 100 of Chapter 8.

## Additional Documentation

There are many SYSTEM 2000 publications available. To receive a free *Publications Catalog*, write to the following address:

SAS Institute Inc.
Book Sales Department
SAS Campus Drive
Cary, NC 27513

The books listed below should help you find answers to questions you may have about SYSTEM 2000 software.

- *SYSTEM 2000 Introductory Guide for SAS Software Users, Version 6, First Edition* (order #A55010) provides introductory information for SYSTEM 2000 software. It also explains how to use the SAS System with SYSTEM 2000 databases and provides examples.

- *SAS/ACCESS Interface to SYSTEM 2000 Data Management Software: Usage and Reference, Version 6, First Edition* (order #A56064) documents the ACCESS procedure, the DBLOAD procedure, and the QUEST procedure for the SAS/ACCESS interface to SYSTEM 2000 software. It explains how to create SAS/ACCESS descriptor files and how to use SAS programs with SYSTEM 2000 databases. Examples are provided.

- *SYSTEM 2000 Quick Reference Guide, Version 12, First Edition* (order #A55020) contains syntax, usage rules, and examples for all DEFINE, CONTROL, QUEST, and system-wide commands. Commands are organized alphabetically within each language, and the pages are color-coded for easy reference.

- *SYSTEM 2000 DEFINE Language, Version 12, First Edition* (order #A55012) serves as a usage and reference manual for the DEFINE language. This manual explains how to create and modify a SYSTEM 2000 database definition. It includes a detailed description and examples for each DEFINE command.

- *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition* (order #A55011) provides concepts and reference material for the SYSTEM 2000 QUEST language and for system-wide commands. For example, this book documents retrieval commands, such as LIST and PRINT, the where-clause, the ordering-clause, update commands, such as INSERT TREE and CHANGE value, and so on.

- *SYSTEM 2000 CONTROL Language, Version 12, First Edition* (order #A55013) describes how to use the CONTROL language to perform administrative tasks for SYSTEM 2000 databases, such as saving and restoring databases, assigning password security, specifying input and output files, and using the Coordinated Recovery feature.

- *SYSTEM 2000 REPORT Language, Version 12, First Edition* (order #A55014) explains how to use the REPORT language to produce reports from information contained in a SYSTEM 2000 database. This manual contains detailed reference material for using REPORT commands under MVS and CMS. Also included are examples of each command, of report composition, and of actual output.

- *SYSTEM 2000 Messages and Codes, Version 12, First Edition* (order #A55015) contains all messages and codes issued by SYSTEM 2000 software. Probable causes and corrective actions accompany all error messages and warnings.

- *SYSTEM 2000 Product Support Manual, Version 12, First Edition* (order #A55016) describes how to configure SYSTEM 2000 software for single-user and Multi-User environments, how to specify SYSTEM 2000 files, and how to use user exits. Also included are descriptions of the executable modules, all execution parameters, the Accounting Log, and the Diagnostic Log.

- *SYSTEM 2000 CICS Interface, Release 11.5 under IBM OS* (order #A5511) provides supplemental details for using SYSTEM 2000 software under CICS at the macro level.

- *SYSTEM 2000 Interface to CICS (Command Level), Version 12, Second Edition* (order #A55324) provides supplemental details for using SYSTEM 2000 software under CICS (command level). It includes system requirements, execution parameters, and available options. This edition replaces the first edition (#A55018).

- *SYSTEM 2000 CMS Supplement, Version 12, First Edition* (order #A55019) contains techniques, options, and commands that pertain to SYSTEM 2000 software under CMS.

- *SYSTEM 2000 DBMS Interactive Report Writing with Genius, Release 11.5 under IBM OS and CMS* (order #A5577) explains how to use the interactive Genius facility for producing reports and listings of data from SYSTEM 2000 databases. This manual contains detailed reference material and examples for all Genius prompts and user responses under MVS and CMS.

- *SYSTEM 2000 DBMS QueX User's Guide, Release 11.5A under IBM CICS, TSO, and CMS* (order #A5563) explains how to use the menu-driven QueX facility to enter data into and retrieve data from SYSTEM 2000 databases. This manual includes a step-by-step tutorial and reference information for QueX sessions.

- *SYSTEM 2000 DBMS QueX DBA Guide, Release 11.5A under IBM CICS, TSO, and CMS* (order #A5588) explains how to build user views for QueX sessions. This manual includes instructions for building, rebuilding, and deleting user views, for defining and deleting links in user views, and for maintaining the QueX system.

- *Technical Report: S2-106 Multi-User Tuning Tools for SYSTEM 2000 Software, Release 11.6 under IBM OS and CMS* (order #A55001) describes the Tuning Tools feature for SYSTEM 2000 software. The Multi-User status console commands display details about thread, scratch pad, buffer, and queue usage. The Accounting and Diagnostic Log reports display information about system performance or about specific jobs and job types running in the system. Data are extracted from these logs into a SAS data set for use in reports, and they are summarized into historical SAS data sets for more general monthly, quarterly, or annual reports.

- *Technical Report: S2-107 XBUF Caching Feature in SYSTEM 2000 Software, Release 11.6 under IBM OS* (order #A55002) documents the Extended Buffer Manager (XBUF) feature, which provides several caching techniques for both single-user and Multi-User jobs. These techniques include simple "front-end" XBUF macros, with which you can take advantage of dual logging, statistics gathering, cache modeling, and caching of database files without having to employ the more complicated CACHE macro. Later, if you need to tailor caching for special applications, you can refer to the CACHE macro documentation also included in this report.

- *Technical Report: S2-110 QUEUE Language for SYSTEM 2000 Software, Release 11.6 under IBM OS and CMS* (order #A55009) documents all aspects of the QUEUE language. Used together with the QUEST language manual, this technical report provides complete information on using the QUEUE facility in SYSTEM 2000 software.

- *SYSTEM 2000 Software: Changes and Enhancements, Release 12.1* (order #A55046) describes the changes and enhancements to SYSTEM 2000 software for Release 12.1.

# Changes and Enhancements

## Introduction

This section lists the changes and enhancements made to the PLEX facility of SYSTEM 2000 software for Version 12. Changes specific to Release 12.1 are at the end of this section. Throughout this book, vertical bars in the margin indicate the text is new for Release 12.1. (Changes for Version 12.0 are not marked.)

This Changes and Enhancements information is intended for users who have previous experience with the PLEX facility.

## Converting Existing Databases to the Version 12 Format

All databases created with earlier releases of SYSTEM 2000 software must be rebuilt in the Version 12 format. The purpose and function of each database table have not changed. However, the internal address pointers are now longer to accommodate more data records.

## Compatibility of Existing PLEX Programs with Version 12

Existing PLEX programs do not have to be precompiled or recompiled for Version 12. For CICS PLEX programs, Version 12 offers both a macro-level and a command-level interface. However, if you use the Version 12 preprocessors, command-level code will be generated, and you must use the command-level interface.

## Dynamic Allocation of User Files

You no longer have to allocate the file for the Message File. SYSTEM 2000 software will allocate it dynamically if it does not find the S2KMSG DD statement in your JCL.

## Dynamically Allocating Database Files

SYSTEM 2000 software dynamically allocates existing database files if they are not already allocated in the JCL or CLIST. Dynamic allocation of database files means you no longer have to include the database DD statements in the Multi-User initialization step or in a single-user job.

The new PREFIX execution parameters control dynamic database file allocation. This parameter is discussed in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

Savefiles and Keepfiles can also be dynamically allocated. The Keepfile is allocated dynamically when needed if it is not already allocated. The Savefile is now a QSAM file. SAVE and RESTORE syntax contains options for specifying the data set name, volume, and so on if you do not want to use the defaults for dynamic allocation.

The Savefile and Keepfile DDnames are the first seven characters of the database name (with Xs replacing blanks) plus a suffix of S or K, respectively. The new DDnames allow several users to save and restore (and keep updates) several different databases at the same time.

In Version 12, TAPES2K and KEEPFILE are no longer honored as DDnames. Also, all update logging is indirect mode. Direct mode is no longer supported, and the keywords INDIRECT and DIRECT have been dropped from the SAVE and RESTORE command syntax.

## Dynamic Allocation of Work Files

SYSTEM 2000 software can now dynamically allocate all work files, that is, scratch pads, sort files, the S2KUSERS file, the S2KPARMS file, the S2KSNAP file, and so on if they are not already allocated in the JCL or CLIST. Eight new execution parameters are available to change the default scratch pad and sort file allocation parameters. Allocation of work files is discussed in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

## User Exits and Execution Parameters

Several new user exits and execution parameters are included in Version 12. See Chapters 5 and 6 in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

## 3390 Disk Support

Version 12 fully supports 3390 disk devices. Also, a new half-track block size of 27,972 is now available. The default disk device is 3380.

## Maximum Database Cycle Number

The maximum database cycle number is 2,147,483,647.

## CALC Keys and GETS Not Supported

CALC keys and the GETS command are no longer supported.

## More Database Components and Larger File Sizes

The maximum number of components is now 10,000. This number includes stored strings and functions, as well as schema items and schema records. The default maximum in the NEW DATA BASE IS command is still 430. (See *SYSTEM 2000 DEFINE Language, Version 12, First Edition* and *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.)

Version 12 allows a significant increase in the number of data records in a database and the storage capacity of the Data Table. In general, the maximum data record count is approximately seven times greater than earlier versions. The byte count for data is approximately three times greater.

A database can now have up to 119,304,647 data records. The byte count for the Data Table also increased substantially; the actual capacity depends on the disk type and block size. The index tables have not changed. Database files are described in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

The maximum file sizes for a single database are as follows:

File 1 10,000 components and 200 secondary passwords.

File 2 2,147,483,648 bytes (changed for release 12.1)

File 3 268,435,455 characters (unchanged).

File 4 357,195,375 maximum number of multiple occurrence pointers (unchanged).

File 5 119,304,647 data records. There is one Hierarchical Table entry for each data record. The entries are fixed length, and the entry size grew from 14 to 18 bytes. Earlier versions allowed 16,777,215 data records.

File 6 The Data Table contains the actual data for a database, collected by schema record type and structured by pointers in the Hierarchical Table (File 5). The maximum number of tracks that can be used for database File 6 is 65,535 (X'FFFF'). Thus, the maximum number of bytes for File 6 depends on the block size and disk device. For example, a 3380 disk using a block size of 23,464 (two blocks per track) could use up to 131,071 blocks or 3,075,426,480 bytes (non-numeric characters). Earlier versions allowed 1,073,741,823 bytes.

## 31-Bit Mode

Version 12, when run under MVS/XA, MVS/ESA, VM/XA, or VM/ESA, exploits the 31-bit address capabilities and provides virtual storage constraint relief by moving most program and data storage above the 16-megabyte line.

Even though your existing PLEX programs are probably 24-bit mode, they will work with Version 12 in single-user jobs or in the Multi-User environment without changes. All communications between your PLEX program and SYSTEM 2000 software are via routine S2KPLR, which ensures the correct addressing mode. Although it is not necessary, you can take advantage of 31-bit addressing by having the SYSTEM 2000 control blocks within your PLEX program reside above the 16-megabyte line, along with the rest of your code. This allows you to have more memory to define your data areas and in-memory tables.

31-bit addressing mode could affect your user-exit code. Your user-exit program is entered in the same addressing mode as SYS2K or S2KPLI. In 31-bit mode, avoid cluttering the high-order byte of a register or word that contains an address (for example, with flags) since all 31 bits are used as an address. Also, the addresses passed to your user-exit program are likely to be above the 16-megabyte line, and you must be in 31-bit mode to reference them. For details, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

## Additional Enhancements for Release 12.1

In addition to the changes and enhancements for Version 12, some enhancements particular to Release 12.1 are also contained in this book. These additions are marked with a vertical bar (|) in the margin to indicate they are particular to Release 12.1.

**$FOR Precompiler Directive** The $FOR directive was designed to replace the GO TO processing formerly available in the FOR command in SYSTEM 2000. The newer versions of the CICS interface do not support the FOR...GO TO command, and COBOL II does not allow a label address as a parameter on a CALL. $FOR automatically generates code that checks the return code from a SYSTEM 2000 command, and causes the precompiler to build blocks of information. $FOR does not generate a call into the SYSTEM 2000 interface. To take advantage of the $FOR directive, you must recompile your PLEX programs.

**$CFOR Precompiler Directive** The $CFOR directives work in much the same way as $FOR directives, but they are in effect only for the PLEX command to which they are appended. They specify an internal label to be branched to when the interface returns to the PLEX program and the $CFOR conditions are met. You must recompile your PLEX programs to take advantage of the $CFOR directive.

**YEARCUTOFF Command** The YEARCUTOFF command is provided to accommodate dates that carry into the year 2000 and beyond.

**Miscellaneous** Changes to other sections of this manual include the effects of the new CICS Interface on existing programs, and changes to linking in PL/I programs.

# USAGE CONCEPTS

# USAGE CONCEPTS

Overview

Getting Started

Chapter 1

# Overview

*SYSTEM 2000 FACILITIES 1*

*PLEX CAPABILITIES 2*

*THE PLEX PROCESSOR 3*

*PLEX COMMANDS 3*

*PROGRAM STRUCTURE 3*

*THE SCOPE OF PLEX AND THIS MANUAL 4*

## SYSTEM 2000 FACILITIES

SYSTEM 2000 Data Management Software is a general purpose database management system. SYSTEM 2000 software has two major facilities: the Self-Contained Facility (SCF) and the Programming Language Extension Facility (PLEX).

The Self-Contained Facility offers five basic languages for defining, accessing, and controlling databases. You can call any of these languages during an SCF session.

**DEFINE** The DEFINE language allows you to define and redefine a SYSTEM 2000 database.

**CONTROL** The CONTROL language allows you to perform administrative functions, such as saving and restoring databases, assigning passwords, and maintaining an Update Log or a Rollback Log.

**QUEST** The QUEST language allows you to access a database for retrievals and updates. (This language was formerly known as ACCESS.)

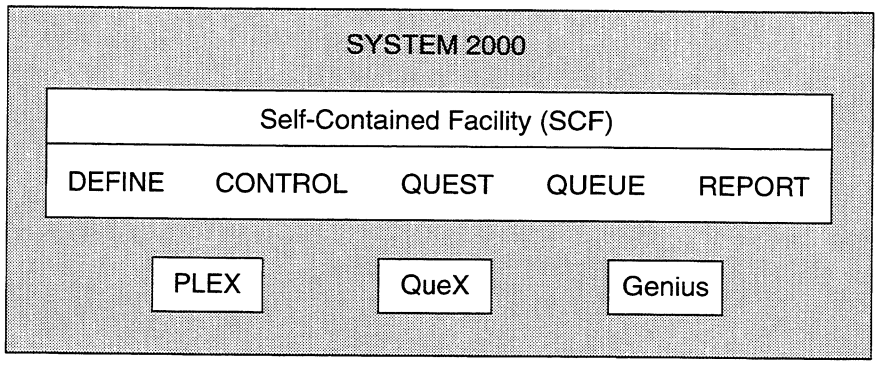
**QUEUE** The QUEUE language allows you to group updates and retrievals into batches of commands for more efficient processing of large runs.

**REPORT** The REPORT language allows you to produce reports with running subtotals, special formats, and logical pages within the physical page.

The PLEX (Programming Language Extension) Facility allows you to access and update SYSTEM 2000 databases within a COBOL, FORTRAN, PL/I, or Assembler program.

QueX and Genius are also available for accessing SYSTEM 2000 databases. QueX provides menus for viewing, modifying, or entering data in a SYSTEM 2000 database. Genius is a conversational system that prompts you for report specifications that display database information in columnar format. Illus. 1.1 on page 2 shows these facilities.

**Illus. 1.1** SYSTEM 2000 Facilities




In addition to these facilities, SYSTEM 2000 software offers many system support functions and tools for fine-tuning the execution configuration at your site. These options include multiple user access, coordinated recovery of multiple databases, accounting information, performance statistics, user-tailored configurations, and user exits.

# PLEX CAPABILITIES

With PLEX you can combine the general capabilities of a programming language, such as looping, conditional branching, flexible I/O formats, arithmetic operations, and data editing, with SYSTEM 2000 retrieval, update, and maintenance capabilities to serve a wide variety of applications. Some of the capabilities of PLEX are as follows:

- You can access multiple databases concurrently in a program.
- You can use the results of one query for subsequent queries or for updates on the same or a different database.
- You can use PLEX to produce various kinds of reports in different formats, obtaining data from SYSTEM 2000 databases or other files.
- You can establish or remove network relationships dynamically among data, even across databases.
- PLEX load mode provides an efficient method for populating databases.
- PLEX provides a tool to navigate through a SYSTEM 2000 database from one record to another. Through appropriate PLEX commands you can choose the starting point, the area of interest, and the direction of navigation.
- Programs that manipulate files for an existing application can be modified to become PLEX programs that manipulate a database.

SYSTEM 2000 software is very flexible. PLEX augments this flexibility with all of the capabilities of the host programming language.

# THE PLEX PROCESSOR

The PLEX processor translates PLEX commands (given in your program) into data descriptions, MOVE statements, and calls to the Program Service Processor with the necessary parameters. Depending on the command, the Program Service Processor invokes the required SYSTEM 2000 module at execution time.

A PLEX program is a logical organization of programming statements interspersed with commands for SYSTEM 2000 software. Since the PLEX commands are not part of the programming language, a PLEX program cannot be directly compiled. It is passed through the PLEX processor first, then expanded source code from the PLEX processor is given to the compiler as source input. Finally, the object code from the compiler is linked with the Program Service Processor and executed.

# PLEX COMMANDS

As mentioned earlier, a PLEX program contains program statements and PLEX commands. Like program statements, PLEX commands can be divided into two categories, for example, in COBOL, those appearing in the DATA DIVISION and those appearing in the PROCEDURE DIVISION.

The DATA DIVISION commands are translated into COBOL record descriptions by the PLEX processor. They establish the input/output areas for data communication between the program and SYSTEM 2000 software.

The executable commands are translated into subprogram calls; the subprogram called is the Program Service Processor of SYSTEM 2000 software. The executable commands are of different types. They can retrieve data from a database, update a database, select a subset of a database for retrieval or update, and handle database administration statistics. Other commands offer debugging tools and obtain timing statistics.

# PROGRAM STRUCTURE

A COBOL PLEX program looks similar to a COBOL program. Illus. 1.2 on page 4 shows the general structure of a COBOL PLEX program. COBOL statements and PLEX commands for data description appear in the DATA DIVISION, while executable COBOL statements and PLEX commands appear in the PROCEDURE DIVISION. The dots in the illustration represent COBOL statements. All the executable PLEX commands appear between the commands START S2K and STOP S2K. All executable PLEX commands that refer to a database are executed between the OPEN and CLOSE commands for that database.

Illus. 1.2 shows the structure of a simple program that refers to a single database, but PLEX also supports modular programming. A PLEX program can use several subprograms. They can be processed by the PLEX processor either in a batch or separately. You can pass variables declared through the PLEX data description commands from one subprogram to another.

**Illus. 1.2** COBOL PLEX Program Structure

```
DATA DIVISION.
       .
       .
(PLEX data description commands)
$BLOCKS.  (See $BLOCKS Directive on page 19-2.)
       .
       .
PROCEDURE DIVISION.
       .
       .
START S2K.
(The first executable PLEX command, which does some initializations)
       .
       .
OPEN database.
       .
       .
(PLEX commands and COBOL statements)
       .
       .
CLOSE database.
STOP S2K.
       .
       .
```

# THE SCOPE OF PLEX AND THIS MANUAL

You can execute PLEX programs in single-user jobs and in Multi-User and Multi-User/Multi-Thread environments. They can be executed in teleprocessing environments and with the CICS and TSO TP monitors. COBOL PLEX supports all IBM COBOL compilers that include standard ANSI features.

At logically predefined points within SYSTEM 2000 software, exits are included to allow you greater flexibility in performing specialized application requirements. SYSTEM 2000 software provides interfaces for Multi-User, Multi-Thread, and the Universal Software Interface (user exits). PLEX users can take advantage of user exits to handle special requirements for different applications. For a detailed description of user exits, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*

You cannot define databases with the PLEX facility. PLEX accesses databases that have already been defined via the CONTROL and DEFINE languages in the Self-Contained Facility. However, PLEX users must know and understand the definition of the databases they want to use. You do not need to know the details of the Self-Contained Facility (SCF), though occasionally the knowledge of some SCF commands helps in debugging PLEX programs.

Chapter 2

# Getting Started


*PROGRAM-DATABASE COMMUNICATION 6*
- *COMMBLOCK 7*
- *Subschema Record 7*
- *Data Independence 8*

*INITIAL STEPS 9*
- *Passwords 10*
- *START S2K 10*
- *Opening a Database 10*

*INTRODUCTION TO RETRIEVAL 11*
- *SYSTEM 2000 Record Relationships 11*
- *Physical and Logical Order of Data Records 16*
- *The Logical View of a Database 21*
- *Retrieving Data Record Addresses and Data Values 24*

*RETRIEVAL IN LOGICAL ORDER 25*
- *The Stack and Navigation through the Database 26*
  - *Multiple Stacks and COPYS 27*

*GET1 28*
- *GETD 29*
- *GETA 33*

*SUBSET RETRIEVAL 36*
- *Fixed Where-clauses 36*
- *DYNAMIC Where-Clauses 38*
- *Normalization 38*
  - *Normalization of the First Kind 39*
  - *Normalization of the Second Kind 40*
  - *Normalization of the Third Kind 41*
- *The Locate File and LOCATE Command 42*
- *ORDER BY 43*
- *Multiple Locate Files and COPYL 43*
- *GET 43*
- *GET1 with a Where-clause 48*
- *GETP 48*

*LINKED RETRIEVAL 48*
- *Symbolic Relations 48*
- *LINK 49*

*UPDATING A DATABASE 52*
- *Preparing for an Update 52*
- *Value Modification 54*
  - *REMOVE 54*

*MODIFY 54*
*Structure Modification 57*
*REMOVE TREE 57*
*INSERT 57*
*MOVE TREE 61*
*Execution Modes 61*
*Immediate Mode 62*
*Queue Mode 62*
*Load Mode 62*
*Use of Table Space 62*
*Clearing Updated Pages 64*
*The Damage Flag 65*
*The Database Cycle Number 67*

*SAVING, RESTORING, AND RECOVERING DATABASES 67*
*SAVE, RESTORE, and Update Logging 67*
*Using the Update Log 69*
*Using Coordinated Recovery 72*

*SECURITY AND INTEGRITY 77*
*Passwords 77*
*Access Authorities 77*
*Privileged Commands 77*
*Security by Entry 78*
*Exclusive and Non-exclusive Use of a Database 78*
*HOLD Option 80*

*MULTIPLE DATABASES 86*

*TERMINATION 88*
*Closing a Database 88*
*STOP S2K 88*

*PROGRAM STRUCTURE AND PROCESSING 88*
*COBOL PLEX Program Structure 88*
*PLEX Processor Directives 89*
*Job Steps 89*

# PROGRAM-DATABASE COMMUNICATION

PLEX offers a two-way transfer of information between the program and the database. Through SYSTEM 2000 software, data are transferred to the database and selected data are brought from the database, along with direct responses from SYSTEM 2000 software.

Information sent to SYSTEM 2000 software by the program specifies the database or databases to be used, the parts of the data structure the program will use, and the password for access authority. For all operations to retrieve data, the program issues a PLEX command indicating the kind of operation to be performed and the data to be accessed. For updating the database, the program specifies the data to be inserted, replaced, or removed.

In response, SYSTEM 2000 software indicates whether the PLEX command is valid and, if it is, goes to the database to retrieve the data or perform the update. Once the operation is completed, SYSTEM 2000 software relays to the program several different indications of the outcome. These include a numerical return code regarding the operation, a damage flag showing the status of the database (damaged or undamaged), and, after a successful operation, the actual data values that were requested.

The communication between a program and SYSTEM 2000 software requires two special kinds of areas in the program: a communication block, called a COMMBLOCK, and one or more transfer areas, called subschema records, to hold data values. The PLEX processor sets up a third communication area, the S2KDUM control block, in response to the $BLOCKS directive, which is required in every PLEX program. $BLOCKS and the S2KDUM control block are discussed in **$BLOCKS Directive** on page 19-2. The COMMBLOCK and subschema record descriptions are discussed in the next two sections.

## COMMBLOCK

A PLEX program requires a communication block (COMMBLOCK) for each database the program uses. Each COMMBLOCK consists of fourteen fields (and an optional fifteenth field). Each field gives information for the processing of the database.

The program provides the values for two fields. Values for the others are sent to the program by SYSTEM 2000 software. One program-supplied value is the password for database access authority. One value sent by SYSTEM 2000 software is the return code, which is issued after each command given by the program. The return code indicates the result of the command. It tells whether the command was executed and, if the command had associated data, whether the data were retrieved or updated as requested.

So the COMMBLOCK is a two-way communication area. The program places in it certain information to be used by SYSTEM 2000 software, and SYSTEM 2000 software sends return codes to it telling the status of each step of the program's handling of a database. The rest of the COMMBLOCK fields are covered in detail in **COMMBLOCK Fields** on page 5-3.

## Subschema Record

In addition to the COMMBLOCKs, the program describes subschemas for the databases it will be accessing. A subschema represents the view of the database from the perspective of the application program. It is a subset of the database schema describing the organization of the database and the kinds of data stored in the database. In PLEX, the subschema consists of a collection of subschema record (SSR) descriptions.

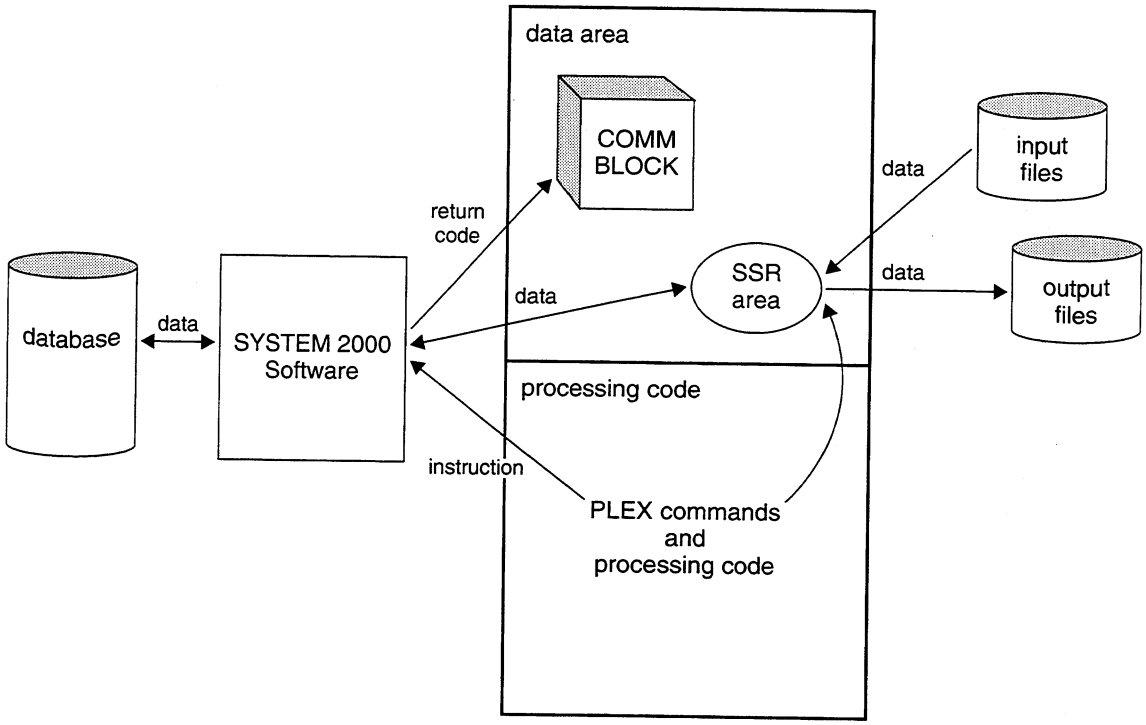
A subschema record description serves two purposes. It is a data transfer area, and it specifies a format. The data transfer area is established to hold values

- that the program reads in from an input file
- that programming statements moved to variables included in the subschema records
- that SYSTEM 2000 software retrieves from the database
- that the program writes to an output file.

Illus. 2.1 on page 8 shows both the COMMBLOCK and a subschema record area in the process of communication between the program and SYSTEM 2000 software.

Values in a SYSTEM 2000 database are stored collectively as data records and are called data values. A database can store many different kinds of records. The formatting portion of a subschema record description specifies the items of one specific schema record that will be used in processing the database. Taken together, the subschema record descriptions in a program make up that program's view of the database. They designate the parts of the database that hold the data to be retrieved, changed, removed, or inserted.

**Illus. 2.1** The COMMBLOCK, a Subschema Record Area, and the Data Transfer Process




## Data Independence

Data values are passed back and forth between an application program and a database through subschema records. SYSTEM 2000 software converts them from the representations in the database to the representations specified by the program in its subschema record descriptions. You can specify one of several options in the COBOL USAGE clause for each item in a subschema record description.

The items defined in a SYSTEM 2000 database are of nine different types: CHARACTER, TEXT, UNDEFINED, DATE, INTEGER, DECIMAL, MONEY, REAL, and DOUBLE. Character strings in which all the blanks are significant are of type TEXT. In character strings of type CHARACTER, all blanks are insignificant except single spaces between words. UNDEFINED values contain any of the 256 EBCDIC characters. Integers are given the type INTEGER and decimal numbers are of type DECIMAL. MONEY type items are decimal numbers that can be displayed with the insertion characters $ and CR. The type DATE is provided for storing dates as numbers that can be used in computations in the Self-Contained Facility. REAL and

DOUBLE items store single precision and double precision numbers.

Here is the correspondence between SYSTEM 2000 item types and COBOL usage.

| SYSTEM 2000 item types | COBOL usage |
|---|---|
| CHARACTER<br>TEXT<br>UNDEFINED | DISPLAY |
| DATE<br>INTEGER<br>DECIMAL<br>MONEY | DISPLAY or<br>COMP or<br>COMP-3 |
| REAL<br>DOUBLE | COMP-1 or<br>COMP-2 |

The flexibility to specify the usage and picture of each subschema item demonstrates the data independence possible with PLEX. An item can be defined one way in the database schema, and a PLEX program can look at subschema values for that item in a different way. For example, as the table shows, a COBOL program has three choices of usage for describing variables to hold values of an INTEGER type item.

SYSTEM 2000 software performs all the usage/type conversions necessary for the transfer of data during retrievals, updates, and comparisons. When different programs use the same database, the same items can be accessed in different forms. Therefore, you need not be concerned with the way database values are internally stored or with the access methods used.

The data independence allowed in PLEX extends to the names that can be given to items in a subschema record. Syntax discussed in **Subschema Records for COBOL Programs** on page 5-22 allows an item to appear in a subschema record with a name different from the one it was given in the database schema. This flexibility in naming subschema items is especially helpful in two situations. First, when a schema item name is not a program-compatible data name, the program can specify one that is compatible. Second, when a PLEX program uses items with identical names in different database schemas, unique names can be specified for those items.

# INITIAL STEPS

After the necessary communication and record transfer areas are described, a PLEX program

- places a valid password in one of the COMMBLOCK fields
- specifies the number of subset files the program will use
- opens a database, that is, makes the database available to the program for processing.

## Passwords

Passwords in SYSTEM 2000 software serve the typical purpose: they allow only authorized personnel to access the database. In addition, the SYSTEM 2000 security code decides, according to the authority the password carries, which parts of the database a program can use. SYSTEM 2000 passwords can have different degrees of authority, depending on how they were assigned for a particular database.

The master password holder assigns passwords and password access authorities outside PLEX, using the CONTROL language. Then the program places the password in the fifth COMMBLOCK field. You can specify the password in either of two ways. The password can appear in a MOVE statement in the program code, or it can be read into the COMMBLOCK from some outside source, to keep it secure.

The password in the COMMBLOCK is validated when the database is opened. This password applies to all the accesses to that database until either the database is closed or another database is referred to. A database can be closed and then reopened with a different password.

## START S2K

The first SYSTEM 2000 command to be executed in a program is START S2K. This command appears in the COBOL PROCEDURE DIVISION and must be executed before any retrieval or update commands are issued.

START S2K specifies the number of files (up to 16) to be used for retrieving a subset of the database. These files are called Locate Files. (See **The Locate File and LOCATE Command** on page 42 for more information about Locate Files.) The START S2K command also specifies to SYSTEM 2000 software whether the run will be in a single-user or Multi-User environment.

After the START S2K command (and after all other PLEX commands), the program should check the return code that SYSTEM 2000 software places in the COMMBLOCK. The return code indicates whether the command specifications were processed without error.

## Opening a Database

The last important step in initializing PLEX processing is opening a database. You issue an OPEN command, which specifies the name of the database. In a Multi-User environment, the command can also reserve the database for the exclusive use of the program. In response to the OPEN command, SYSTEM 2000 software checks the password in the COMMBLOCK and makes the database available if a valid password has been specified.

PLEX gives you the ability to have more than one database available at one time. A PLEX program can have up to 63 databases available at once without explicitly closing any of them, although only one database can be accessed at a time.

A program that has several databases available at the same time can refer to one database, then to a second, and then again to the first. When the program begins a sequence of commands referring to the second database, the COMMBLOCK for that database must contain a valid password. Similarly, when the program later gives a command referring again to the first database, its COMMBLOCK must contain a valid password. This password can be different from the one used in the original access of the database.

Immediately after giving the OPEN command, the program should check the return code and the damage flag (also in the COMMBLOCK) to make sure that the command was accepted and the database is undamaged, that is, suitable for both retrieval and updating. Retrievals can be processed on a damaged database; updates cannot.

# INTRODUCTION TO RETRIEVAL

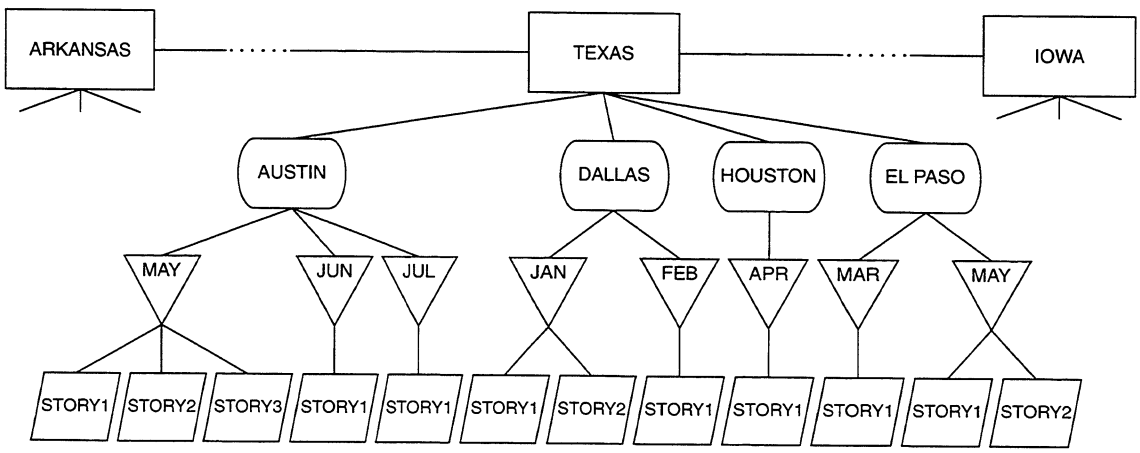
This section discusses the highlights of database structure, the physical and logical arrangement of data records, the PLEX program's logical view of a database, and the process of retrieval.

## SYSTEM 2000 Record Relationships

In addition to holding data values, a SYSTEM 2000 database has a built-in, logical structure of relationships among the data records. The user specifies this logical structure in the database schema.

The main data structure is hierarchical. The records can be compared to the folders in a file, for instance, a newspaper clipping file. If each file drawer contains folders on one subject, for example, a state, each folder in the drawer contains packets of articles on one city in that state. Each packet contains clippings about events that happened in one month. Illus. 2.2 shows these relationships as hierarchical data trees.

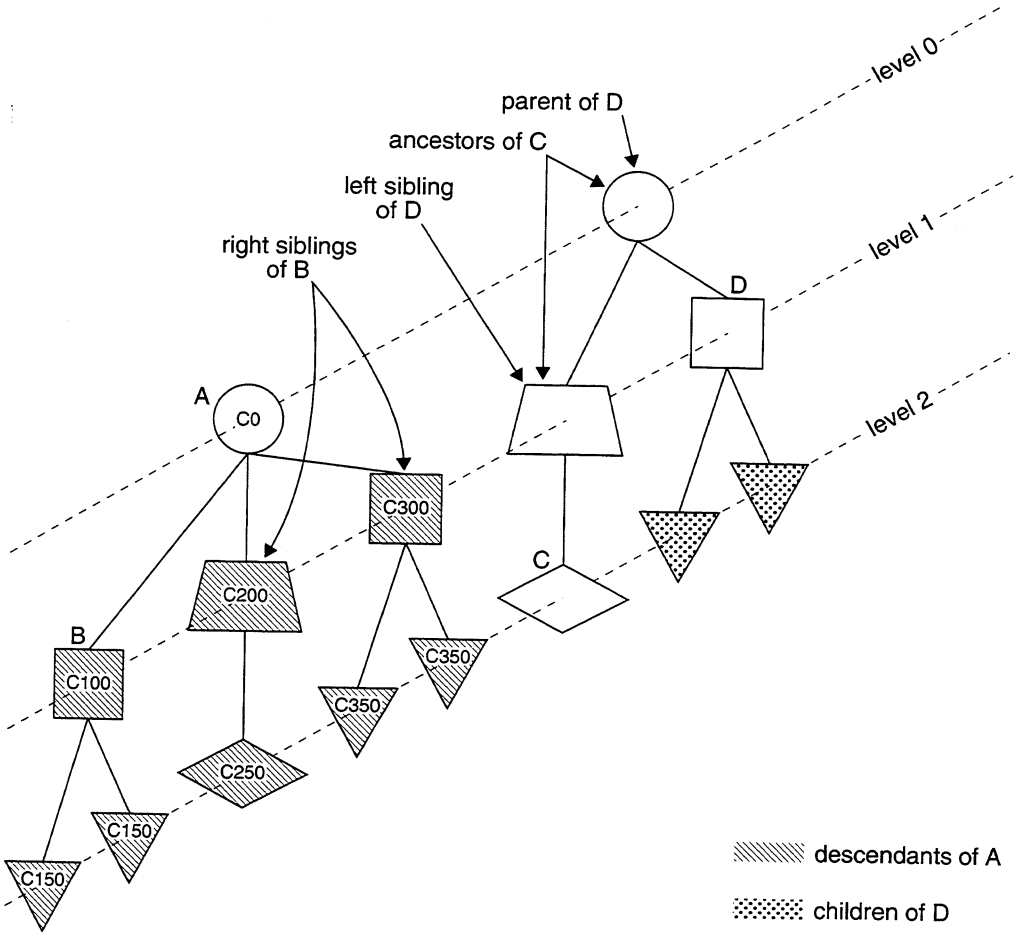
**Illus. 2.2** Diagram of the Hierarchy of Clipping File Items



Four levels are apparent in this hierarchy, starting with level 0 at the top and going down to level 3.

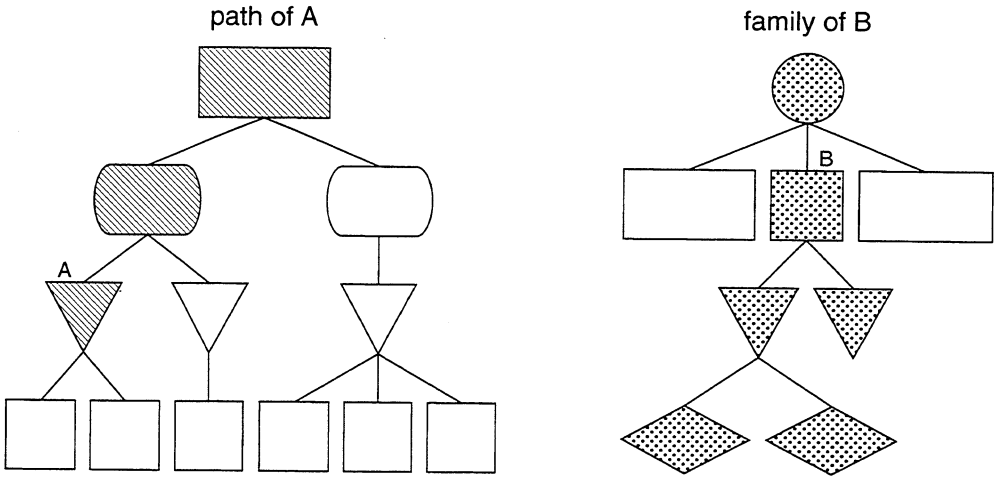
Any node in the tree at levels 0 through 2 is an ancestor of the related nodes below it. Considering the node marked AUSTIN as an ancestor, the nodes marked MAY, JUNE, and JULY, as well as those marked STORY1, STORY2, and so on, are all descendants of AUSTIN. The ancestor at the level immediately above a node is the parent of that node. Similarly, a descendant at the level immediately below a node is a child of that node. The nodes at level 1 of the tree are siblings of each other, and the same applies to nodes with the same parent at levels 2 and 3. Illus. 2.3 shows these relationships.

**Illus. 2.3** Hierarchical Relationships in Data Trees




The set consisting of a node and all of its ancestors is a path. The family of a node comprises the node, all of its ancestors, and all of its descendants. Illus. 2.4 on page 13 shows a path and a family.

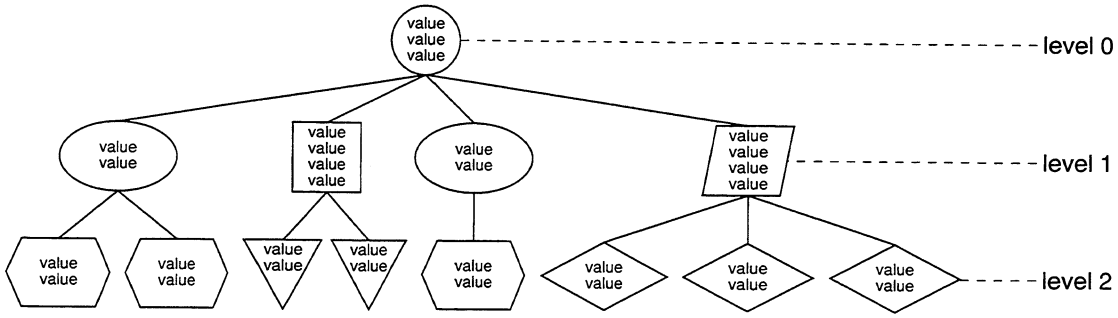
**Illus. 2.4** Path and Family




The data trees illustrated so far show one piece of information at each node. A node can actually contain several pieces of information, several data values. That is, data values at each node are grouped in a data record.

Each group of logically related data records is called a *logical entry*. For example, in the EMPLOYEE database, each logical entry contains data about one employee. The tree in Illus. 2.5 has three kinds of data records at level 1. Each node (shape) represents a data record. Notice that a logical entry, like the one here, can have different kinds of data records at the same level.

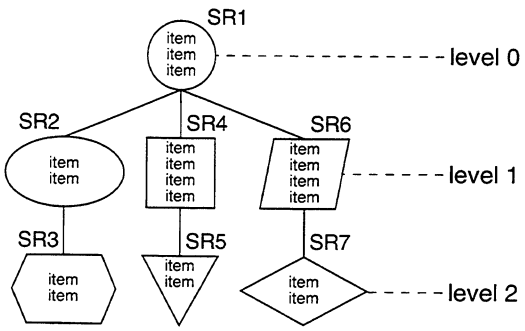
**Illus. 2.5** Data Records in a Logical Entry



The database schema serves as a pattern for creating and interpreting logical entries. A schema consists of schema records and schema items, specifying characteristics of the data and relationships among records. A schema tree graphically represents the database schema, which you construct with the DEFINE language. A data tree shows relationships among the data records, which contain the specific data values.

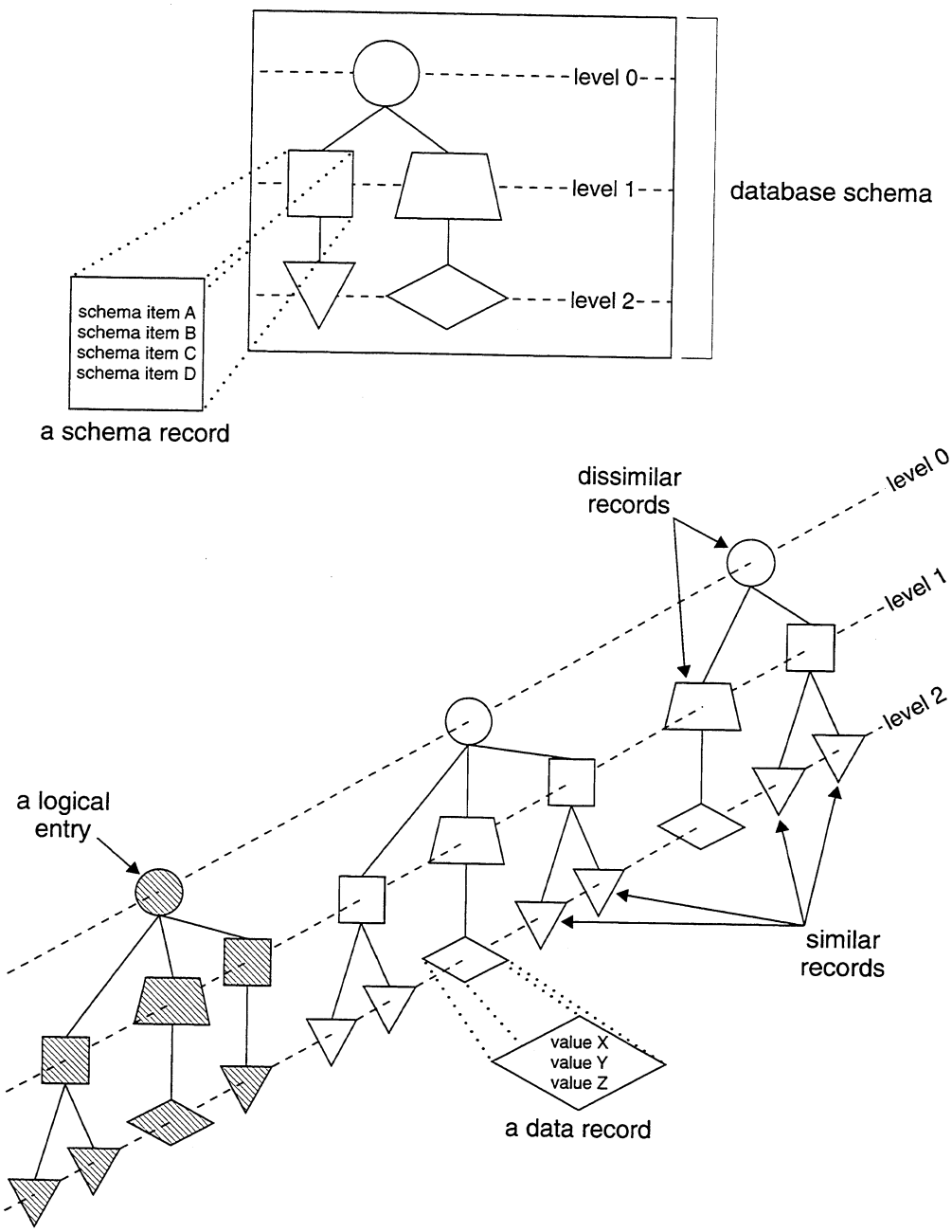
For the tree of data records in Illus. 2.5 on page 13, the corresponding schema tree, made up of schema records, is shown in Illus. 2.6.

**Illus. 2.6** A Schema Tree

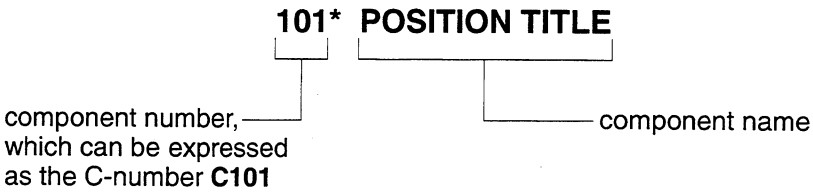



Each schema record represents one type of record, so each schema record appears only once in the schema tree. In a data tree, however, several data records can belong to a single schema record. These are called similar data records. That is, each data record of a certain type is an occurrence of the schema record of that type. In general, the correspondence between data trees and a schema tree is shown in Illus. 2.7 on page 15.

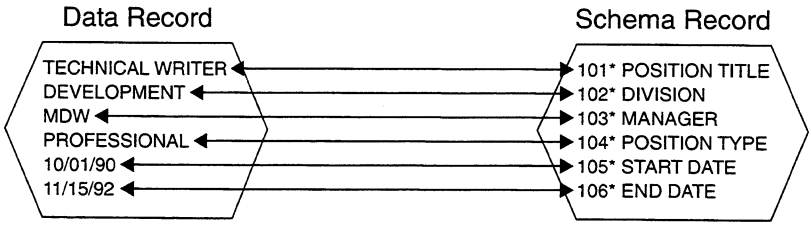
**Illus. 2.7** The Hierarchical Structure of a SYSTEM 2000 Database



Within a data record, the data values correspond to schema items of the schema record (SR). The master password holder gives a component name and a component number to each schema record component when defining the database.




The correspondence between the values in a data record and the items of a schema record is illustrated next.



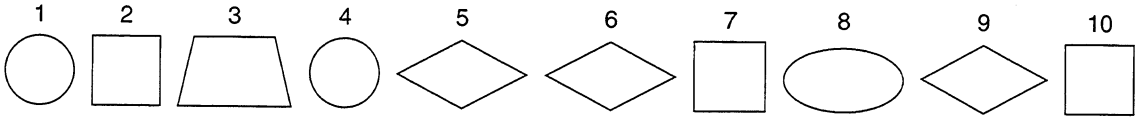

Thus far, this discussion has explained how the data in a database are logically structured, that is, how individual data values are grouped into data records that are related to each other in whatever hierarchy was defined for the database. But what does SYSTEM 2000 software use to carry out this logical structuring? And how does the user's logical data structure relate to the actual storage of data in computer memory? The next section covers some of the techniques SYSTEM 2000 software uses to maintain a database in a user-defined logical structure, which is independent of the physical storage order of data values.

## Physical and Logical Order of Data Records

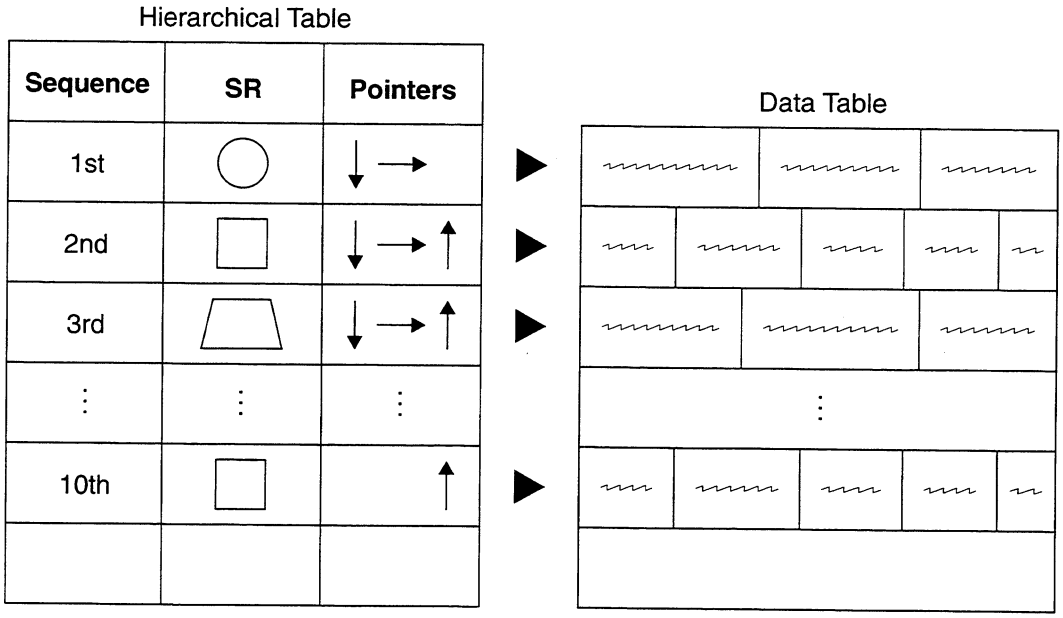
In order to perform retrieval operations, SYSTEM 2000 software keeps track of where each data record is stored and how the data records are related to one another. To do this, the software maintains the Hierarchical Table (HT) and the Data Table (DT), two of the seven database tables. See the *SYSTEM 2000 Product Support Manual, Version 12, First Edition* for details about database tables.

The Hierarchical Table contains data record pointers that indicate the logical relationships among data records in the database. The Data Table holds the actual values of all the data records. The order in which sets of pointers for data records are stored in the HT is not the logical order of data; it is rather the physical order of data storage. This order parallels the sequence of data records in the DT.

Suppose various data records appear in physical storage sequence as shown here.

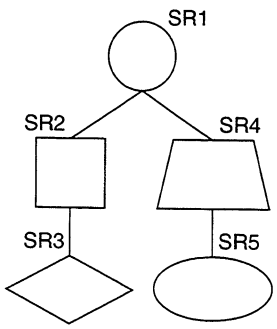



The HT and the DT are physically assembled as follows.




The HT contains a set of pointers for each data record, showing its place in the logical structure. These pointers tell which other data record is the first descendant, which is the next sibling, and which is the immediate ancestor, in logical order. The immediate ancestor pointer for a record at level 0 points to the preceding level 0 record in logical order. The following illustrations show the assignment of Hierarchical Table pointers.

How do the HT pointers relate to the database schema? A schema tree could be defined as shown here. Level 1 has two types of records, and level 2 has two types of records, one for each level 1 schema record.



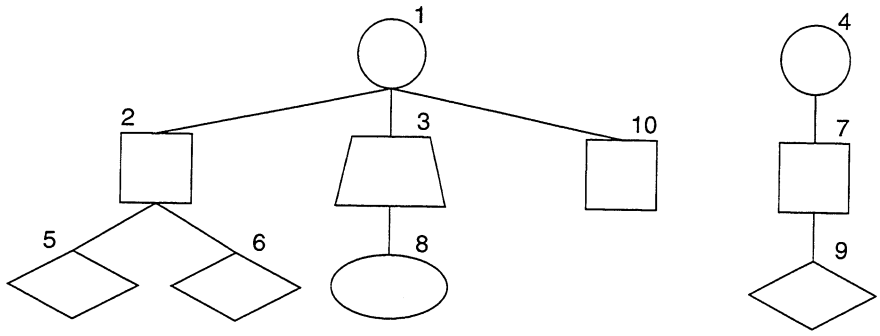
The ten data records shown previously could have been entered in this database such that the resulting HT pointers would look like those in Illus. 2.8.

**Illus. 2.8** Hierarchical Table with Pointers

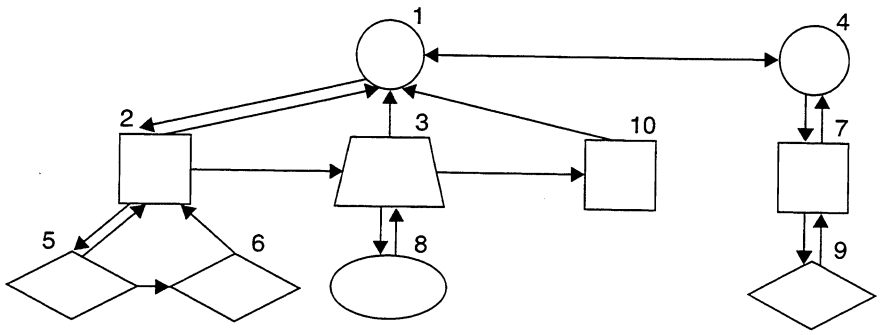
Hierarchical Table

| SR and Sequence | POINTERS |
|---|---|
| 1 ○ | ↓2 →4 |
| 2 □ | ↓5 →3 ↑1 |
| 3 ⏢ | ↓8 →10 ↑1 |
| 4 ○ | ↓7 ↑1 |
| 5 ◇ | →6 ↑2 |
| 6 ◇ | ↑2 |
| 7 □ | ↓9 ↑4 |
| 8 ⬭ | ↑3 |
| 9 ◇ | ↑7 |
| 10 □ | ↑1 |

The corresponding logical structure of the data are shown by the data trees below.

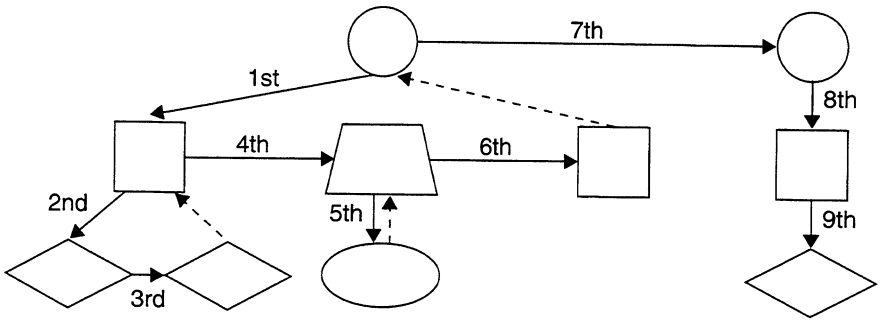



Here are the HT pointers for this data tree.



Thus, the physical order of data storage might not be the same as the logical order among data records. You usually are concerned with the logical order of the data, which is true in PLEX, since retrieval commands traverse the logical structure.

How does SYSTEM 2000 software move in logical order through data trees? Movement is top-to-bottom and left-to-right, as illustrated below.




To retrieve data, you specify which records in the logical data structure are to be selected. You can also specify which items should be retrieved from the selected record(s). In a PLEX program, the subschema record descriptions specify the types of schema records and the items of interest, that is, the program's view of the database. In the next section this topic is taken up again, to show how it relates directly to the logical structure of the database.

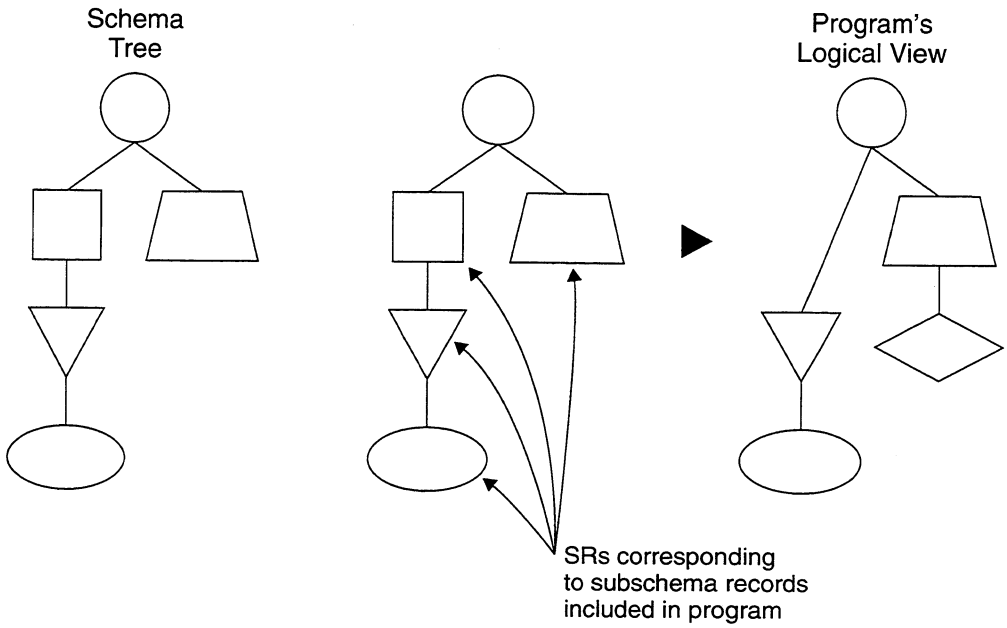
## The Logical View of a Database

In a PLEX program, the logical view of a database is reflected in its subschema record (SSR) descriptions for that database. This logical view can encompass the entire logical structure of the database, or it can focus on specific parts.

Besides setting up a record transfer area, a subschema record description specifies the type of data records the program will access. In other words, a subschema record description corresponds to a schema record in the database schema.

Illus. 2.9 shows how a program's SSRs select the parts of the schema structure to be used.

**Illus. 2.9** The Parts of a Program's Logical View

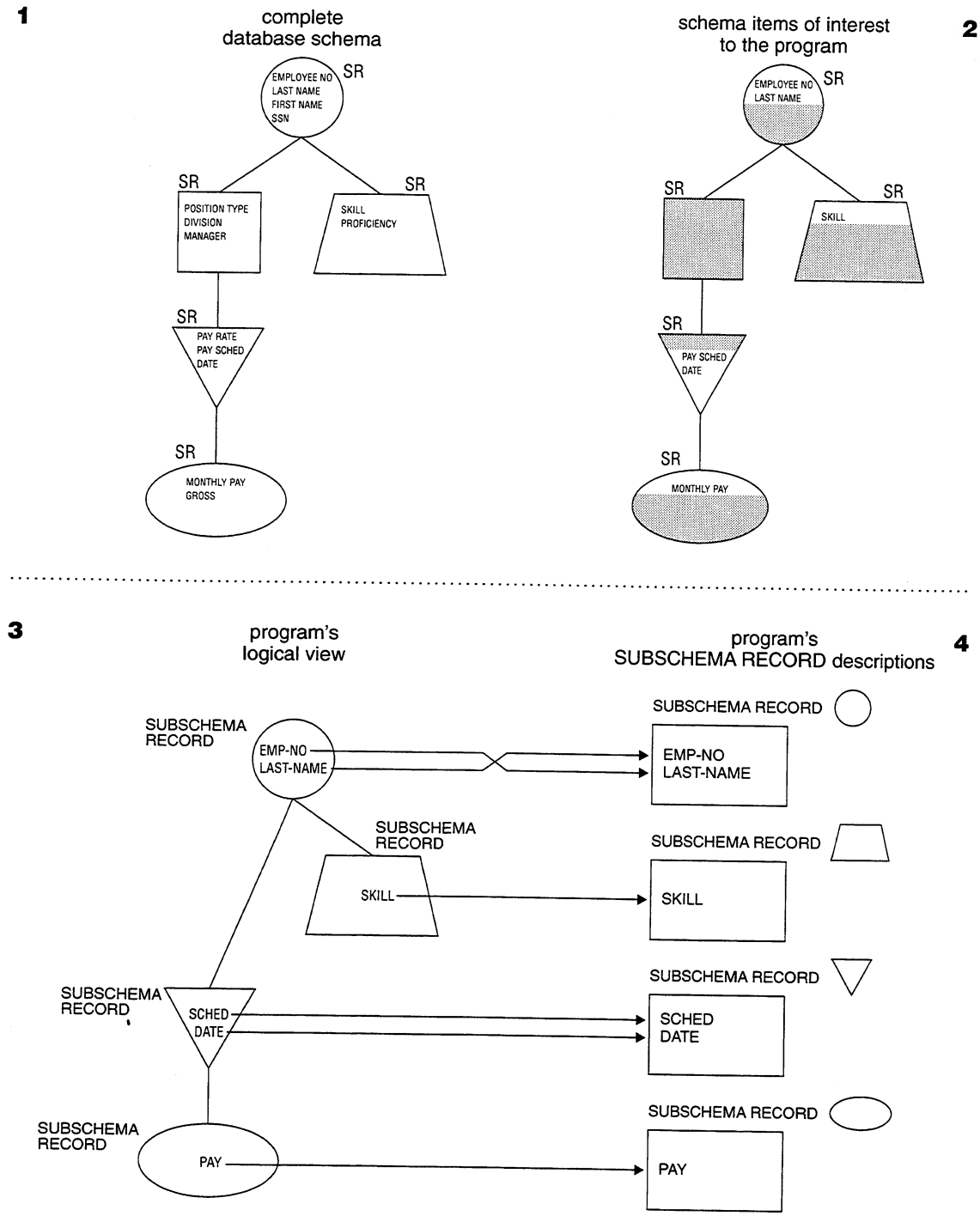



The actual schema of the database is not changed by the subschema record descriptions. The program simply shows SYSTEM 2000 software that data are to be retrieved or changed only in the specified parts of the database.

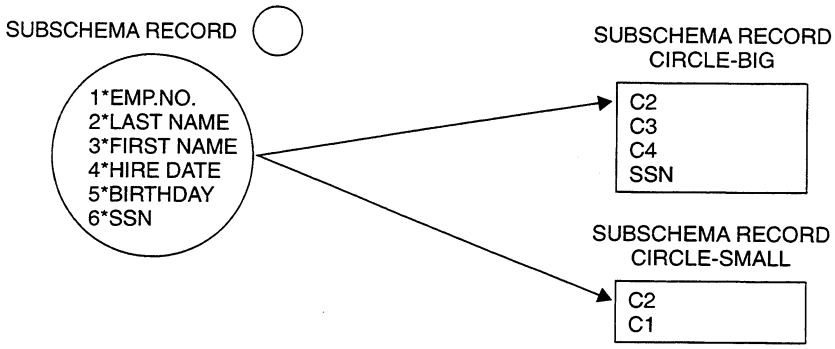
Subschema record descriptions also specify which items the program can use. Some or all items can be declared for a schema record. In addition, the program can give each subschema item any COBOL-compatible name and can choose from a variety of valid COBOL usages and pictures. You can list subschema items in any order, regardless of their order in the database schema. See the appropriate appendix for FORTRAN, PL/I, and Assembler subschema record specifications.

Illus. 2.10 on page 22 shows the schema records the program will access and the items within those schema records that it will examine.

**Illus. 2.10** Specifying a Program's Logical View



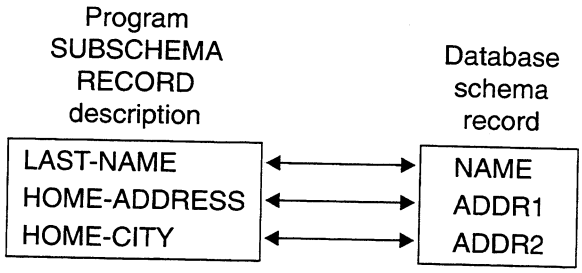
You can include more than one subschema record for a given schema record. For example, perhaps a program uses four items of a schema record for one purpose but needs only two of those items for another. In this situation, the program could include for schema record CIRCLE one subschema record with four items in it called CIRCLE-BIG, and a smaller subschema record with only two items, called CIRCLE-SMALL.



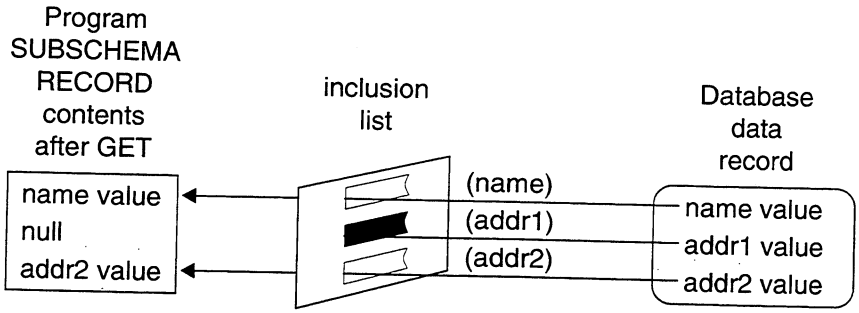

Finally, a PLEX program can use an inclusion list with a retrieval or update command to specify items in a subschema record that are to be affected in an operation. There are several types of inclusion lists. Illus. 2.11 on page 24 shows how an inclusion list works, specifying in this case only two of the three subschema items for a retrieval command.

**Illus. 2.11** An Inclusion List for Retrieval

Given the following correspondence:




PLEX retrieval command GET executes:




The primary consideration throughout all the variations allowed for SSRs is that each SSR description corresponds to only one schema record. That is, a subschema record description lists variables for items of a single schema record and not for any other schema record. Also, the program must have password authority for the items referred to by an SSR.

## Retrieving Data Record Addresses and Data Values

PLEX retrievals involve locations of the data records specified and data values for the subschema items requested.

A location (address) indicates a data record's place in the physical order of data records shown in the Hierarchical Table. (In most of the data trees shown in this book, addresses of the data records appear near the corresponding shapes, usually at the top right corner.) When SYSTEM 2000 software retrieves a data record, it places the record's address in an internal work area, called a stack. Both the software and the program use the stack to keep track of where retrievals take place in the database. The stack is discussed in detail in the next section.

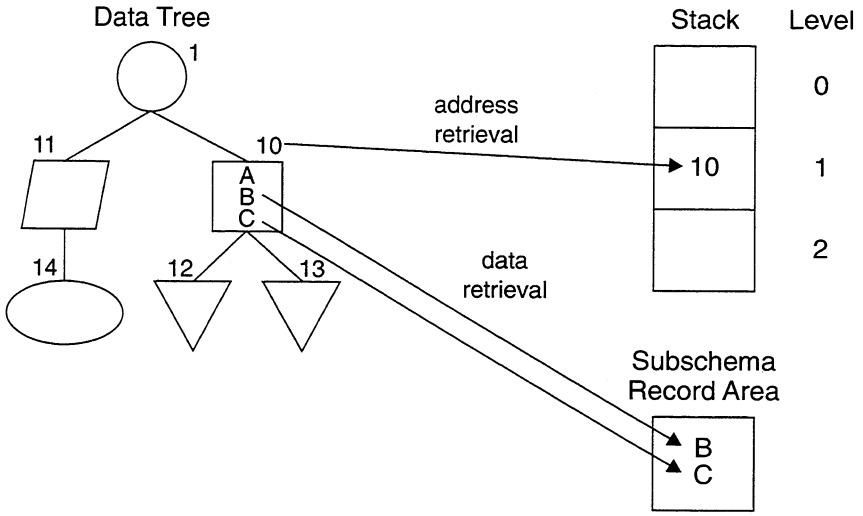
The values from a data record are placed in the subschema record area specified in the retrieval command being processed.

The retrieval process is shown in Illus. 2.12 on page 25.

**Note**: Your program can avoid part of the retrieval process when necessary. As a preliminary step for an update operation, the program retrieves a data record in order to show SYSTEM 2000 software where in the logical data structure to perform the update. Depending on the nature of the update, the program might not need to retrieve any values

from that record. In this situation, the program can specify USING 0 in a retrieval command, causing retrieval of only the record address, not data values.

**Illus. 2.12** Retrieval Into the Stack and the Subschema Record Area




# RETRIEVAL IN LOGICAL ORDER

As discussed previously, SYSTEM 2000 software processes data records in two different ways in retrieval, keeping track of both their physical and their logical order. This discussion introduces PLEX commands that retrieve data records in logical order and shows you how PLEX navigates through the database along a path, obtaining descendants, ancestors, or siblings of data records.

## The Stack and Navigation through the Database

Retrieval of a data record is actually two retrievals. The desired data values are retrieved and placed in a subschema record in your program. Also, the address of the data record is retrieved and placed in an internal area called the stack.

The stack in use for a given command consists of 32 levels and can contain up to 32 entries, one at each level. There is a one-to-one correspondence between the levels of the stack and the levels of the database. If a data record is at level 0 in the database, its address is placed at level 0 of the stack. An entry in the stack has no connection with a logical entry in the database.

A PLEX program uses the stack to control navigation through the database. First, it determines a starting point. The software enters a starting point in the stack when you issue a GET, GETP, or GET1 command. The starting point is called a primary stack entry. The stack can contain only one primary entry at a time.

A primary entry clears all previous entries from the stack. After a primary entry is made, the entire family of that data record becomes available to the program for retrievals. That is, you can retrieve any ancestor or descendant of the primary entry with appropriate GETA and GETD commands. Addresses retrieved by these commands become secondary stack entries.

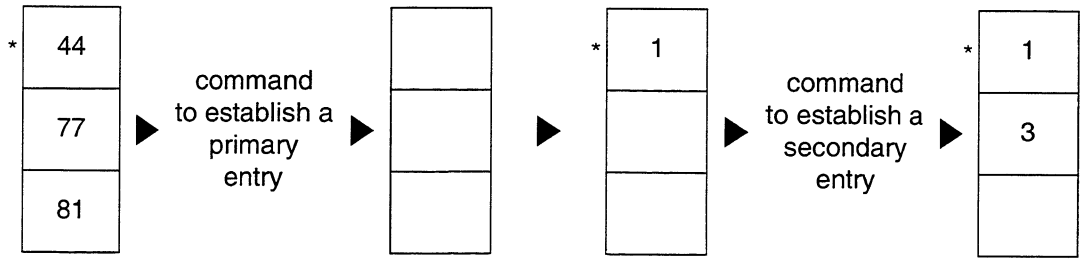
At any time there can be more than one secondary entry in the stack, but you can add secondary entries in the stack only if a primary entry already exists.

As mentioned, the family of a data record consists of itself, all its ancestors, and all its descendants. The path of a data record is a subset of the family; it consists of the data record and its ancestors only. The primary stack entry determines a family of the data records available for future retrievals. Secondary stack entries determine a path of data records in that family.

As SYSTEM 2000 software performs each retrieval, it maintains the stack, ensuring that the stack entries are all from the same path. If an ancestor of the data record for the primary is retrieved, the stack entries still reflect the path of the data record for the lowest-level entry already in the stack. But if a descendant of the data record for the primary is retrieved, the stack entries reflect the path of this descendant data record. In this situation, previous stack entries for data records not in the new path are cleared. Thus, the stack represents the most recently chosen path.

Illus. 2.13 on page 27 shows establishing primary and secondary entries in the stack.

**Illus. 2.13** Address Retrieval with Stack




**Note**: The asterisk identifies the primary entry.

### Multiple Stacks and COPYS

So far this discussion has involved only one stack. A PLEX program can establish sixteen different stacks; they are referenced by numbers 0 through 15. (The Locate File for a given number is always associated with the same database as the stack referenced by that number.) A program can use more than one stack at a time, which is useful when retrieval from one path depends on retrieval from another path. Bill of Materials processing is an example of such an application.

When a program uses several stacks, SYSTEM 2000 software keeps track of entries in all of them. Stack 0 always contains 32 levels, even when it is not being used in a command. Entries at all 32 levels of stack 0 are preserved for later reference. For stacks 1 through 15, stack entries are handled differently. For example, while a program is using stack 8 in a command, that stack contains 32 levels. But when the program refers to some other stack, SYSTEM 2000 software preserves 10 levels of stack 8, starting with the uppermost level that has an entry. If the program later refers again to stack 8, the stack expands to 32 levels, including the 10 levels that were preserved.

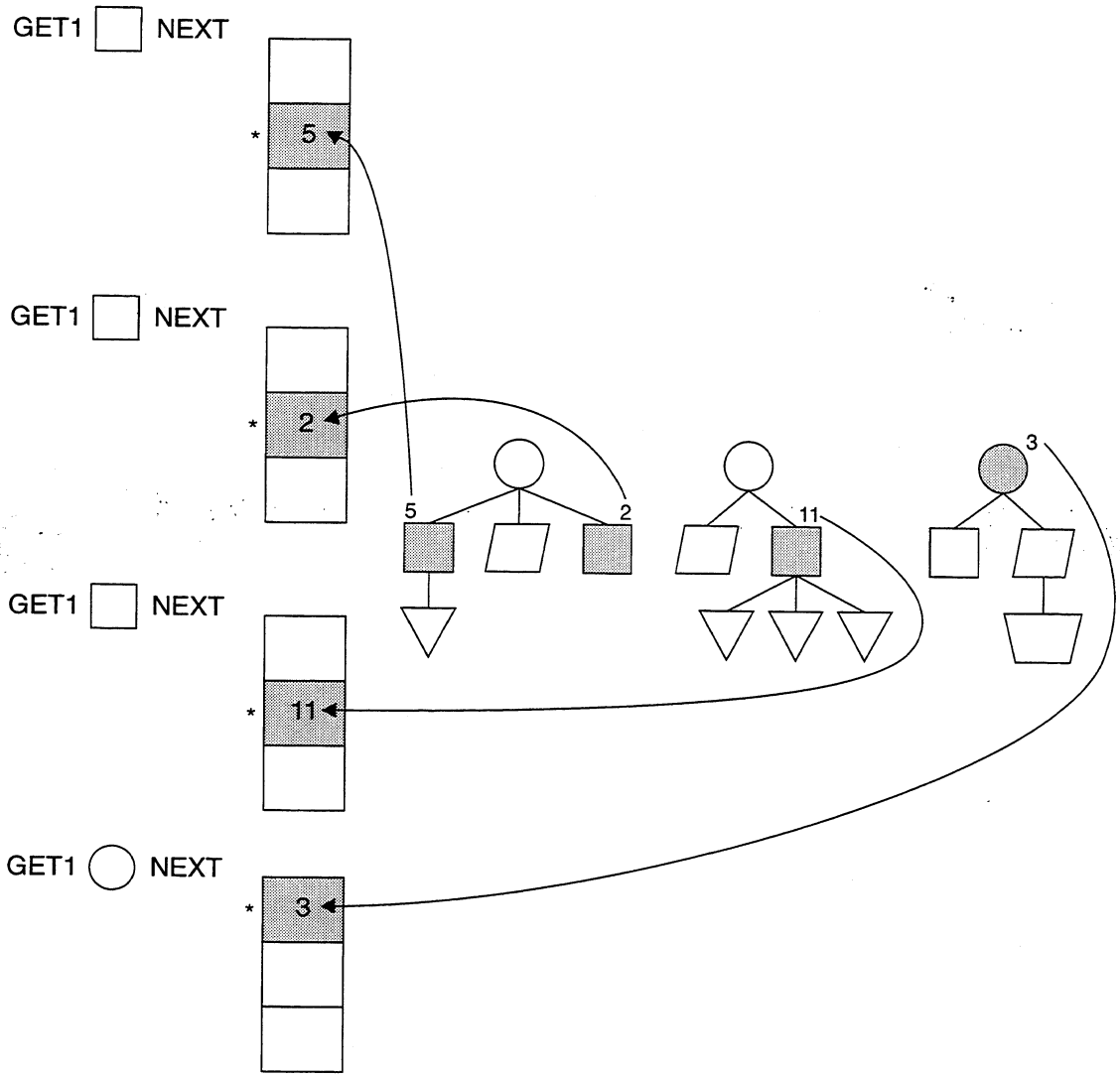
A program can make a copy of a stack, using the COPYS command. In this command, the program specifies both the stack to be copied and the stack that will become the copy. As a result, two stacks, the original and the copy, contain the same entries.

As a program uses a stack to navigate from data record to data record, the stack entries change accordingly, to indicate the most recently retrieved record at each level of the data tree.

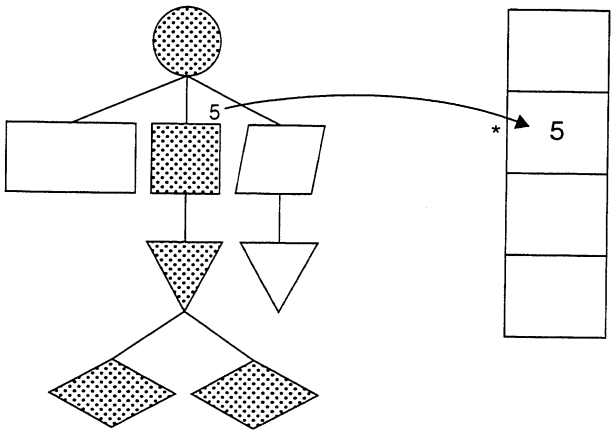
# GET1

The GET1 command gets one data record from the database. You can use GET1 to move logically through the data trees in the database. GET1 NEXT (in a repeated loop) retrieves data records for one type of schema record, one by one in logical order. NEXT means the next data record in logical order. The GET1 command places the data in a specified subschema record. It also puts the address of the data record in the stack referenced by the program. All previous entries are cleared from the stack, and the retrieved address becomes the primary entry. Illus. 2.14 shows the changes in the contents of the stack and shows how data records are retrieved in logical order with the GET1 command.

**Illus. 2.14** Stack Configurations with GET1 NEXT



Since GET1 establishes a primary entry in the stack, it sets up the possibility of retrieving ancestors and descendants using GETA and GETD commands, as shown below.



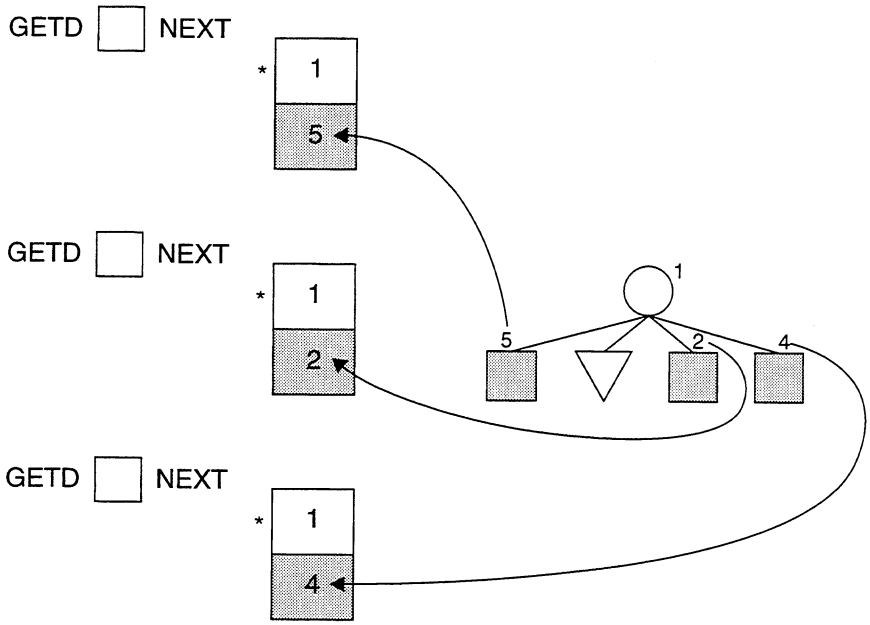

You can specify the S2KCOUNT option in a GET1 command to skip a certain number of data records of a given type and then retrieve one. You can also specify retrieval conditions with a where-clause, which is discussed later.

## GETD

The GETD command retrieves a descendant of the data record for the primary stack entry. It places the data in a specified subschema record and enters the address of the data record retrieved as a secondary entry in the stack. The software clears stack entries for data records not in the path of this descendant.

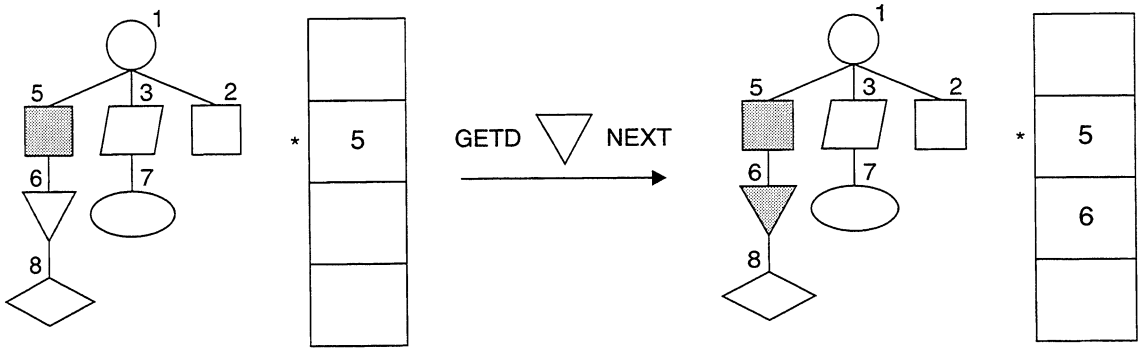
The data record whose address is the primary stack entry can have descendants corresponding to several schema records. With appropriate GETD NEXT commands, each of these descendants can be retrieved. Illus. 2.15 on page 30 shows how the GETD command processes all similar descendants of the data record for the primary entry one by one in logical order. Illus. 2.16 on page 31 shows how the execution of the GETD command affects the stack.

**Illus. 2.15** Retrieval with GETD NEXT

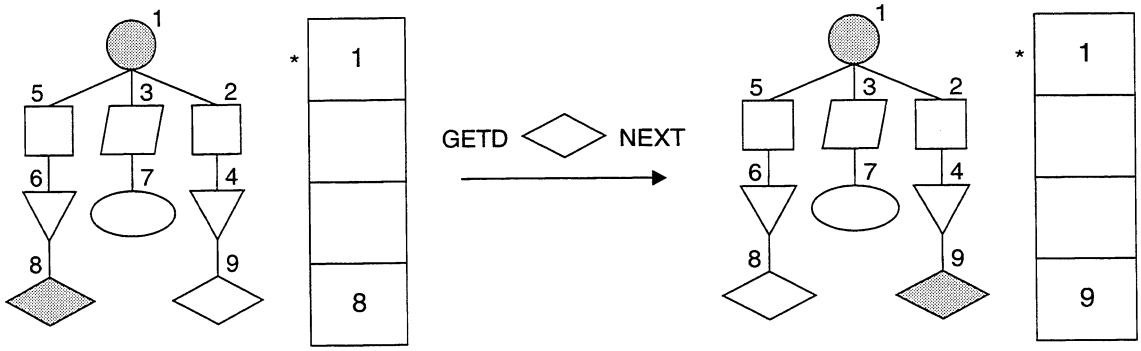


**Illus. 2.16** Stack Configurations with GETD

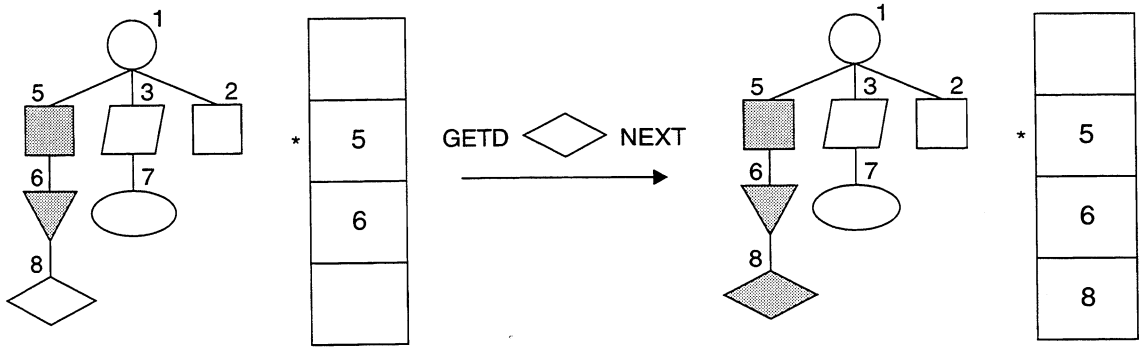
**Example 1**




**Example 2**



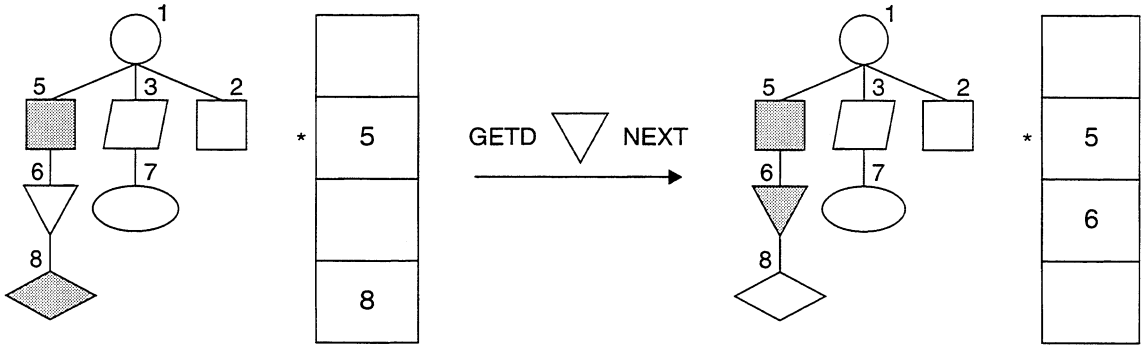

**Example 3**



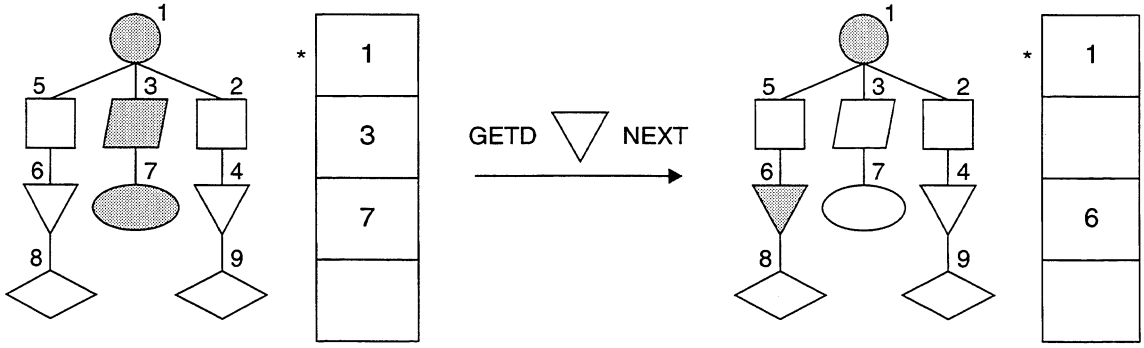

*continued on next page*

**Illus. 2.16** *continued*

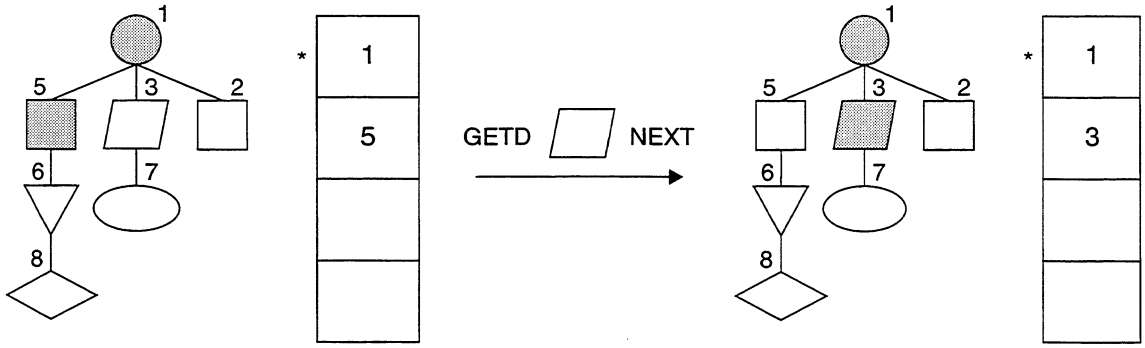
**Example 4**




**Example 5**




**Example 6**



The result of a GETD command depends on the stack and on the specified subschema record. The contents of the stack determine which data records corresponding to the subschema record are available for retrieval. The group of data records available for retrieval is the domain of the command. Thus, for example, the command GETD ... NEXT means get the next record for the specified SSR in the domain.

As Illus. 2.17 on page 34 shows, the domain of a GETD command is a subset of the descendants of the data record for the primary entry. More specifically, it is a subset of those descendants that correspond to the subschema record specified in the command. The extent of the domain is determined by the lowest-level stack entry. That is, the domain consists of those descendants of the primary entry (that correspond to GETD's specified SSR) that are also the descendants of such a stack entry.

The record to be retrieved from the domain is specified in the command. GETD FIRST means the first data record in the domain. GETD LAST implies the last and GETD NEXT implies the record next to the one whose address is already in the stack. GETD PRESENT means the current record.

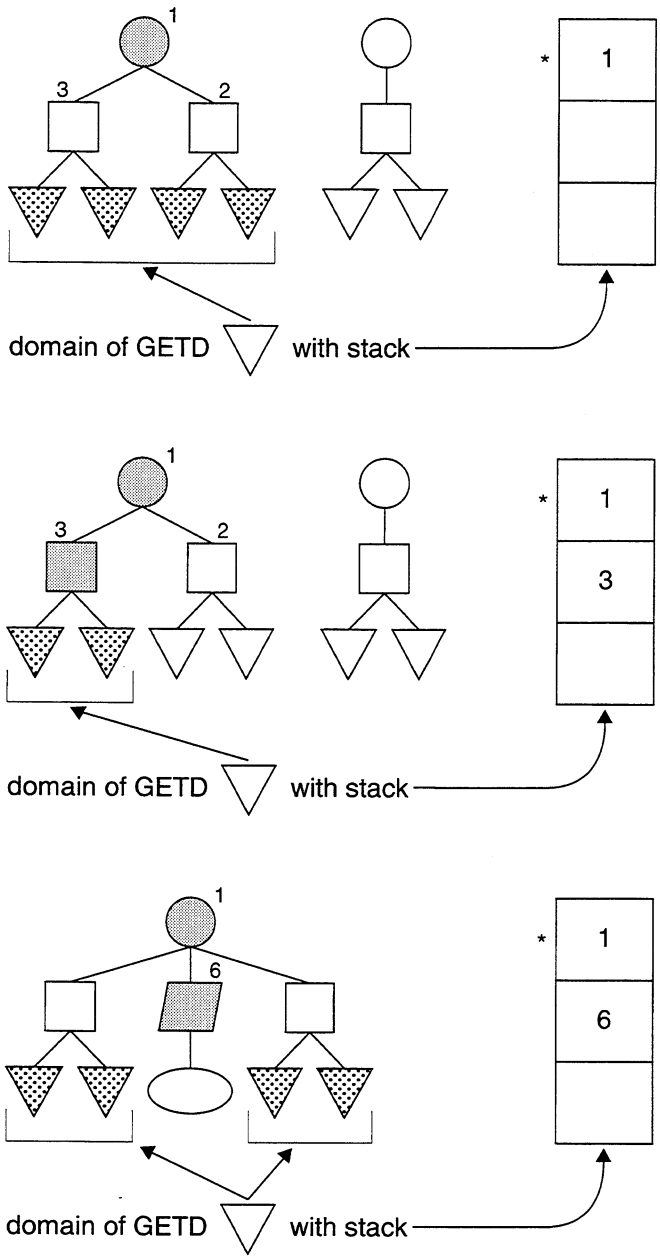
Thus, the stack plays a central role in the execution of the GETD command.

## GETA

The GETA command retrieves any ancestor of either the primary or a secondary stack entry. It places the data in a specified subschema record. It also enters the address of the retrieved data record as a secondary stack entry at the appropriate level. Illus. 2.18 on page 35 shows the effect of GETA execution on the stack. The GETA command alters only one level of the stack.

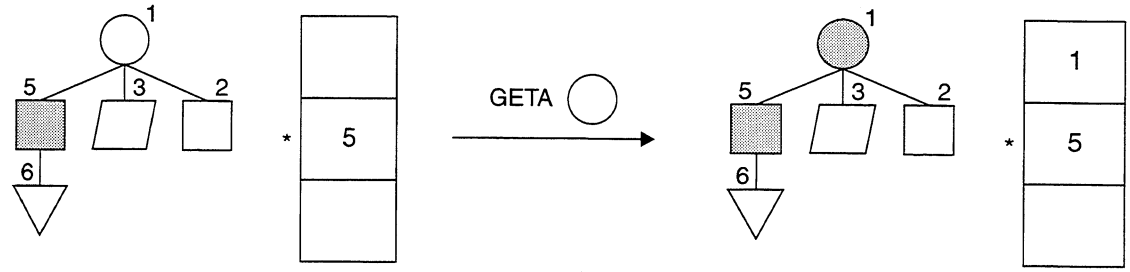
For example, GETA means get a record that is the ancestor of some entry in the stack. Here again, the stack determines which data record belonging to the specified schema record must be retrieved.

**Illus. 2.17** The Domain of GETD

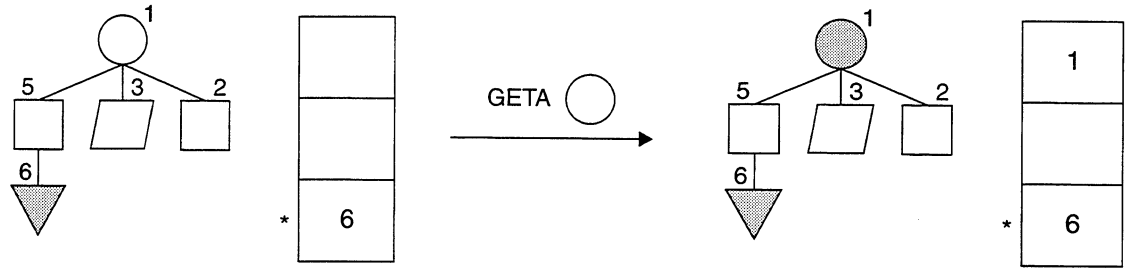


**Illus. 2.18** Stack Configurations with GETA

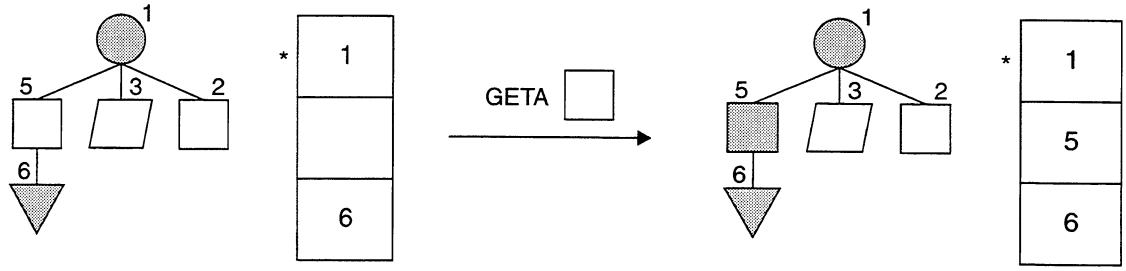
**Example 1**



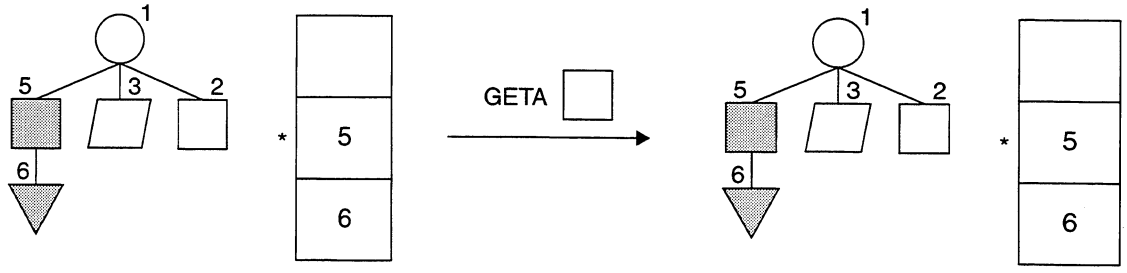

**Example 2**




**Example 3**




**Example 4**



# SUBSET RETRIEVAL

This section discusses how you can specify a where-clause to select a subset of the database for processing and how data can be retrieved from this subset. PLEX offers two types of where-clauses: fixed and DYNAMIC.

## Fixed Where-clauses

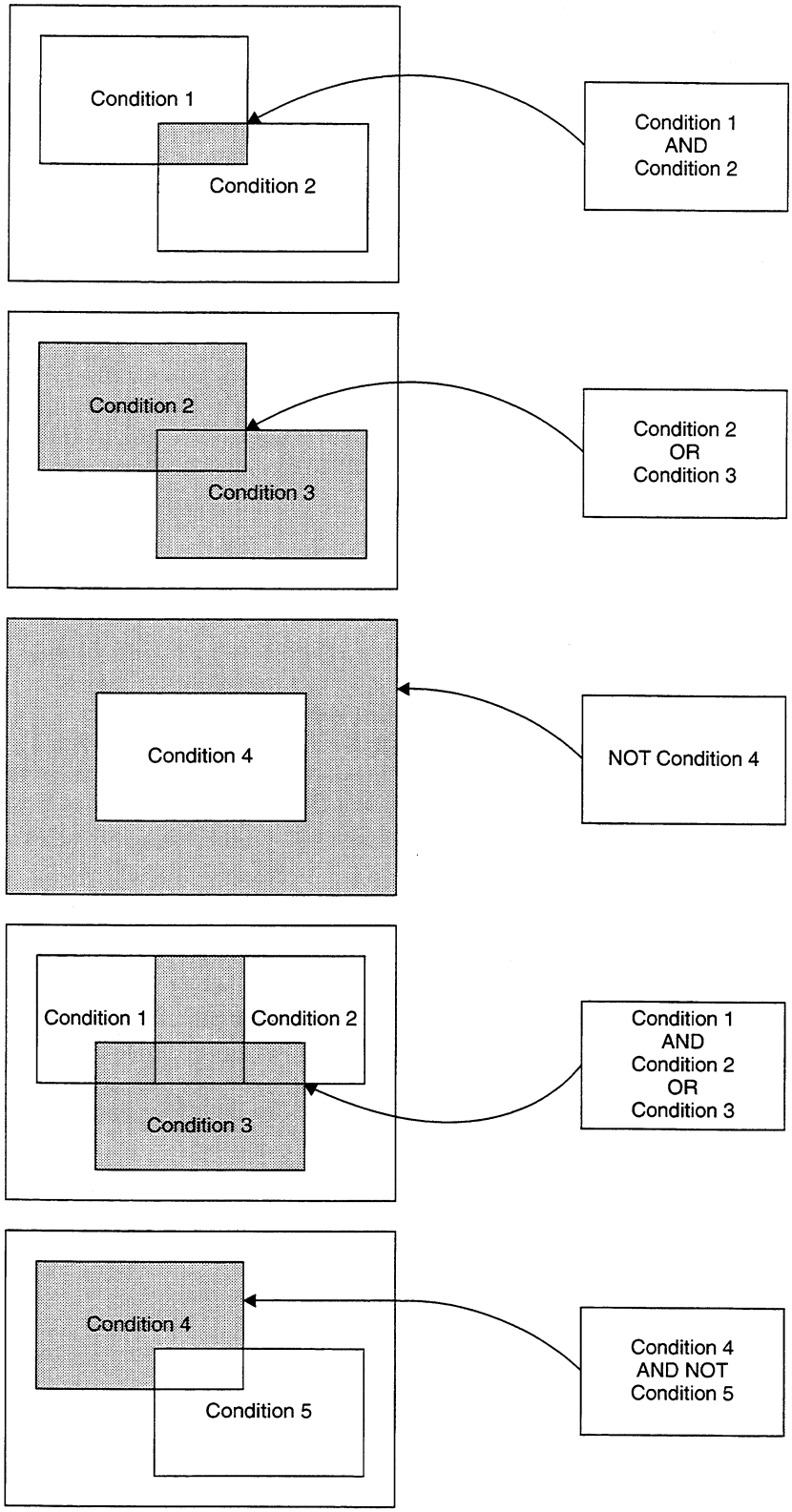
The fixed where-clause is coded and compiled within your program. It contains one or more conditions, joined by AND, OR, or NOT. The where-clause processor manipulates lists of data records that satisfy the given conditions and form a single subset of the database. Illus. 2.19 on page 37 shows some examples of subsets selected with various where-clauses.

You can form conditions using any of the operators EQ, NE, LT, LE, GT, GE, (or the equivalent symbols =, !=, <, <=, >, >=), SPANS, EXISTS, or FAILS. You can also form conditions by specifying that an existing database subset become a condition. Examples of some conditions are shown below:

```
SOCIAL-SECURITY-NO EQ SSN
JOB EQ JOBCLASS
HIREDATE LT VARDATE
SALARY GT VARSAL
POSITION EXISTS
EMPLOYEE FAILS
```

The left operand is the name of a subschema item, and it represents an item in a schema record. The right operand, if present, is a variable or a subschema item. A data value in the database can be compared with the value of a program variable. The existence or nonexistence of values in the database can be checked by using EXISTS and FAILS.

**Illus. 2.19** Subset Selection Using AND, OR, and NOT



SYSTEM 2000 software also offers the HAS and AT operators, which operate on the lists of data records satisfying conditions. The HAS operator checks to see whether a condition is true in a given subtree. The AT operator checks to see whether a given condition is true of data records at specific positions under their parents; for example, it can find each data record that is the first child of its parent and also satisfies the given condition.

When a database is defined, schema items can be designated as key items. For quick reference, SYSTEM 2000 software builds and maintains an Index for values of key items. It uses multi-level indexing and efficient search algorithms to find a key value. Conditions in most where-clauses can contain either key or non-key values.

## DYNAMIC Where-Clauses

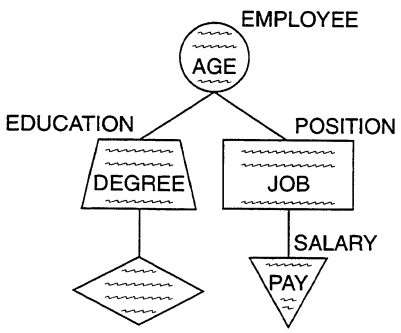
A DYNAMIC where-clause is an SCF QUEST where-clause. Your program can build DYNAMIC where-clauses during program execution, or it can read them in from user input. A DYNAMIC where-clause can contain all the QUEST where-clause operators except the SAME operator. These where-clauses are passed directly to SYSTEM 2000 software for processing; the PLEX processor has no knowledge of them.

## Normalization

A simple condition always deals with a single item or a single schema record (SR) of the schema tree. Therefore, a subset of a database formed by a simple condition contains similar data records (corresponding to the same SR). When a where-clause includes conditions that have associated subsets at different levels of the database, or if you want a subset at a level different from the level of any condition, some adjustments are made in the subsets. Ultimately, a single subset of the database will be selected. However, before that selection takes place, the levels of subsets qualified through the conditions must be adjusted upward or downward to a common level.

If the level of a subset of data records is adjusted to a higher database level, the ancestors at that level replace the original data records in the subset. The resulting subset consists only of these ancestors. Conversely, when the level of a subset is adjusted to a lower level, the descendants at that level replace the original data records in the subset. The process of adjusting the levels of all qualified subsets to a common level is called normalization.

There are three kinds of normalization. The illustrations of them in the next few pages use the following schema tree:

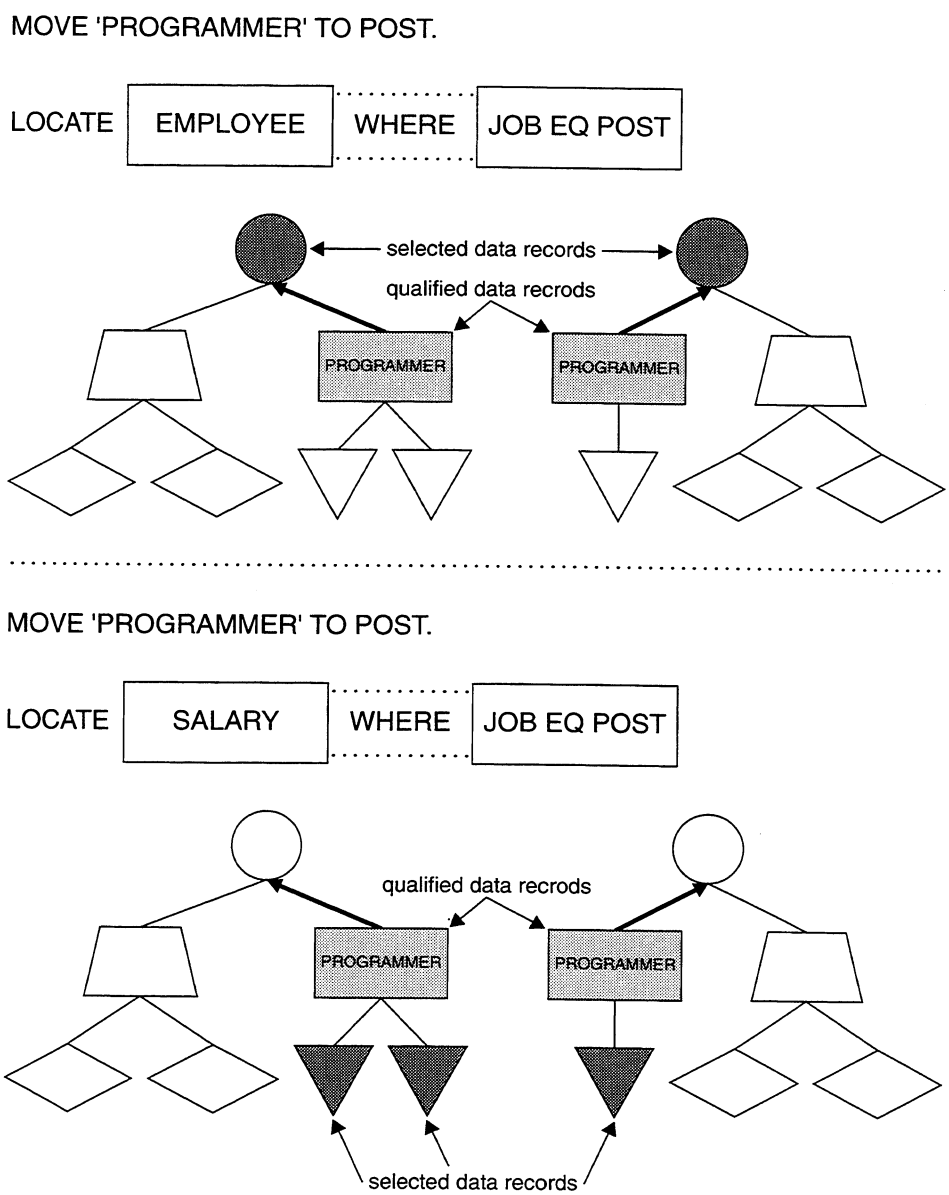


### Normalization of the First Kind

When the desired subset is at a different level than the level of the condition, SYSTEM 2000 software finds the subset using the subset generated by the condition, as shown in Illus. 2.20.

The value of the variable POST is PROGRAMMER. The where-clause qualifies the data records at level 1 containing PROGRAMMER. But the object of the first LOCATE command (EMPLOYEE records) is at level 0, while the object of the second LOCATE command (SALARY records) is at level 2. Thus, SYSTEM 2000 software normalizes upward in the first case, and downward in the second case to locate the desired data records.

**Illus. 2.20** Normalization of the First Kind



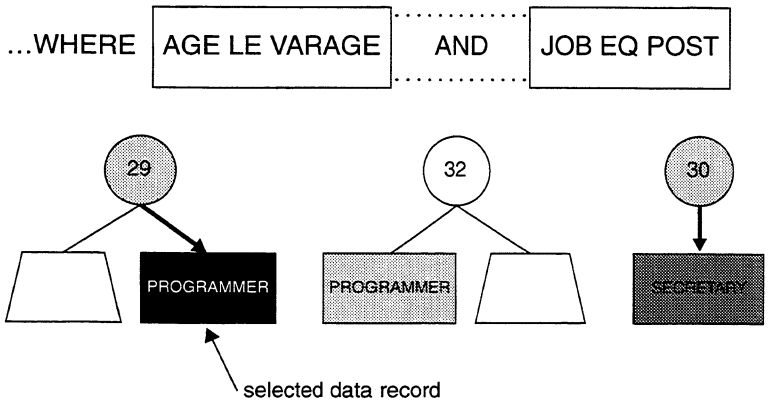
**Normalization of the Second Kind**

AND and OR can operate on similar records only. When two conditions at different levels are to be combined by AND or OR, SYSTEM 2000 software automatically normalizes downward as shown in Illus. 2.21.

**Illus. 2.21** Normalization of the Second Kind

MOVE 30 TO VARAGE

MOVE 'PROGRAMMER' TO POST.



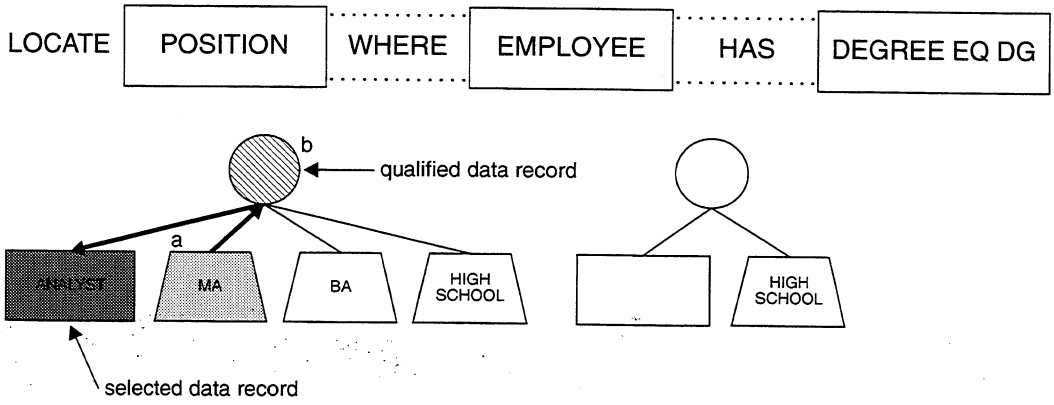

The values of the variables VARAGE and POST are 30 and PROGRAMMER, respectively. Two data records at level 0 satisfy the condition AGE LE VARAGE, and the condition JOB EQ POST qualifies the two data records at level 1 containing PROGRAMMER. SYSTEM 2000 software normalizes the first condition to level 1. Then the intersection can be taken to process AND. The data record containing PROGRAMMER under AGE of 29 becomes the selected record.

### Normalization of the Third Kind

With the HAS operator, the left operand represents a schema record, and the right operand is a condition or expression (a compound condition). The schema record on the left is usually an ancestor of the schema record in the condition on the right. SYSTEM 2000 software normalizes the subset produced by the condition(s) on the right to the level of the schema record on the left as shown in Illus. 2.22.

**Illus. 2.22** Normalization of the Third Kind

MOVE 'MA' TO DG.



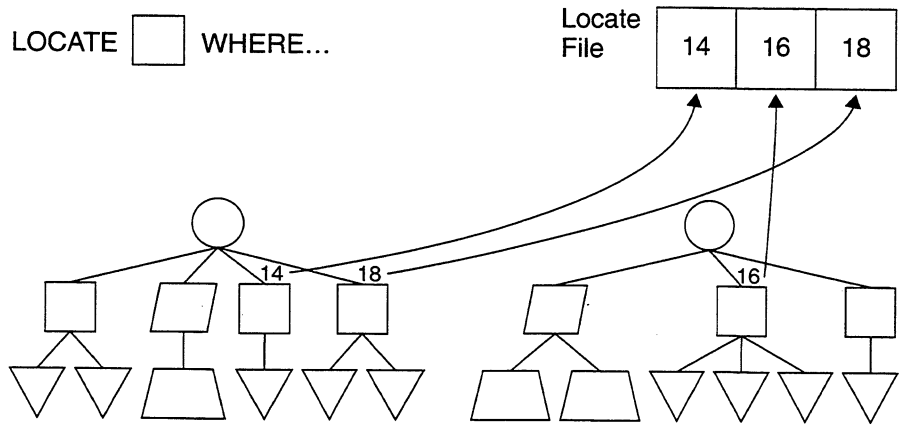

DEGREE EQ DG qualifies data record "a" at level 1. The EMPLOYEE record is at level 0. So the subset is normalized upward to derive the data record b. Then it must be normalized downward to select the POSITION record, which is at level 1.

## The Locate File and LOCATE Command

You specify a where-clause in a LOCATE command to locate a subset of data records satisfying the conditions in the where-clause. The number of records selected is placed in the sixth COMMBLOCK field, and the addresses of the selected data records are placed in a file called the Locate File. You can retrieve these data records using the GET command discussed in **GET** on page 43.

After the where-clause processing is completed, the software sorts the addresses of the selected data records into physical order, which is the order in which the records are stored in the Data Table. Illus. 2.23 shows the result of executing a LOCATE command.

**Illus. 2.23** Execution of a LOCATE Command




In some situations, sorting the addresses in physical order can be avoided, using the no-sort capability. See Appendix A for a discussion of this capability.

The Locate File that results from where-clause processing can itself be used as a where-clause condition in a later command. In this way, the Locate File readily serves as an already-selected subset of the database without reprocessing the where-clause conditions from which it resulted. In addition, the primary entry in a stack can be specified as a where-clause condition.

SYSTEM 2000 software keeps a pointer to the most recently retrieved address in the Locate File. This pointer is initialized by the LOCATE command and is moved by subsequent GET commands.

## ORDER BY

Addresses in the Locate File can be sorted in various ways using the ORDER BY command. A Locate File can be sorted several times, once for each of several processes, and up to 40 sort keys can be used in each sort. Items and schema records can be used as sort keys in an ORDER BY command, and the addresses in the Locate File are sorted by the values of the corresponding sort key items in the database. With each sort key, you can specify ascending or descending order for sorting. ORDER BY reinitializes the Locate File pointer.

## Multiple Locate Files and COPYL

Each PLEX program can use up to 16 Locate Files. START S2K, the first PLEX command executed in a program, reserves the Locate Files required. You reference the Locate Files by the numbers 0 through 15. The Locate File for a given number is always associated with the same database as the stack referenced by that number.

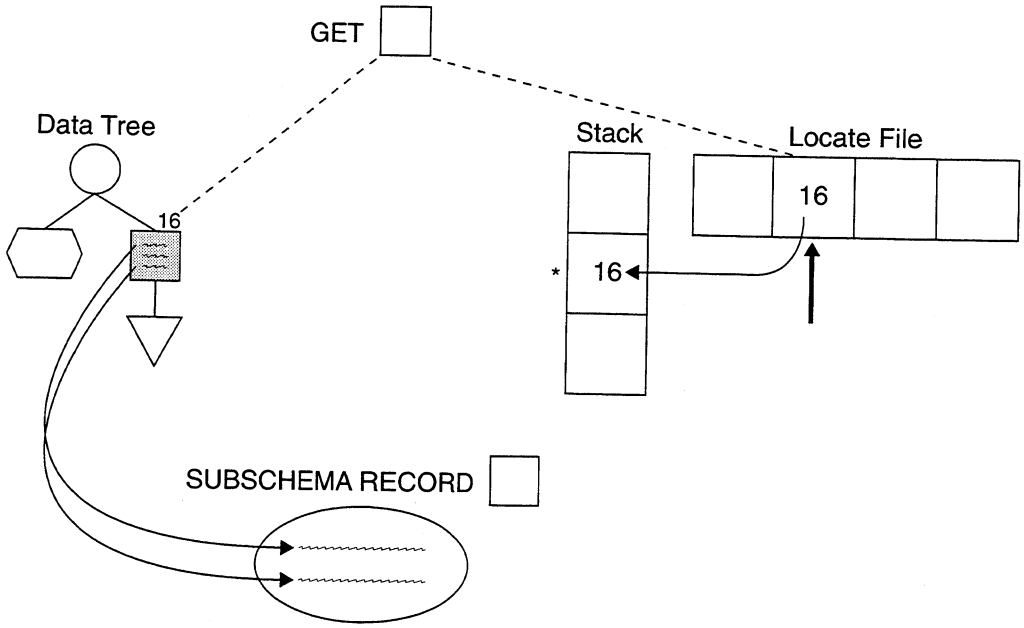
Just as a program can copy a stack (using the COPYS command), it can copy a Locate File using the COPYL command. In this way, for example, the original arrangement of addresses can be preserved in one Locate File, while that same arrangement is changed or processed in a different way in another Locate File. There is a one-to-one correspondence between Locate Files and stacks. The Locate File collects the addresses of data records, and these data records are retrieved one-by-one from the Locate File, using a command called GET. The GET command enters an address from the Locate File in the stack. The reference numbers of the Locate File and the stack used in the GET command must match when you use multiple Locate Files and stacks.

## GET

The GET command retrieves a data record whose address exists in a Locate File. Like the GET1 command, GET retrieves the data values from a data record into a subschema record and enters the address of the data record in the stack as a primary entry. A GET command causes the Locate File pointer to be moved to the address of the data record that is retrieved. Illus. 2.24 on page 44 shows retrieval with a GET command.

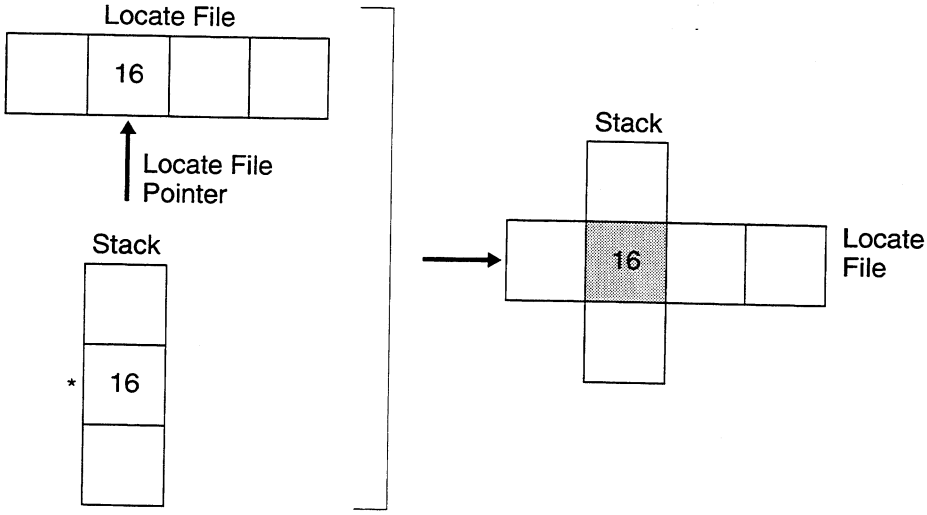
You can issue GET NEXT repeatedly to retrieve all the addresses from the Locate File to the stack one-by-one from left to right as shown in Illus. 2.25 on page 45. Repeated use of GET PREVIOUS brings the addresses from the Locate File to the stack one-by-one from right to left as shown in Illus. 2.26 on page 46. (Other syntax options are available with the GET command, as discussed in **GET** on page 8-100.)

**Illus. 2.24** Retrieval with GET

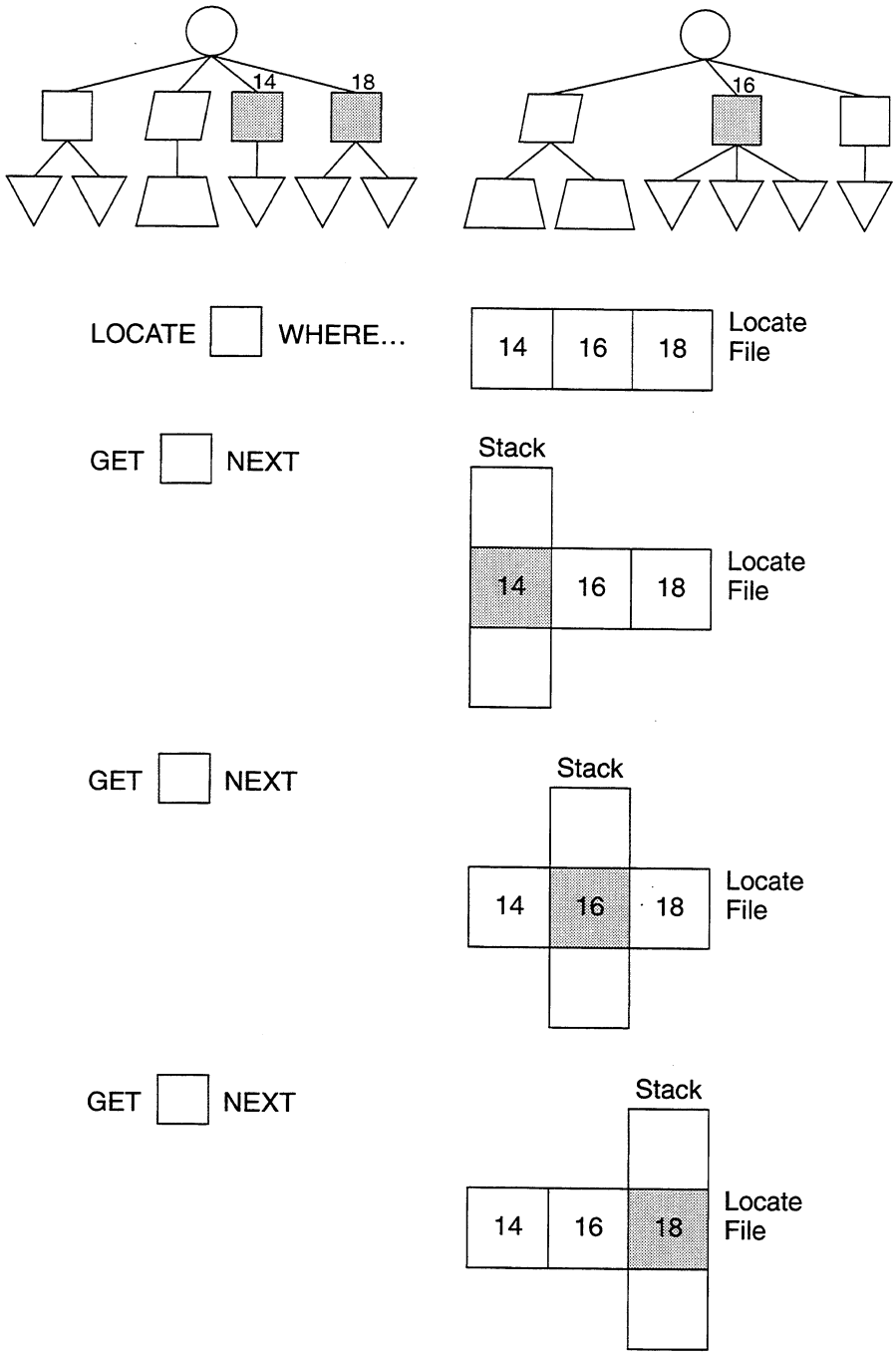



With GET, the record address is placed in the stack, and data values are brought into the subschema record.

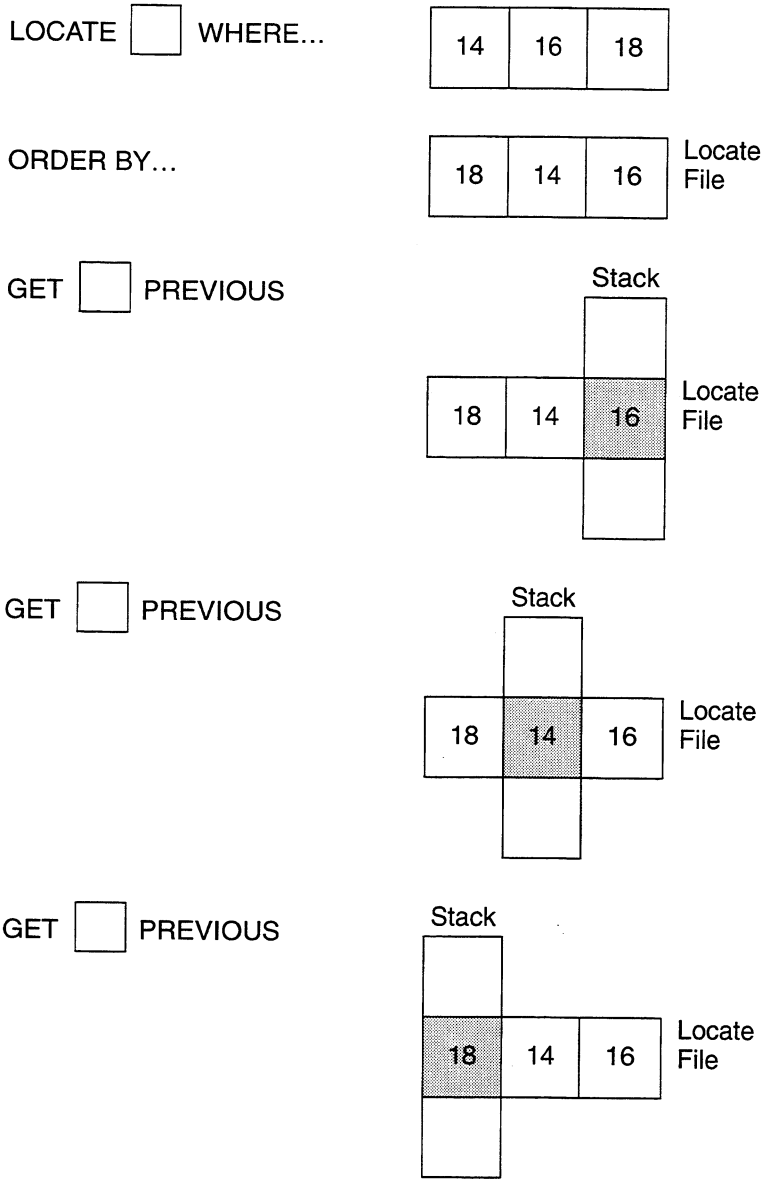
Note: From now on, a stack and a Locate File that are used together with a GET command are shown intersecting at the primary stack entry. The placement of the stack shows the position of the Locate File pointer.



**Illus. 2.25** Stack with GET NEXT



**Illus. 2.26** Stack with GET PREVIOUS

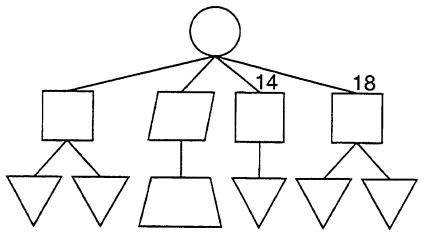


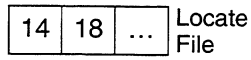

Since GET enters a primary entry in the stack, it sets up the potential for secondary retrievals using GETA and GETD. Thus, you can give any combination of the LOCATE, ORDER BY, GET, GETA, and GETD commands, as shown in Illus. 2.27 on page 47.

**Illus. 2.27** Possible Combination of Retrieval Commands

LOCATE □ WHERE ...

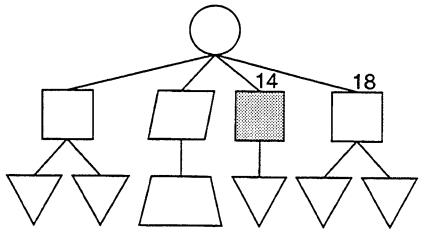
<Locate the subset;
then check
the return code.>

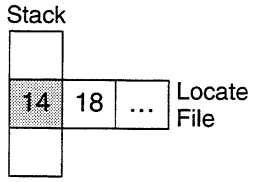



GET □ NEXT

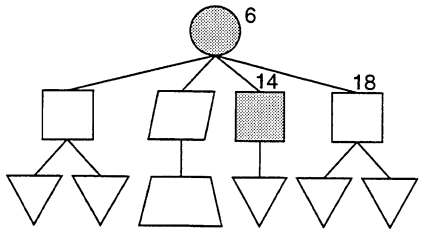
<Get the next □ ;
then check
the return code.>

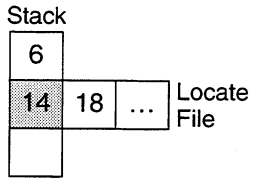



GETA ○

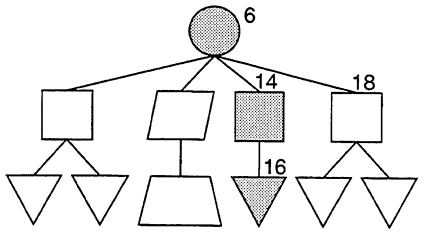
<Obtain ancestor;
then check
the return code.>

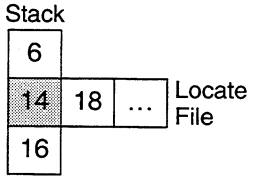



GETD ▽

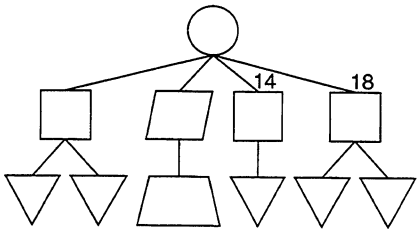
<Retrieve descendant;
then check
the return code.>

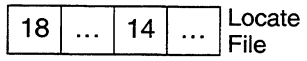



ORDER BY...

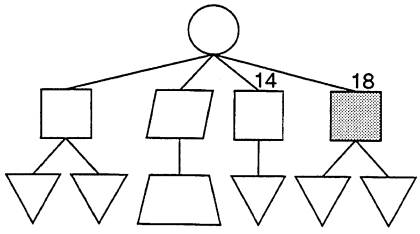
<Sort the Locate File
addresses; then check
the return code.>

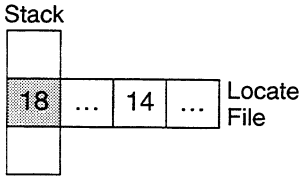



GET □ NEXT

<Get the next □ ;
then check
the return code.>



### GET1 with a Where-clause

If the subset resulting from a where-clause is to consist of a single data record, you do not need a Locate File. You can retrieve this unique data record directly with a GET1 command that includes a where-clause. The where-clause need not be at the same level as the specified subschema record.

Example:

```
GET1 EMPLOYEE WHERE SOCIAL-SECURITY-NUMBER EQ GIVEN-NUMBER
```

### GETP

The command GETP (get position) retrieves a single data record in order to establish a desired position in the database. A successful GETP command establishes a primary stack entry. The command requires a single key condition where-clause. However, unlike previously discussed retrieval commands, GETP avoids normal where-clause processing involving lists of qualified records. Instead, GETP uses the Index to select the record which satisfies the key condition. Multiple occurrences of the specified value are permitted, but generally the command is intended for use with a unique value. The specified value might exist or not exist; GETP tries to find a record that contains a value equal to or closest to the specified value.

GETP is particularly useful when an update pertains to a single record in a long sibling chain. For most GETP applications, the data retrieval function is incidental to positioning. Because of its relative efficiency, GETP is preferred to GET1 . . . WHERE . . . in applications that involve a single key condition containing a unique value.

Example:

```
GETP ENTRY-REC WHERE EMPLOYEE NUMBER EQ NUM
```

## LINKED RETRIEVAL

SYSTEM 2000 software supports a hierarchical data model because both the schema and the logical data structure are constructed as trees. However, symbolic relations are also supported through the LINK command.

### Symbolic Relations

You can establish a symbolic relationship (link) dynamically between two subschema records with the LINK command. The SSRs being linked can belong to the same database or to different databases. Symbolic relationships are non-hierarchical, and they are not identified to SYSTEM 2000 software at the time a database is defined. You design links dynamically as needed in your PLEX program.

A symbolic relation is a one-way relation. If SSR1 is linked to SSR2, for example, the retrieval of a data record for SSR1, results in the retrieval of a data record for SSR2, but not vice versa. Symbolic relations are shown with arrows in Illus. 2.28 on page 50. The subschema record from which the arrow starts is called the source, and the subschema record where the arrow terminates is called the target.

## LINK

The LINK command can either establish or cancel a symbolic relation. But the actual retrieval of a data record belonging to the target of the link takes place only after a data record for the source SSR is retrieved. A data record belonging to the source SSR in the first LINK statement can be retrieved with a GET1, GET, GETA, GETD, or GETP command.

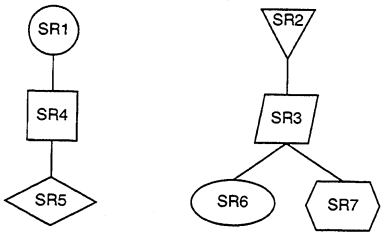
You can establish up to ten links in a PLEX program. The target of one link can be the source of another link, which causes a series of retrievals called a cascade. Multiple links can start from the same source. Thus, complex networks can be established with the LINK command, as shown in Illus. 2.29 on page 51. When this network is activated by a single retrieval, a cascade of retrievals results.

The LINK command contains a via-clause that determines which data record of the target SSR is to be retrieved. The via-clause is like a where-clause, and it specifies conditions for items of the target SSR. These conditions can use values obtained by the retrieval of the source.

The result of one of the linked retrievals is the same as that of a LOCATE command whose subschema record is the target and whose where-clause states the via-clause conditions, followed by a GET FIRST command.

SYSTEM 2000 software maintains a link-history table in a control block called S2KDUM. Your program can examine this table to see if all the retrievals in a cascade were successful.

**Illus. 2.28** Establishing Symbolic Relations with LINK Commands

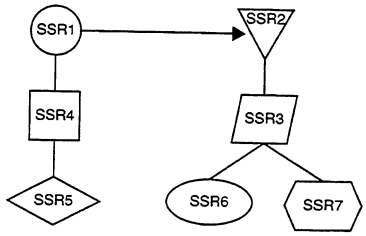



definition tree 1 definition tree 2

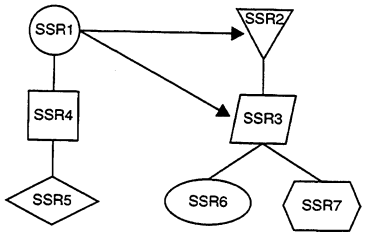
These two trees of schema records can belong to the same schema tree or to two different schema trees.

These LINK commands specify Locate Files and stack numbers as well as these subschema records corresponding to the schema records above.

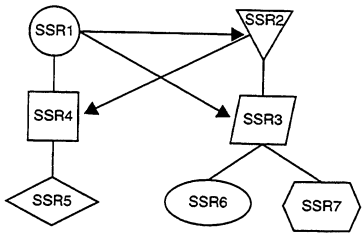
LINK1(5) SSR1 TO SSR2



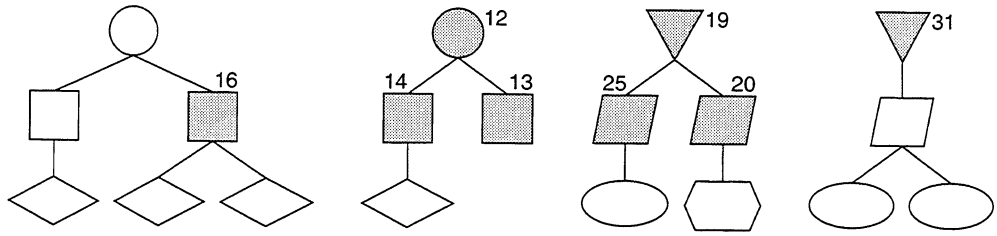

LINK2(6) SSR1 TO SSR3




LINK3(7) SSR2 TO SSR4

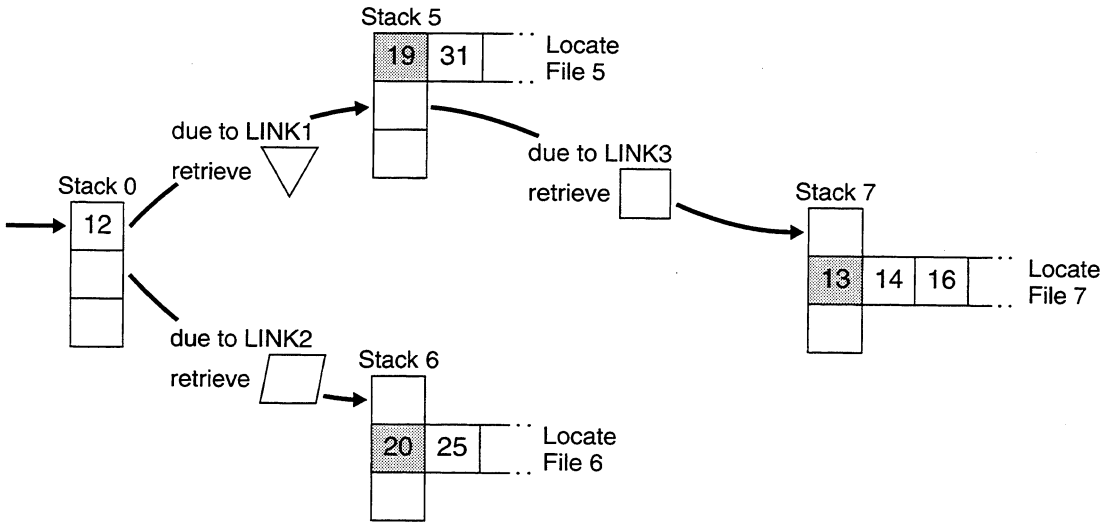


**Illus. 2.29** Linked Retrieval




Suppose these are the data trees corresponding to the two schema trees in the previous illustration.

As a result of GET ◯ into stack 0, the following retrievals can occur:



# UPDATING A DATABASE

This section discusses processing updates to a database, that is, the commands and procedures for adding, changing, and removing data and for the loading of data into a newly defined or existing database.

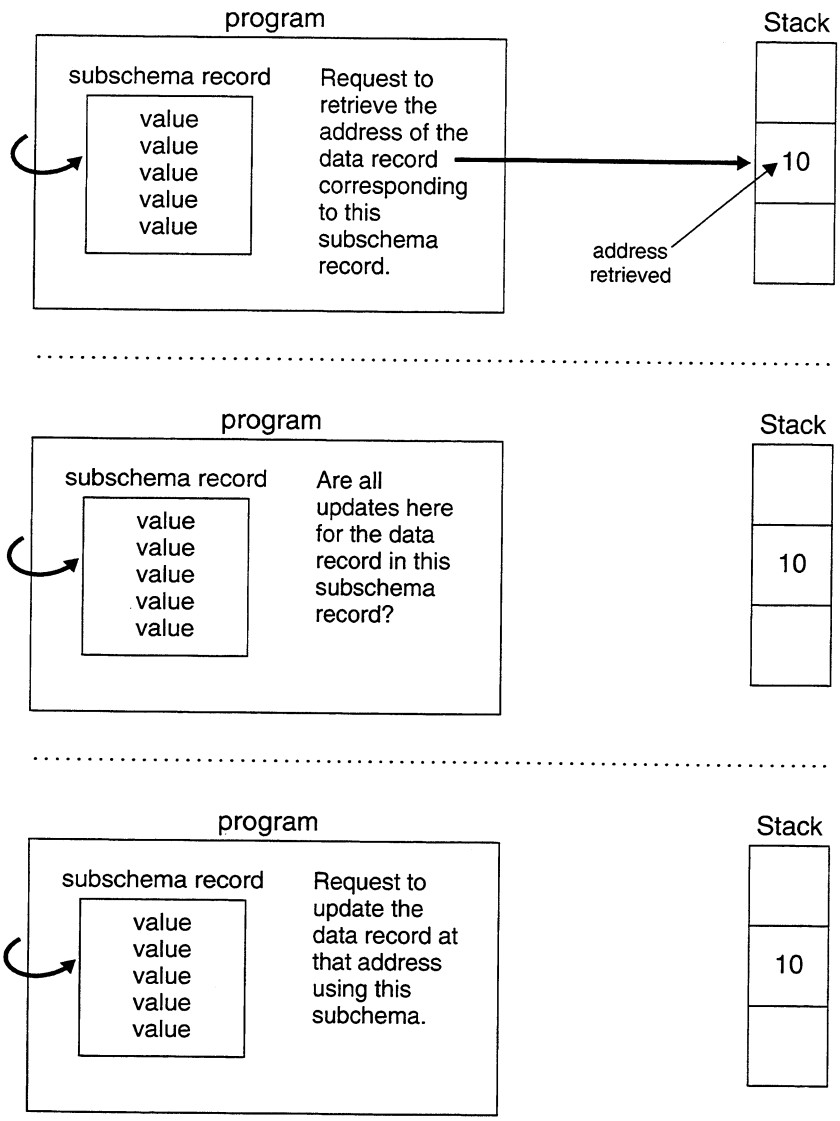
## Preparing for an Update

Before SYSTEM 2000 software can perform an update, the PLEX program must specify where the update is to be made and which data record should be changed. The program first gives a retrieval command to establish a stack entry for the data record. Then, in an update command, it specifies the subschema record holding the values to be inserted or modified in the data record. You can also retrieve a data record address, without retrieving any data values, by specifying USING 0 in a retrieval command.

When you want to replace values in a data record or add a new data record to the database, you specify the subschema record containing the update values. However, when you want to delete values or data records, SYSTEM 2000 software needs to know only the appropriate data record address. The essential preparatory steps for any update are shown in Illus. 2.30 on page 53.

There are two kinds of updates: changing values in data records and modifying the structure of the database by adding or deleting data records. The next two topics cover in more detail the preparation and commands for value modification and structure modification.

**Illus. 2.30** Preparation for an Update



## Value Modification

You can update a data record by adding values or by deleting or changing the data values already in it. Value modification is the replacement or deletion of some or all the values in a data record while maintaining the data record's place in the data structure. For the task of value modification, PLEX provides the commands REMOVE and MODIFY.

### REMOVE

The REMOVE command deletes some or all values in a data record. The places in the data structure that were occupied by those values are preserved. Before you issue a REMOVE command, you issue a retrieval command to establish a stack entry for the data record whose values are to be removed. The REMOVE command specifies a subschema record associated with that record.

In its simplest form, the REMOVE command calls for the removal of all values for items in a data record that correspond to the items in a specified subschema record. The subschema items correspond to items in the schema record (SR) for that data record. Illus. 2.31 on page 55 shows this correspondence between the subschema record and the SR. It also shows the steps for removing some of the values in the data record with the REMOVE command. A data item that has no value is called a null item. If a subschema record description has items for all the values in a data record, a REMOVE command specifying that subschema record causes all items in the data record to become null.

### MODIFY

The MODIFY command changes some or all of the values in a data record. Also with MODIFY, you can give values to null items, and you can make valued items null.

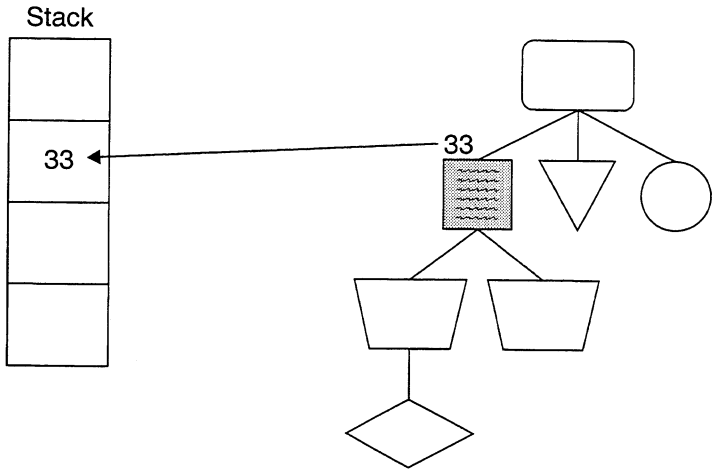
When you want to replace existing values in a data record with new ones, you first give a retrieval command to establish a stack entry for the data record to be modified. Next, you move the new values to a subschema record for the data record just retrieved. Then you issue a MODIFY command that specifies the subschema record containing the new values.

Illus. 2.32 on page 56 shows how SYSTEM 2000 software executes a MODIFY command, replacing the old data values with new ones that were placed in a subschema record. The software stores the new values in the data record according to the correspondence of the subschema items to the schema record items. The same methods apply to assigning values to null items and to making valued items null.

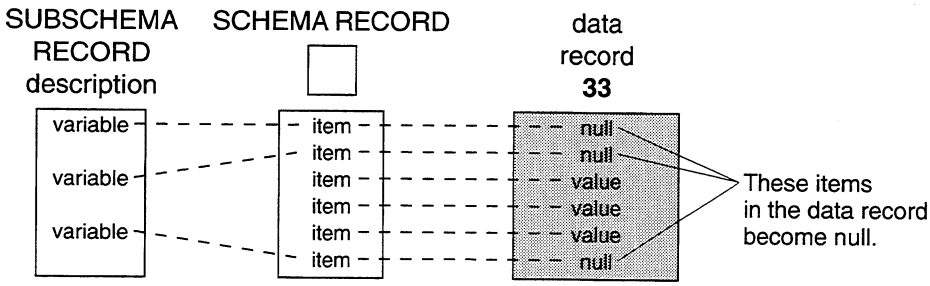
To allow structural changes, PLEX offers two other update commands, which are covered next.

**Illus. 2.31** Using REMOVE to Delete Record Values

**1** Using a retrieval command, put the address of the required data record in the stack.

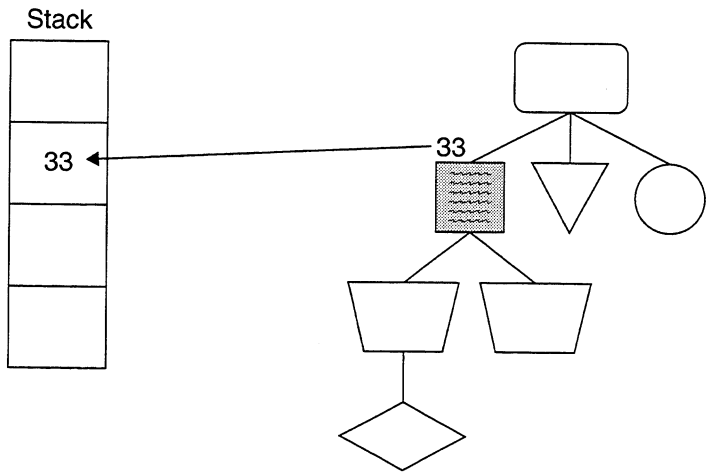



**2** Give a REMOVE command specifying a subschema record associated with the data record.

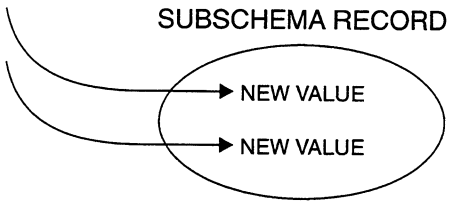


**Illus. 2.32** Using MODIFY to Replace Record Values

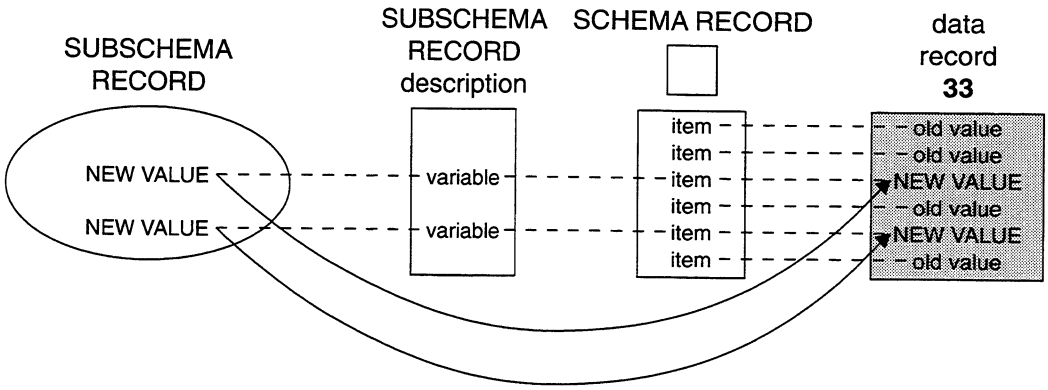
**1** Begin with a command to retrieve the data record to be updated and establish a stack entry.




**2** Move new values into a subschema record associated with that data record.




**3** Give a MODIFY command specifying the subchema record.



## Structure Modification

You can change the data structure of a SYSTEM 2000 database by removing, inserting, or moving a data record or subtree of data records. The PLEX commands that handle these functions are REMOVE TREE, INSERT, and MOVE TREE.

### REMOVE TREE

In a SYSTEM 2000 database, data records are related to each other in a hierarchy. Removing a data record from the structure removes its descendants as well. In effect, the software removes a subtree from the structure, which changes the pointers in the Hierarchical Table.

To prepare for the removal of a subtree, you issue a retrieval command to establish a primary or secondary stack entry for the subtree's top node. Then you issue a REMOVE TREE command referring to that top node. Illus. 2.33 on page 58 shows the basic steps for using the REMOVE TREE command. Illus. 2.34 on page 59 shows other possible ways to change the data structure with REMOVE TREE.

### INSERT

The INSERT command adds a data record to the data structure. A PLEX program can give a series of INSERT commands to add an entire tree of data records, one by one, to the structure. As in all other updates, the program gives a retrieval command beforehand to indicate the location in the data structure where the insertion will be made.

You specify the location of the update in two ways. First, you establish a stack entry, using a retrieval command. This stack entry represents either the parent or a sibling of the new data record to be inserted. The second specification for location is the place (relative to the parent or sibling) where the data record will be inserted.

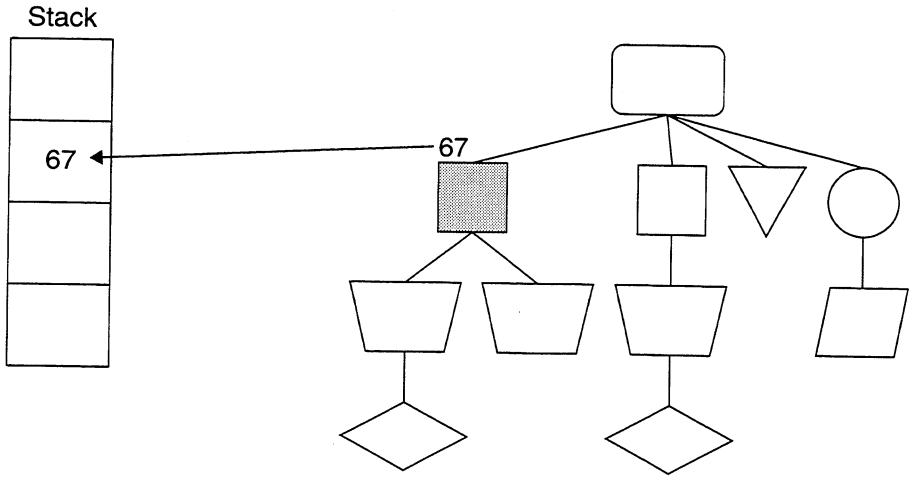
For example, if you establish a stack entry only for the parent, you can give an INSERT command specifying that the new data record should be inserted before all that parent's children. The new data record would then be leftmost in logical order under the parent. Similarly, the INSERT command could specify that the data record should be inserted in logical order under the parent after the last child corresponding to the same schema record. Illus. 2.35 on page 60 shows an insertion of a subtree of data records in relation to a parent data record.

On the other hand, you can establish a location for a sibling of the data record to be inserted. The specification of BEFORE or AFTER would indicate the position of the new data record in logical order, relative to that sibling. Thus, the data record could be inserted in logical order immediately before or after the chosen sibling. Illus. 2.36 on page 60 shows an insertion of a subtree of data records in relation to a sibling of the top node of the new subtree.

SYSTEM 2000 software processes a series of INSERT commands and other update commands in one of several ways, depending on the execution mode designated in the PLEX program. The three execution modes are discussed in the next section.

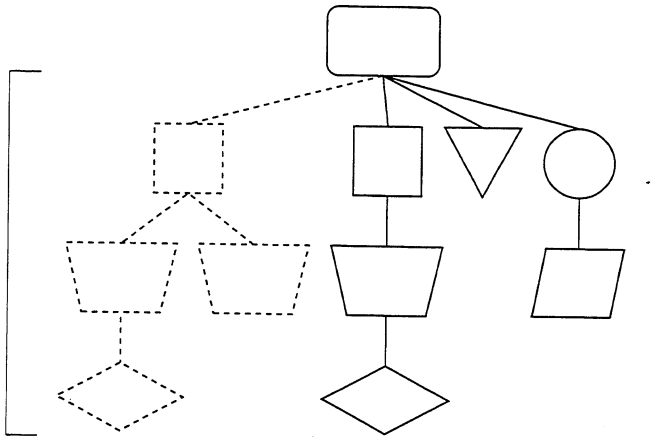
**Illus. 2.33** Using REMOVE TREE to Delete Parts of the Data Structure

**1** Begin with a command to retrieve the data record at the top of the subtree to be removed, establishing a stack entry.



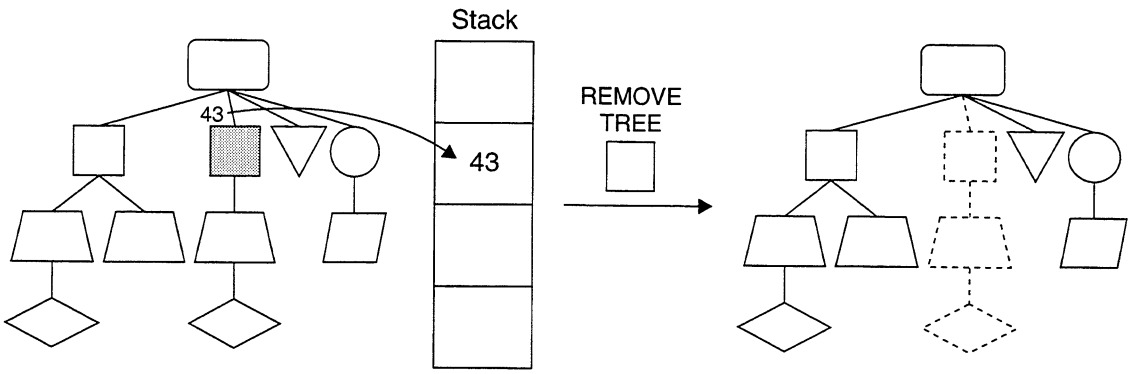

**2** Give a REMOVE TREE command, specifying a subschema record associated with that data record.

This subtree has been removed and the structural pointers changed.

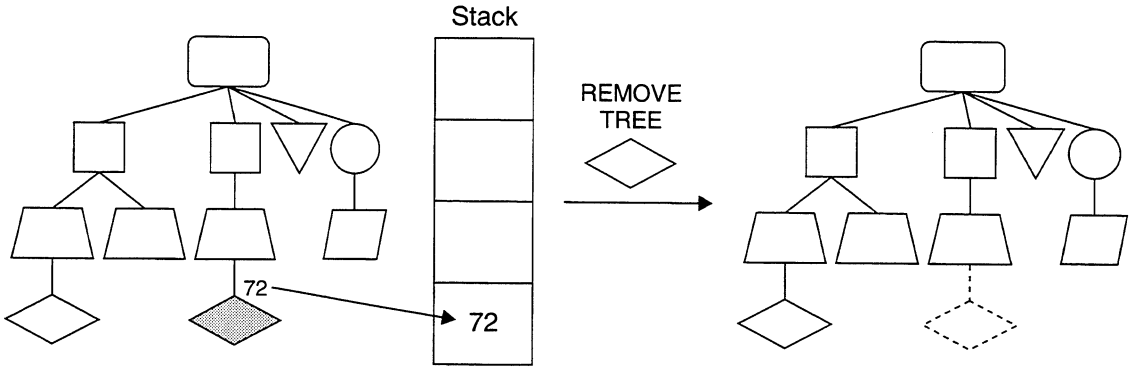


**Illus. 2.34** Possibilities with REMOVE TREE

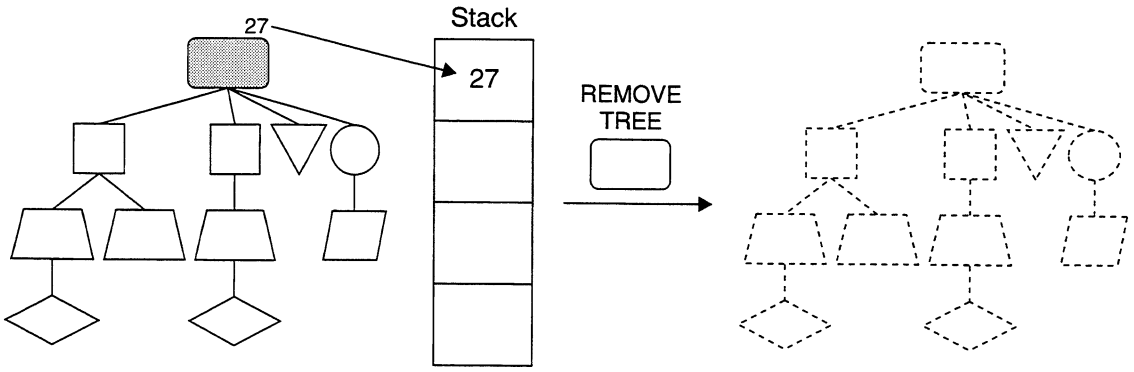
**1**



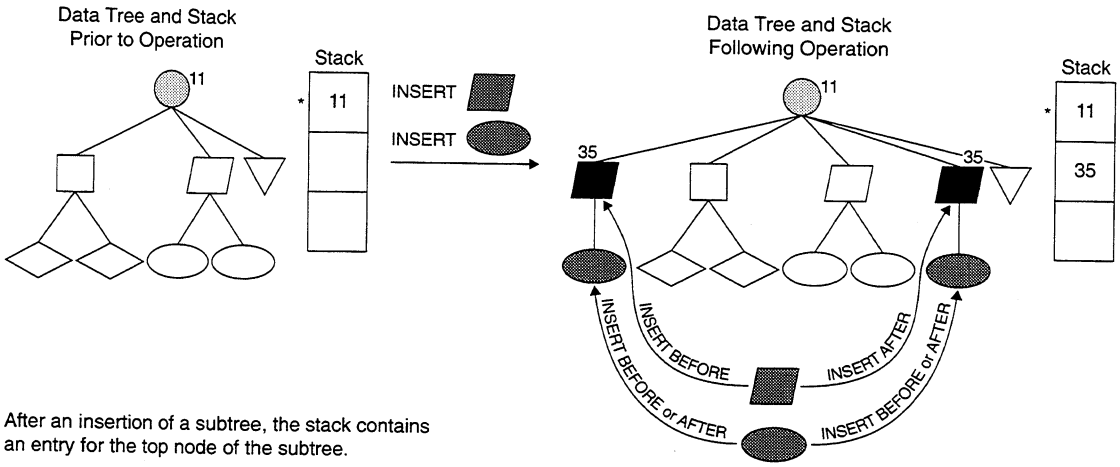

**2**




**3**

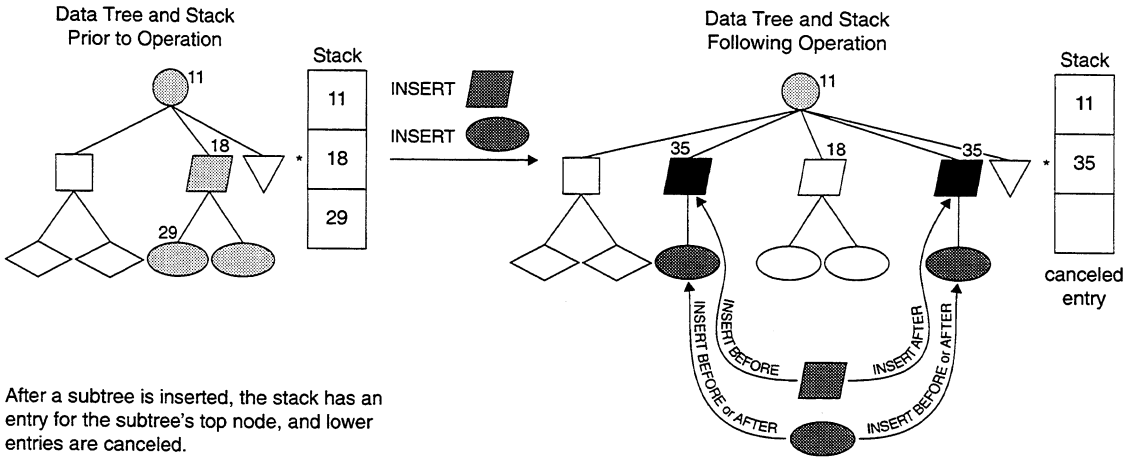


**Illus. 2.35** INSERT Placement with Respect to Parent




After an insertion of a subtree, the stack contains an entry for the top node of the subtree.

**Illus. 2.36** INSERT Placement with Respect to Sibling




After a subtree is inserted, the stack has an entry for the subtree's top node, and lower entries are canceled.

### MOVE TREE

The MOVE TREE command moves a data tree from one logical position in a database to another. The move is a logical move because only the data record pointers are changed in the Hierarchical Table; the physical records in the Data Table and the indexes are not changed.

To prepare for moving a tree, you must establish a "from" stack with a retrieval command. The top node of the tree to be moved must be a primary or a secondary stack entry in the "from" stack. Next, you establish a "to" stack with a retrieval command. In this stack, there must be a primary or secondary stack entry for the parent of the top node of the tree or for a sibling similar to the top node of the tree. The "to" stack is used to establish the position of insertion for the moved tree, as a child of a new parent node or sibling.

The MOVE TREE command specifies the subschema record to be moved as well as the "from" stack number and the "to" stack number. For example, the following commands move SSRB and all of its descendant records to a new logical position below the parent SSRA of the "to" stack:

```
GET1 (1) SSRB WHERE  . . . ("from" stack)
       :
       :
GET1 (2) SSRA WHERE . . .  ("to" stack with SSRA as the parent of SSRB)
       :
       :
MOVE TREE SSRB (1) TO (2).
```

The rules for insertion are the same as for immediate mode INSERT. You can specify AFTER or BEFORE; the default is AFTER.

The stack entries in the "from" stack are moved to the "to" stack, beginning with the top node of the moved tree. The top node is left as a stack entry in the "from" stack and lower level (descendant moved records) stack entries are cleared.

## Execution Modes

The three execution modes are immediate mode, queue mode, and load mode. Although retrieval commands in both immediate and queue modes are processed immediately, update commands are processed in different ways in those modes. The execution modes differ mainly in regard to when the changes are actually made in the database tables.

### Immediate Mode

A program is in immediate mode by default if it is not in queue mode or load mode.

The software processes each retrieval or update command completely before the next command in immediate mode. Processing update commands includes changing the Hierarchical Table (HT), the Data Table (DT), and the Index to reflect the updated data values or structure. Processing of INSERT commands is delayed only when you issue a series of INSERT commands for data records that are all in one tree. In this situation, the software delays the processing and inserts the entire tree all at once. Immediate mode is especially useful for occasional, random updates.

### Queue Mode

In queue mode you give a series of update commands as a batch. Your program enters queue mode with the QUEUE command and returns to immediate mode with the TERMINATE command. All retrieval commands given in queue mode are executed immediately. The update commands given between QUEUE and TERMINATE are batched to be executed after the TERMINATE command is given. So the software changes the database tables only after TERMINATE. Regardless of the order in which you give INSERT and REMOVE TREE commands in the batch, SYSTEM 2000 software executes REMOVE TREE commands before other updates and performs all updates in database physical order.

### Load Mode

Some aspects of INSERT processing in immediate and queue modes are combined in load mode. PLEX provides this execution mode for placing new data trees into a database, a process called populating or loading a database. Database populating in PLEX is done most efficiently in load mode.

You can give only INSERT commands in load mode. Your program enters load mode with the LOAD command and returns to immediate mode with the TERMINATE command. During a load mode session, the software makes tree insertions immediately in the Hierarchical Table and the Data Table, but it batches changes to be made in the Index for execution when the TERMINATE command is given.

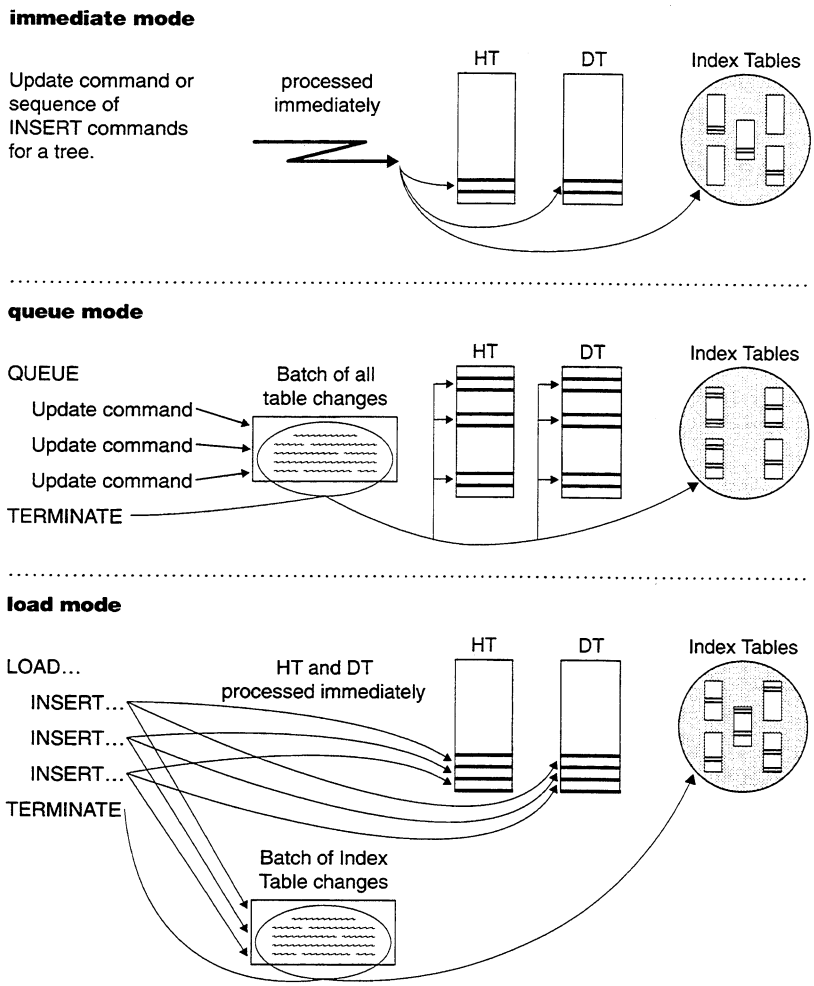
Illus. 2.37 on page 63 shows update processing in the three execution modes.

### Use of Table Space

Illus. 2.37 on page 63 shows that not only does the timing of updates differ among the execution modes, but also the placement of newly inserted data records in the Hierarchical Table and the Data Table depends on the execution mode. When you give an INSERT command in immediate mode or queue mode, the new data record utilizes reusable space in the Hierarchical Table and the Data Table. SYSTEM 2000 software keeps track of reusable space within the tables when a data record is deleted by a REMOVE TREE command. This reusable space can then be used to store a newly inserted record corresponding to the same schema record. However, in load mode, reusable space is not used for newly inserted data records. Instead, the software uses available space, which is space that was allocated to the tables, but was not used previously.

The next section discusses clearing updated pages from memory.

**Illus. 2.37** Update Processing in the Execution Modes



## Clearing Updated Pages

When a database is opened, the database tables reside as files on disk. SYSTEM 2000 software brings portions of these tables (pages) from disk to memory as they are needed whenever the database is accessed. Updates to the database are made on these pages in memory.

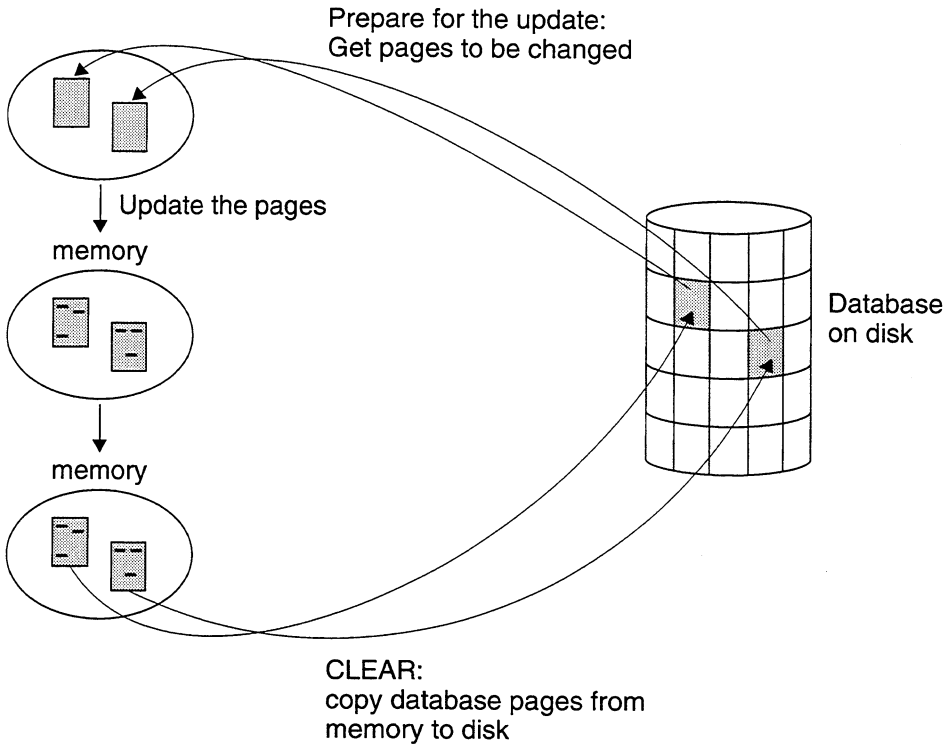
The SYSTEM 2000 buffer manager sends updated pages back to disk when it needs to make buffer space available for other pages required in PLEX processing. Copying from memory to disk is called clearing updated pages.

If a PLEX program terminates abnormally in a single-user environment due to a trapped error, SYSTEM 2000 software clears updated pages. But if the program terminates because of an untrappable error, the updated pages in memory cannot be cleared and the updates are lost. The result of such a termination is a damaged database. To reduce the chance of having a database become damaged, SYSTEM 2000 software provides two commands that you can issue to clear updated pages.

The CLEAR command copies memory-resident pages to disk. You can give the CLEAR command periodically to cause memory-to-disk clearing. Illus. 2.38 on page 65 shows the process of clearing updated pages. CLEAR AUTOMATICALLY clears memory-resident pages after each update is performed. When CLEAR AUTOMATICALLY is in effect, the software clears updated pages after each immediate mode update; otherwise, clearing is performed only when required by the buffer manager, as shown in Illus. 2.39 on page 66. Other commands, covered in the Reference part, also cause a clearing of database pages. The CLEAR and CLEAR AUTOMATICALLY commands clear database table pages and the Update Log buffers. Two other commands, CLEAR UPDATE LOG and CLEAR UPDATE LOG AUTOMATICALLY, are also available to clear Update Log buffers only. The mode of clearing is set by the form of CLEAR command that you specify.

When rollback is enabled, Coordinated Recovery is in effect, and the software clears the database pages only when it resets the Rollback Log.

**Illus. 2.38** Clearing Updated Pages




## The Damage Flag

As previously mentioned, a database becomes damaged when a program terminates abnormally and updated pages in memory cannot be cleared to disk. SYSTEM 2000 software maintains an internal flag, called the damage flag, to indicate whether a database is damaged.

When a memory-resident page is updated, the internal damage flag is set to indicate a damaged database. When updated pages are cleared, the damage flag is reset to show that the database tables reflect the updates.

When your program opens a database, SYSTEM 2000 software places the value of the damage flag in the COMMBLOCK for that database. By checking the value of that flag after opening a database, you can determine whether the database is damaged. If it is, SYSTEM 2000 software allows your program to retrieve from the database but updates are prohibited.

**Illus. 2.39** Using CLEAR and CLEAR AUTOMATICALLY

| | |
|---|---|
| . . . | |
| *(Immediate mode PLEX commands, including update commands)* | *(Buffer manager clears updated pages only when the buffer space they occupy is needed in PLEX processing.)* |
| . . . | |
| CLEAR. | *(clears updated pages)* |
| . . . | |
| *(Immediate mode PLEX commands, including update commands)* | *(Buffer manager clears updated pages as necessary.)* |
| . . . | |
| CLEAR AUTOMATICALLY. | *(clears updated pages)* |
| . . . | |
| *(Update command)* | *(Updated pages are cleared.)* |
| . . . | |
| *(Update command)* | *(Updated pages are cleared.)* |
| . . . | |
| *(Update command)* | *(Updated pages are cleared.)* |
| . . . | |
| CLEAR. | *(clears updated pages)* |
| . . . | |
| *(Immediate mode PLEX commands, including update commands)* | *(Buffer manager clears updated pages as necessary.)* |

### The Database Cycle Number

The database cycle represents an update operation carried out on the database. SYSTEM 2000 software keeps count of the number of cycles that were performed. This database cycle number is placed in the eleventh COMMBLOCK field for the database when the database is opened.

The cycle number is incremented by one after each completed immediate mode update operation. In queue mode and load mode, the cycle number is increased by one after the TERMINATE command completes its processing.

As your program performs update operations, it can display the value of the cycle number field, producing a running record of cycle number changes for later reference. If the current cycle of the database is not satisfactory, PLEX provides several ways to regain a previous cycle.

The next two sections discuss more considerations for restoring a database and for preserving its integrity.

## SAVING, RESTORING, AND RECOVERING DATABASES

This section discusses saving a copy of a database, restoring a copy of an archival database to disk for processing, and recovering a database after an error occurs during an update procedure.

Other important SYSTEM 2000 software features used in the updating and recovery of a database are the Update Log and the rollback process, both of which are discussed in this section.

### SAVE, RESTORE, and Update Logging

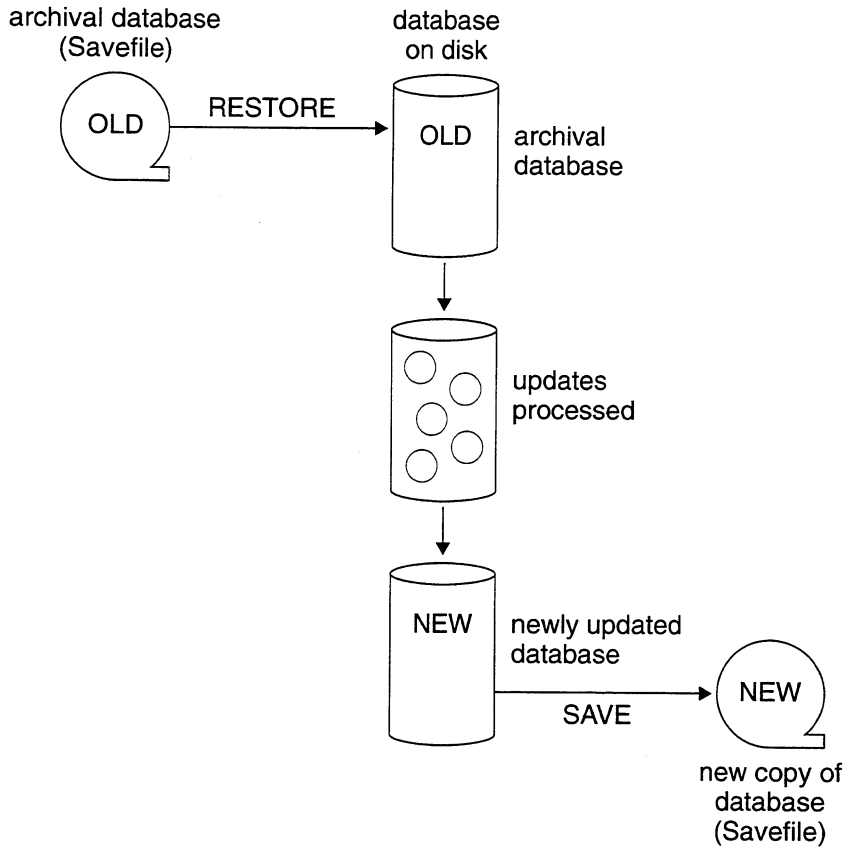
Preserving a database in its current and previous forms is an important procedure in database maintenance. PLEX provides a set of commands that you can use to save the archival copy of a database on disk or tape, to bring a copy to disk, and to keep an Update Log, which is a running record of updates for ready use in a recovery procedure.

The software saves an archival copy of the database (Savefile) when you give the SAVE command. You can copy the Savefile to disk with the RESTORE command. The basic use of SAVE and RESTORE is shown in Illus. 2.40 on page 68.

Both the SAVE and RESTORE commands can activate update logging. Processing a database begins with the RESTORE command, which copies the latest archival database to disk. Subsequent commands request portions (pages) of the database to be copied from disk to memory.

Once update logging is activated, it is available to collect the executed updates. Periodically during the updating, modified database pages are cleared from memory to disk, and occasionally you can issue a SAVE command to make an archival copy of the database as it stands on disk.

**Illus. 2.40** Restoring and Saving a Database



Between RESTORE and SAVE commands, the database might be changing because of update commands. If so, two other operations are also taking place. When the database is updated, the software increments the cycle number and, if update logging is activated, the software records processed updates. The Update Log collects these updates in a partially processed form. Updates in this form can be readily used by SYSTEM 2000 software later if the need arises to apply them.

When would it be necessary to apply those updates to a database? If, for example, erroneous updates are made in a database, the copy on disk contains the undesirable updates. Or if the database becomes damaged, the Update Log is a ready source of processed updates, so that update commands need not be reexecuted. The archival copy might not be immediately useful, either, since it might not contain any of the updates that were made this session, not even the satisfactory ones made before the erroneous ones. But the archival copy, which has no errors, can be used with its associated Keepfile to recover the database to a specific cycle.

First, the Update Log records go to the Update Log. They are written to the Keepfile when users issue the KEEP command. The commands for using the Update Log are described in the next section.

To recover a good version of the database, you can issue the RESTORE command to restore the archival copy of the database to disk. Then you can apply appropriate updates to that version. These updates are the ones performed since the archival copy was saved, up until the database cycle when the erroneous updates were made. The updates are applied from the Keepfile when you give the APPLY command. In a Multi-User environment, update operations by several users on the same database should be coordinated so that possible recovery operations are as systematic as possible.

## Using the Update Log

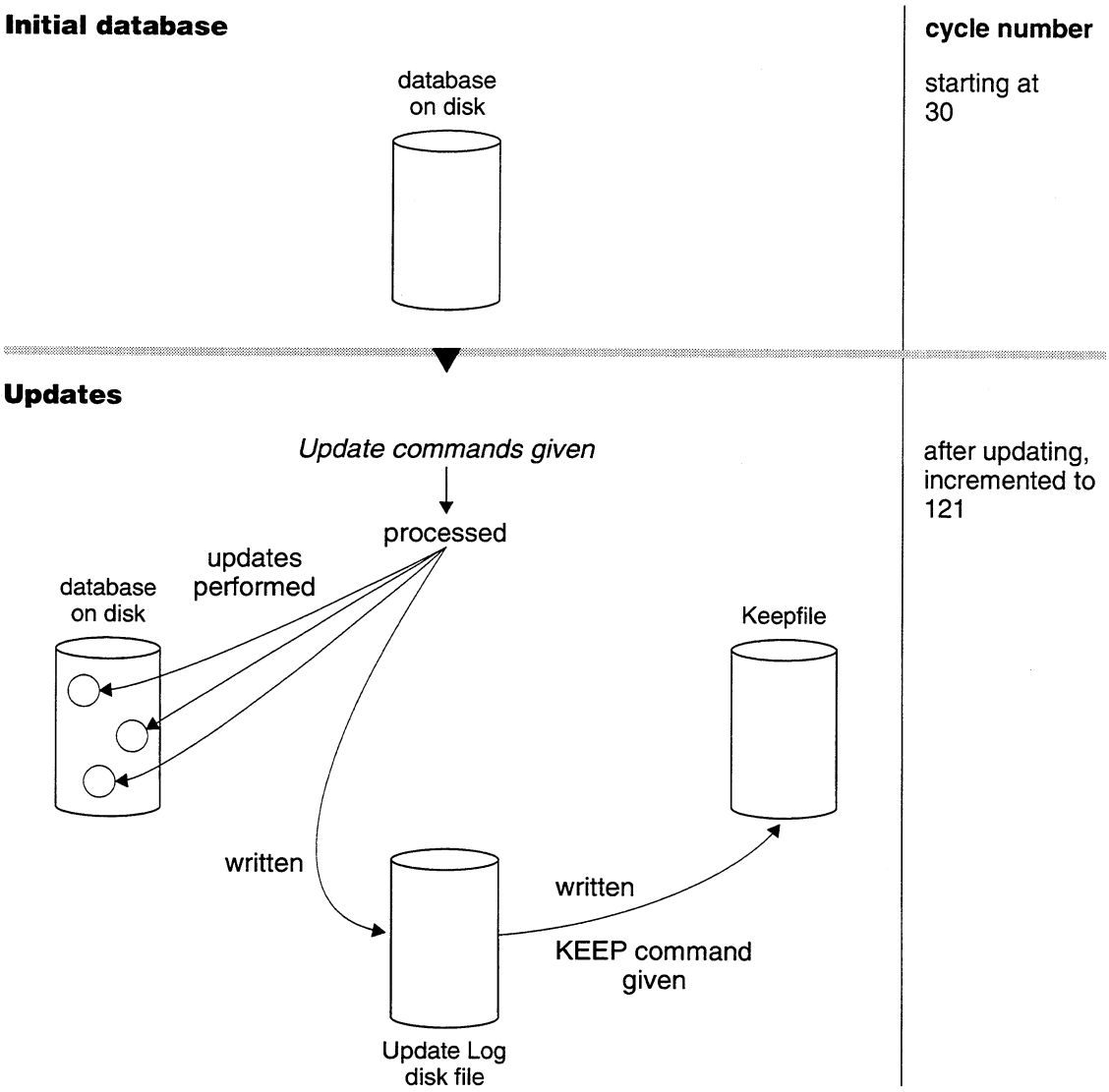
With four commands (KEEP, APPLY, CLEAR, and SUSPEND), PLEX programs can manipulate update logging during the updating of the database. The Update Log (database File 7) records processed updates. Periodically, users issue the KEEP command during a series of update operations to transfer the recorded updates to the Keepfile. This action ensures that the Keepfile holds the executed updates even in the event of a disk crash.

You can use the Keepfile in database recovery. When it becomes necessary to recover a database, you can restore an archival copy of that database to disk with the RESTORE command. Then you can issue an APPLY command to apply update cycles from the Keepfile as necessary to make the database current and accurate. Illus. 2.41 on page 70 shows recovering databases using the Update Log.

The CLEAR UPDATE LOG and CLEAR UPDATE LOG AUTOMATICALLY commands clear only the Update Log buffers for all open databases that have update logging activated. The CLEAR commands also set the mode of clearing. See **CLEAR Commands: Moving Updated Pages to Disk** on page 10-81 for the syntax and discussion of the CLEAR commands.

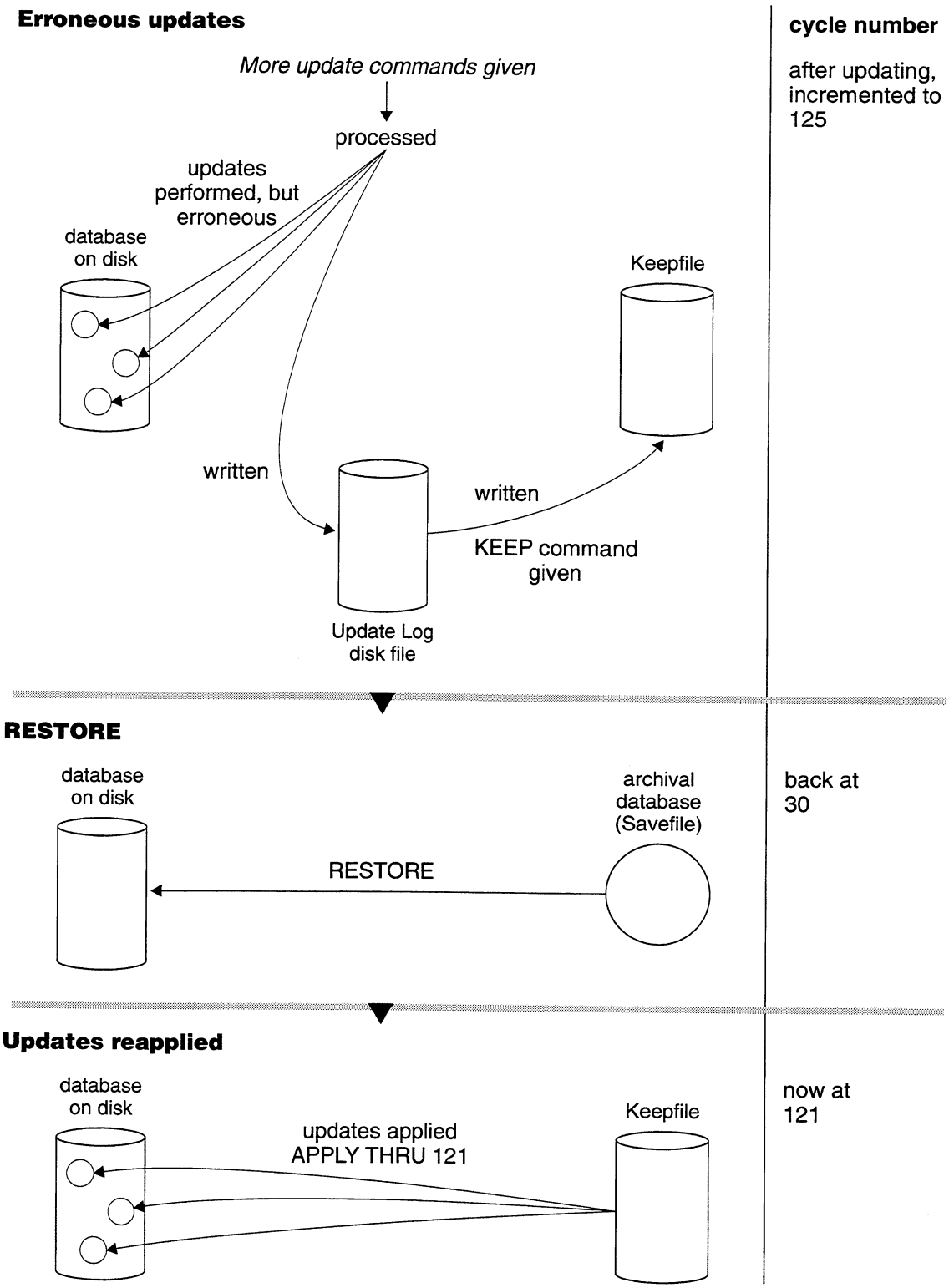
If an Update Log is no longer needed in the course of updating a database, you can issue the SUSPEND command to deactivate update logging. The SUSPEND command deactivates the Update Log, but not the Keepfile. The software does not record updates until update logging is activated again through a SAVE or a RESTORE command. Also, update logging is automatically suspended by restructuring or reloading the database.

**Illus. 2.41** Database Recovery with the Update Log




*continued on next page*

**Illus. 2.41** *continued*



## Using Coordinated Recovery

Until now, this chapter covered the process by which a program can recover a database using an archival database and a Keepfile. That processing starts with an earlier database cycle and brings the database up to a more recent cycle. But SYSTEM 2000 software also provides another kind of recovery, which recovers a database from damage automatically.

With Coordinated Recovery, SYSTEM 2000 software can automatically remove a user's erroneous updates while retaining the successful updates regardless of whether they were performed by that user or others. Coordinated Recovery uses both the Rollback Log and the Update Log. After rollback processing is performed on a database, the software automatically applies successful updates from the Update Log.

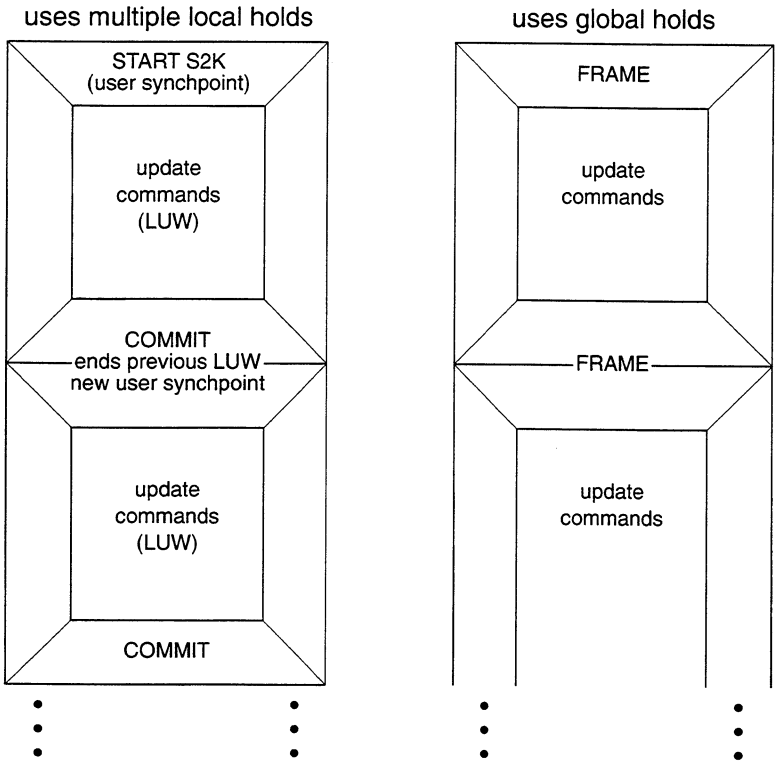
Coordinated Recovery can be performed on a database after the ENABLE ROLLBACK command is given for that database. In order for rollback to be enabled, update logging must be active. After rollback is enabled, a program issues commands between user synchpoints for a database.

User synchpoints are established by certain commands and operations (see the RESET ROLLBACK command). Updates that occur between user synchpoints are considered a *logical unit of work* (LUW) for a user. An LUW is committed permanently when a user synchpoint occurs. You can control this commitment with the COMMIT command or remove the LUW with the ROLLBACK command. The FRAME/END FRAME commands are also available.

Illus. 2.42 on page 73 shows two ways in which an LUW is formed. The latest database cycle before an LUW begins is the cycle to which the database can be reinstated using Coordinated Recovery processing. The software uses the Rollback Log to reinstate the database(s) affected by an uncommitted user LUW. The database(s) are reinstated to their before-image state at the time the last reset of the Rollback Log occurred. Then the software uses the database cycle information in the Master Record to reposition the Update Log and automatically applies updates associated with user LUWs, except for the uncommitted LUW being removed.

For more details about Coordinated Recovery, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*. Also, see the reference material in this manual for details about the COMMIT, ROLLBACK, and RESET ROLLBACK commands.

**Illus. 2.42** Two Ways to Frame a Logical Unit of Work (LUW)



As a program gives update commands between user synchpoints, each one becomes part of a logical unit of work, and certain database pages in memory are selected to be updated. The software copies these pages (before-images) to the database's Rollback Log before the update operation is actually executed. The Rollback Log holds the selected database pages as they appear without changes caused by update commands in the LUW. The Rollback Log contains the before-images of all updates associated with user LUWs until it is reset. Each database has only one Rollback Log, although it can be used by more than one user at a time.

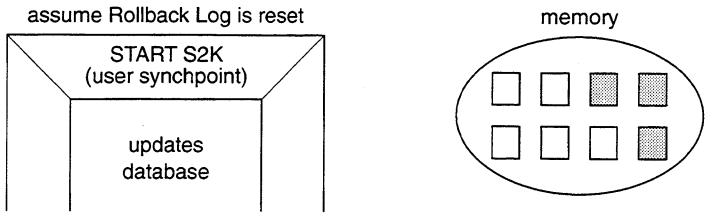
Once the before-image is recorded on the Rollback Log and the update is successful, a record of it is written to the Update Log. The first update after a user synchpoint establishes an LUW for that user. Any updates made to any database by the user before the next synchpoint for that user become part of that LUW. The LUW is considered to be uncommitted until the next synchpoint for the user. Once an LUW becomes committed, it cannot be removed by Coordinated Recovery. More than one user can have an uncommitted LUW on a database at one time, unless exclusive use or a global hold is in effect.

If for any reason a user's uncommitted LUW does not complete successfully and the database becomes damaged, Coordinated Recovery processing can use the before-images in the Rollback Log and the update cycles on the Update Log to remove the erroneous updates associated with that user's LUW. This processing takes place immediately and automatically in the case of a transaction failure. If the failure occurs because of an abnormal termination, the recovery takes place when the affected databases are physically next opened. Illus. 2.43 on page 75 shows the Coordinated Recovery process.

Issuing the COMMIT command is the most efficient means of committing designated updates. Because each user is allowed to have multiple local holds on specific records in a database (or several databases), the entire database need not be locked out to other users during update processing. The COMMIT command causes a synchpoint, commits an LUW (including any updates), and begins a new LUW. For a Multi-User environment with several users updating the same database(s), local holds and the COMMIT command ensure Coordinated Recovery of all data except the LUW that failed (was uncommitted) for a specific user. On the other hand, if you use the FRAME/END FRAME concept of LUWs, global holds are in effect.

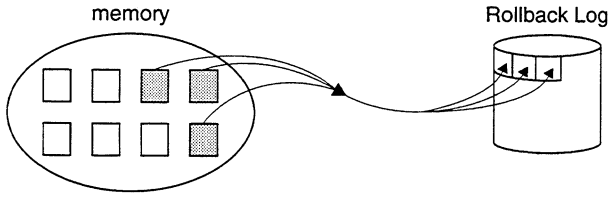
**Illus. 2.43** Example of Coordinated Recovery Processing

**1**



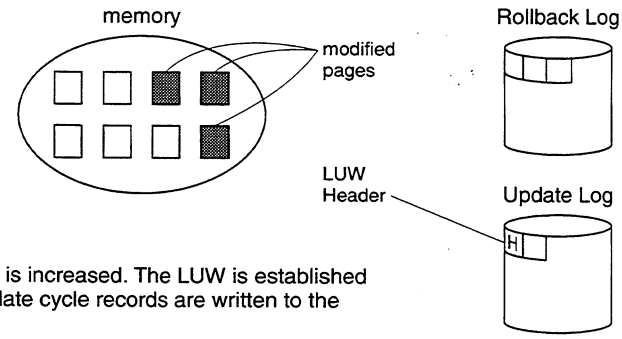

The update starts a logical unit of work (LUW). The pages to be updated are selected.

**2**



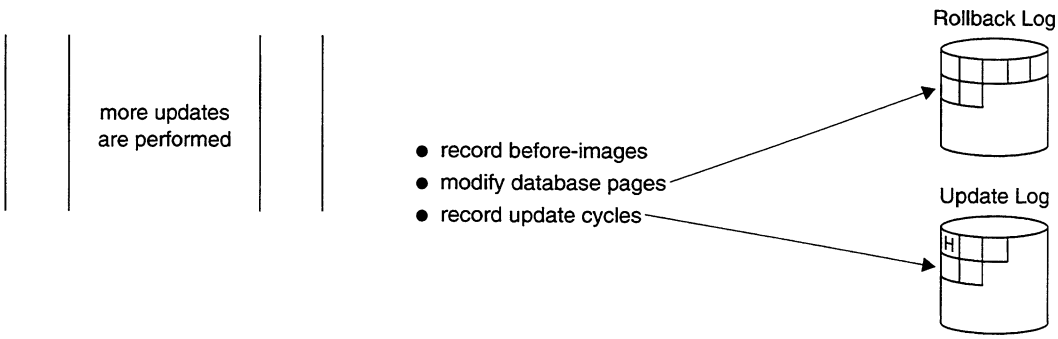

The selected pages are copied to the Rollback Log before the update is performed.

**3**




The update is performed, and the cycle number is increased. The LUW is established as uncommitted when the LUW header and update cycle records are written to the Update Log.

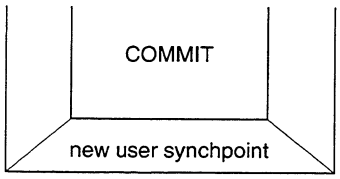
**4**



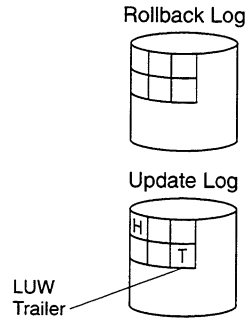

*continued on next page*

**Illus. 2.43** *continued*

**5**



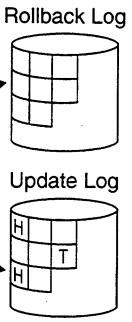

The COMMIT command ends the LUW (committed), updates are permanent, LUW trailer record written to Update Log; Log buffers are cleared; new user synchpoint; Rollback Log is not reset.




**6**



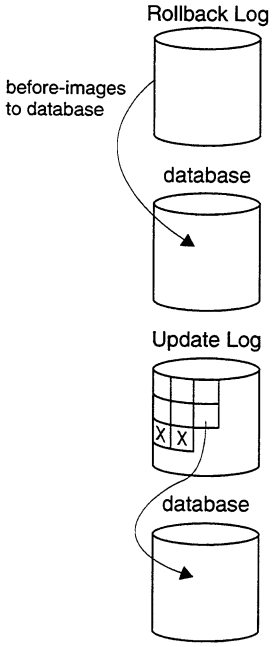

- before images to Rollback Log
- modify database pages
- record new LUW header and update cycles




**7**




- Update failure causes automatic recovery;
- Rollback Log is used to reestablish database to state of last reset;
- Database Master Record information is used to reposition Update Log to cycle at time of last Rollback Log reset;
- Updates associated with the previously committed LUW are applied to database; update records associated with the uncommitted LUW that is backed out because of failure are not applied and are marked as such.




**8** The database is recovered to state of last committed LUW; user processing can now continue. The ROLLBACK command causes the same process as Step 7 to occur. If a failure occured because of abnormal termination, the database is recovered automatically when it is next physically opened. Multiple users with multiple LUWs over multiple databases are recovered similarly.

# SECURITY AND INTEGRITY

This section presents some of the capabilities available in PLEX for the protection and integrity of data in a SYSTEM 2000 database.

## Passwords

Passwords are of three types. The password that creates the database (through the Self-Contained Facility) is the master password for that database. The master password holder can assign (with the SCF CONTROL language) a database administrator (DBA) password and up to 200 secondary passwords to users of the database.

Passwords control access to databases and to specific components of a database. They also restrict use of some commands.

## Access Authorities

With the SCF CONTROL language, the master password holder can assign authorities to valid secondary passwords for access to certain components of a database. The master password automatically has access to all components of a database. The DBA (Database Administrator) password is not assigned authority for any access to data.

There are four types of access authority that you can assign to secondary passwords for any component:

**R** The password holder can retrieve this component.

**W** The password holder can use this component in a where-clause.

**U** The password holder can update this component.

**N** The password holder has no authority for this component.

By assigning appropriate authorities for different components of a database, sensitive data can be protected from accidental or deliberate misuse.

A program places a password in the COMMBLOCK for the database to be accessed. When the database is opened, SYSTEM 2000 software checks this password to determine its type. If it is a secondary password, SYSTEM 2000 software checks the access authority of each component given in every command. The assigned authorities apply to all attempted accesses by the program until the database is closed or until another database is accessed. If an unauthorized access is attempted, an appropriate return code is placed in the COMMBLOCK for the database.

## Privileged Commands

Important administrative functions, such as saving or restoring databases and activating update logging, affect all users. Therefore, SYSTEM 2000 software requires the master password or the DBA password for these commands.

The following PLEX commands can be successfully executed only if the master password or the DBA password was placed in the COMMBLOCK when the database was opened:

```
SAVE
RESTORE
RELEASE
SUSPEND
APPLY
DISABLE/ENABLE  VALUES PADDING
[NO]  FULL PASSES
ENABLE/DISABLE ROLLBACK
RESET ROLLBACK
```

The master password holder can make these commands available to the DBA password by issuing the SCF ENABLE DBA FOR command. The DBA password can be authorized for all of these commands or just specific commands.

## Security by Entry

Security by Entry is another security option provided by SYSTEM 2000 software. This capability provides a lock-and-key arrangement for each logical entry in the database. A secondary password holder is permitted access to only one logical entry at a time and only if the key for the logical entry is presented.

To use Security by Entry, the master password holder declares a key item at level 0 as the Entry Key. Each data value for this Entry Key must be unique. A secondary password holder can access a logical entry only if its Entry Key value is presented as the first condition in the first where-clause. This feature is discussed in more detail in *SYSTEM 2000 CONTROL Language, Version 12, First Edition.*

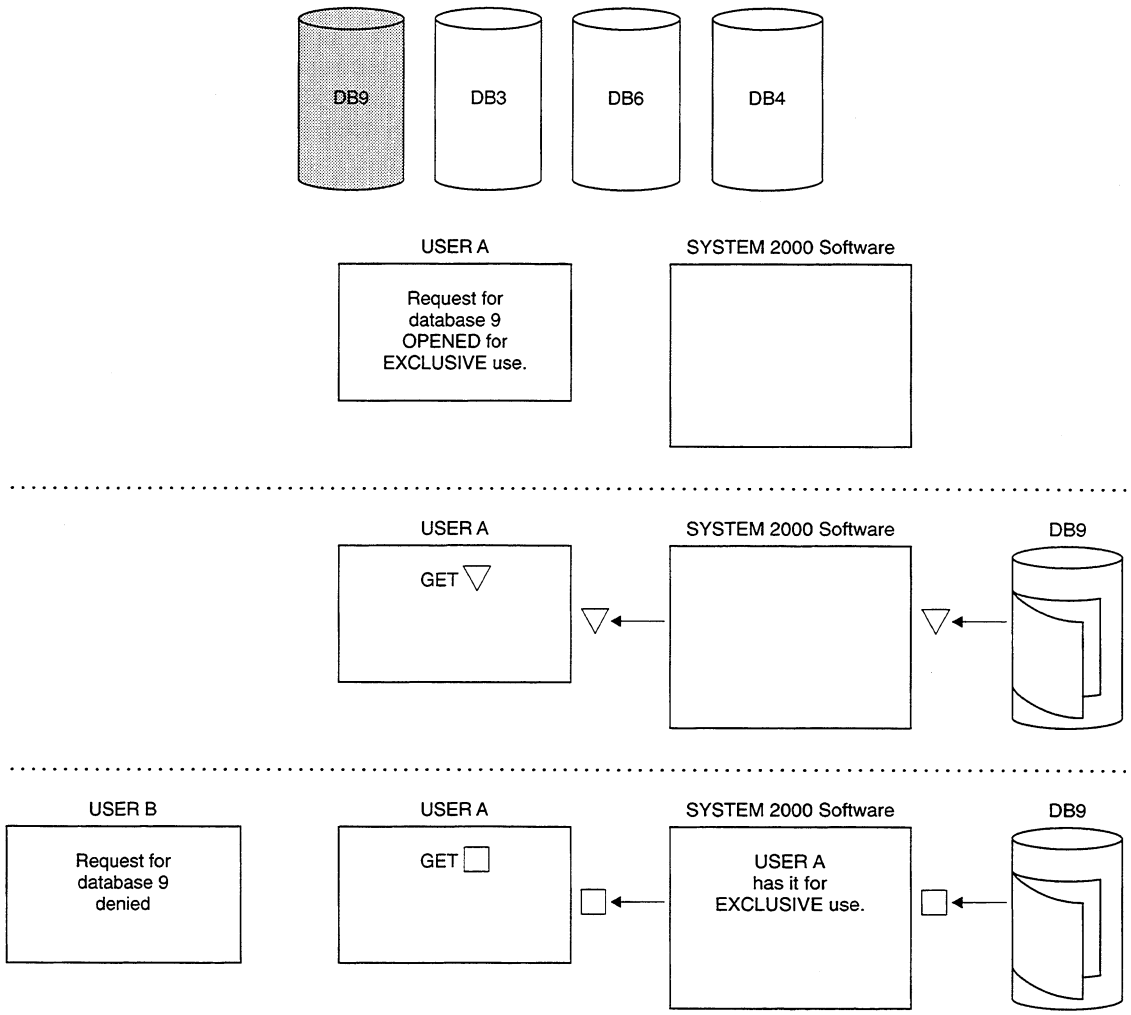
## Exclusive and Non-exclusive Use of a Database

In a Multi-User environment, many programs can use a database at the same time. This is called non-exclusive use. Sometimes the actions of one program affect those of another. For example, if one program releases the database files on disk while another is retrieving data, it could create undesirable results. Therefore, SYSTEM 2000 software requires exclusive use of a database for executing commands such as APPLY and RELEASE.

A program can obtain exclusive use of a database by restoring it from an archival copy. You can also specify the EXCLUSIVE option for a database in the OPEN command. The request for exclusive use of a database is granted when the database is not being used by any other user. When a database is opened for exclusive use, no other user can open (or access) it. Illus. 2.44 on page 79 shows an instance of exclusive use.

Even when it is not required by SYSTEM 2000 software, the PLEX user can use a database exclusively. For example, you can choose exclusive use to protect the results of a LOCATE command so that the database remains unchanged while the located data records are retrieved or updated. Non-exclusive use invites all users to update or retrieve concurrently. The retrieval and update requests are processed in a round-robin fashion.

**Illus. 2.44** Exclusive Use



## HOLD Option

When a database is being accessed under non-exclusive use in a Multi-User environment, a program can temporarily prevent other users from performing updates that could conflict with its own updates. With the HOLD option in a retrieval command (GET1, GET, GETA, GETD, GETP, or LINK), a program requests a hold on one record in the database. In immediate mode, the HOLD option signifies a request for a local hold, which prevents other users from obtaining their own local holds on the specific data record the program retrieves. In contrast, a queue mode HOLD signifies a global hold, which temporarily prevents other users from updating or obtaining local or global holds for that database. When a global or local hold is in effect, other users are still permitted to retrieve other records from the database.

Multiple holds on different records in the same database are allowed. All holds are checked for a deadlock situation. Also, an update to a hold record does not drop the hold on the record.

A local hold is useful when more than one user might be updating the same data record at nearly the same time. Consider the situation where two programs, A and B, both want to increment the value of item X by one, so that X is incremented twice. These operations could occur in the following sequence:

| Program A | Program B |
|---|---|
| GET . . . (*The retrieved value of* X *is* 15.) | GET . . . (*The retrieved value of* X *is 15.)* |
| ADD 1 TO X. (X *is incremented to* 16.) | ADD 1 TO X. (X *is incremented to* 16.) |
| (*With a PLEX command,* X *is updated in the database, with a value of* 16.) | (*With a PLEX command,* X *is updated in the database, with a value of* 16.) |

The result would thus be X = 16, although the intended outcome was X = 17.

Using the HOLD option to obtain a local or global hold before each update, the two programs would cause their operations to be coordinated as follows:

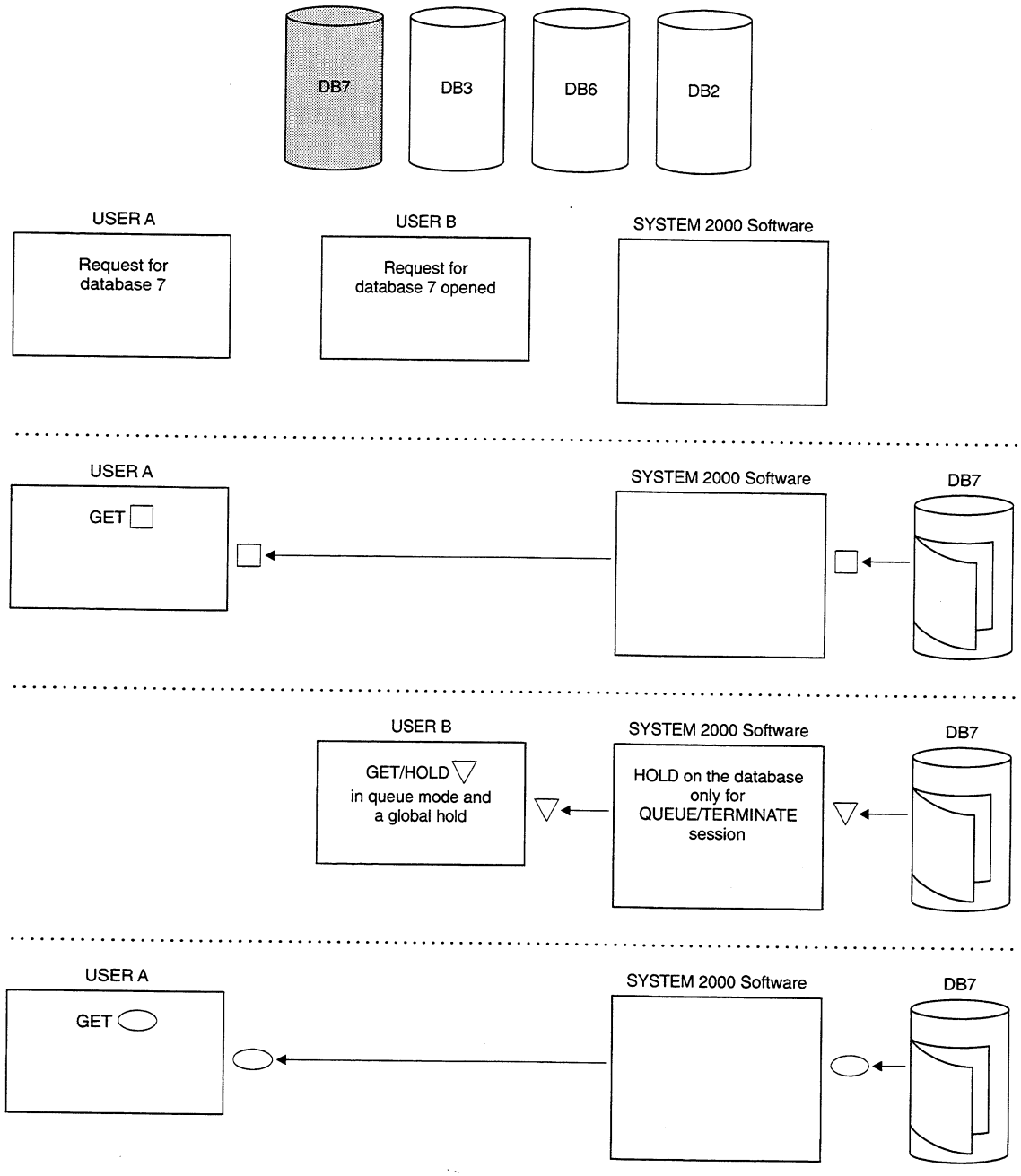
| Program A | Program B |
|---|---|
| GET/HOLD . . . | GET/HOLD . . . (*This hold is delayed until Program A's hold is dropped.*) |
| ADD 1 TO X. (X *is incremented to* 16.) (*By a PLEX command,* X *is updated in the database, with a value of* 16. *The update does not cause the local hold to be dropped.*) | (*Program B now retrieves the updated* X *and obtains its own hold.*) |
| | ADD 1 TO X. (X *is incremented to* 17.) |
| COMMIT: (*drops the hold*) | (*By a PLEX command,* X *is updated in the database, with a value of* 17.) |

The use of a global hold is shown in Illus. 2.45 on page 82. Illus. 2.46 on page 84 shows the use of a local hold.

In each case, the duration of a global or local hold depends on the operations the program performs after it obtains the hold. Details about obtaining and retaining local and global holds are given elsewhere in the Reference part.

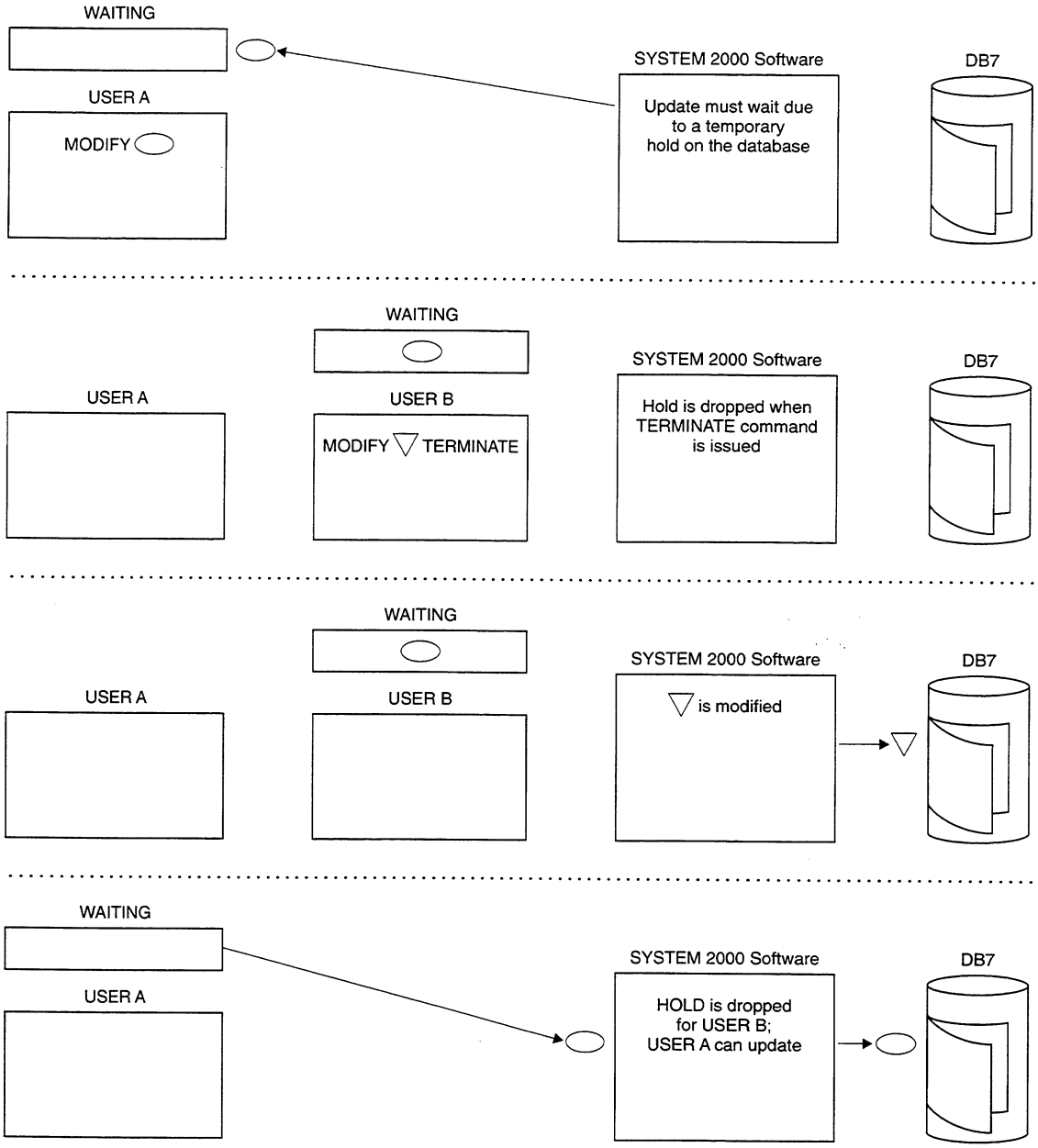
Use the HOLD option to ensure the integrity of data in a Multi-User environment if the value to be modified is a function of the current value. Also, use HOLD for updating an item used commonly by other users in where-clauses for Self-Contained Facility update commands. This would protect other users from the untimely selection or modification of invalid information. The Database Administrator can establish procedures for using the EXCLUSIVE, HOLD, and COMMIT options according to the best interest of the entire user group.

**Illus. 2.45** Use of a Global Hold

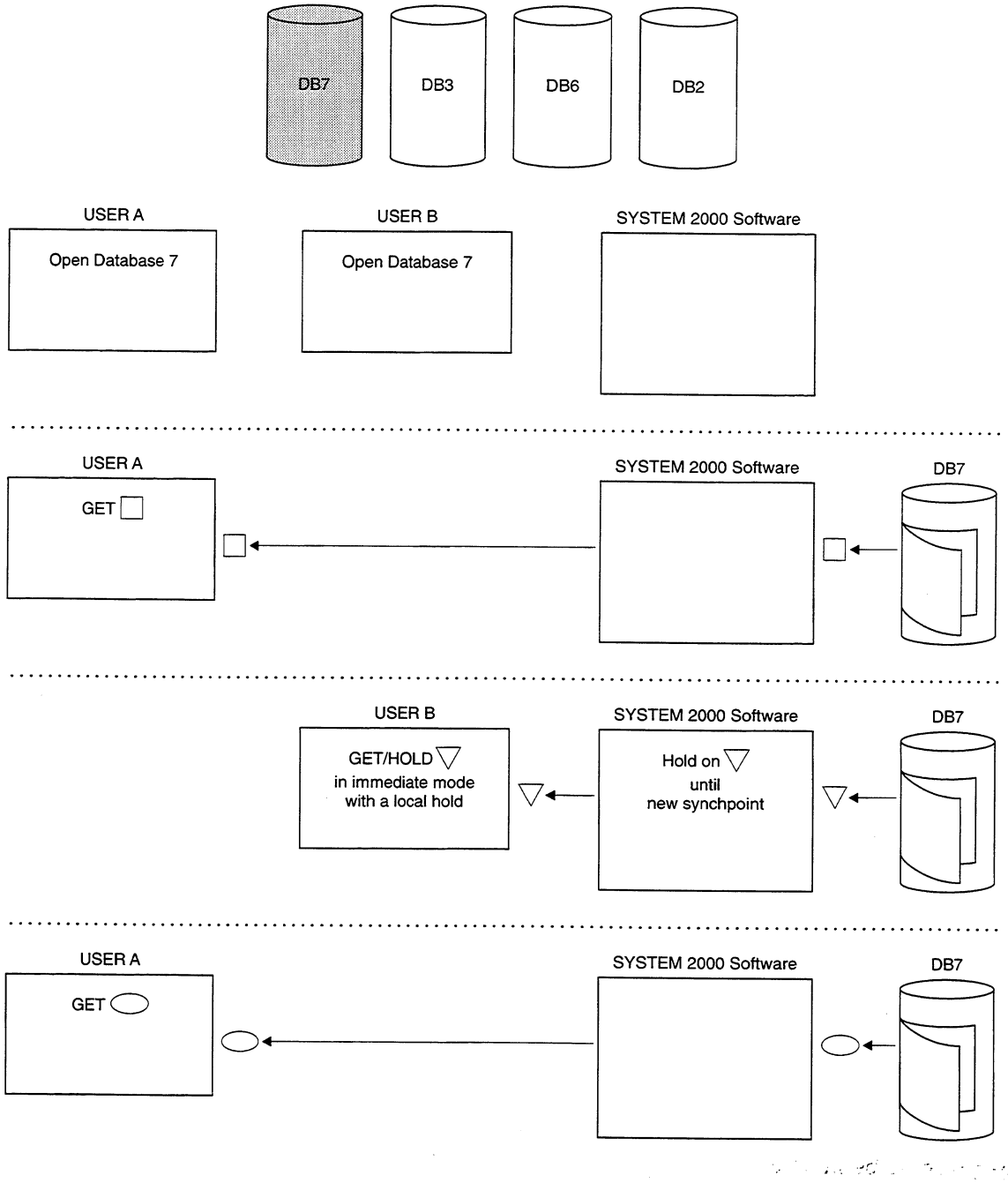



*continued on next page*

**Illus. 2.45** *continued*

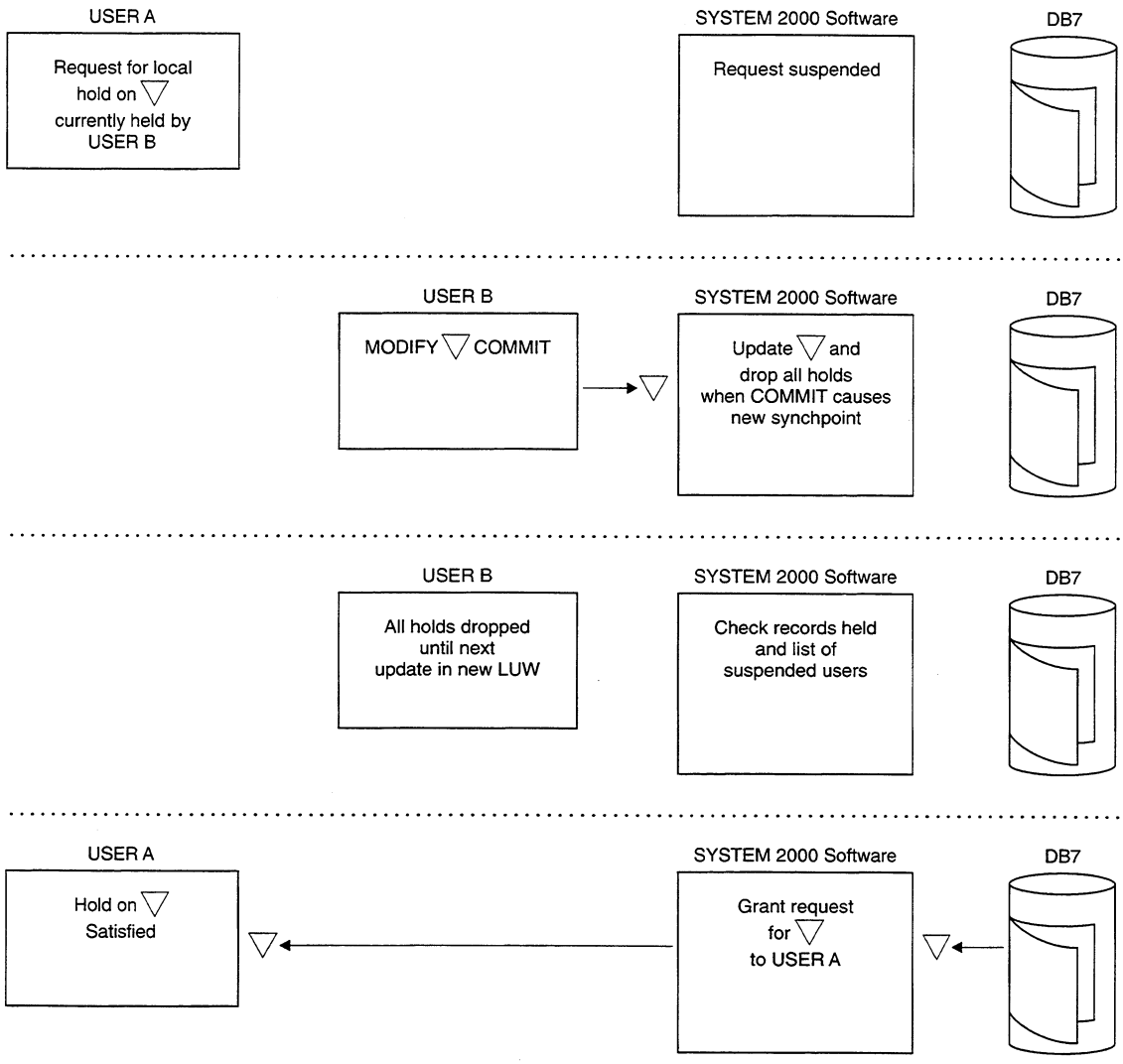


**Illus. 2.46** Use of a Local Hold




*continued on next page*

**Illus. 2.46** *continued*



# MULTIPLE DATABASES

The topics covered in this section are important when you are using more than one database in a PLEX program. The main considerations are

- the required COMMBLOCKs and subschema record descriptions
- the OPEN command
- the choice between exclusive and non-exclusive use in Multi-User environments
- the possibilities of retrieval using the LINK command with available stacks and Locate Files.

Each of these considerations is also discussed in the reference part of this book.

In a PLEX program, up to 63 databases can be open at a time in either a single-user or a Multi-User environment. You must declare a COMMBLOCK for each database that you want to access in the program. For COBOL, each COMMBLOCK must have a unique level 1 group-item name.

Although neither COBOL nor PLEX requires that an item name be unique outside a COMMBLOCK or subschema record, the program is more readable if item names are both meaningful and distinctive. The program can specify distinctive names for items with identical names in different database schemas. For example, perhaps the schemas of the EMPLOYEE and COMPANY databases both include an item called SSN. When these two items are used in the same program, they can be declared as subschema items with names like EMP-SSN and COMP-SSN. Names for certain COMMBLOCK fields, such as return code, can be specified in the same manner, such as EMP-RETURN-CODE and COMP-RETURN-CODE.

Before the program gives an OPEN command for a database, it must place an appropriate password in the COMMBLOCK for that database. A program can give a series of OPEN commands at the outset for all the databases it will use, especially if it needs to perform a series of operations with one database that depend on the results of operations with another. But if the operations with one database are not contingent upon other results, the program can give OPEN commands for just one or two databases to begin with, closing those databases with CLOSE commands before it opens any others.

If a program gives OPEN commands for several databases and accesses them with interleaved commands, it actually accesses only one database at any one time. The other opened databases are still available to the program for further access until the program closes them explicitly.

A program that has several databases open at the same time can refer first to one database, then to a second, and then back to the first. When the program begins a sequence of commands referring to the second database, the COMMBLOCK for that database must contain a valid password. Similarly, when the program later gives a command referring again to the first database, the COMMBLOCK for that database must contain a valid password. This password can be different from the one that originally accessed the database. (Several secondary passwords, each with differing authorities, might have been assigned to the database.)

A database can be opened for exclusive or non-exclusive use in a Multi-User environment. To open a database for exclusive use, the program can include the EXCLUSIVE option in the OPEN command, or it can use several other commands, as discussed in the Reference part of this manual. If a program acquires exclusive use of a database, that database is unavailable to any other SYSTEM 2000 user until the program closes it. But the program must explicitly close that database before it can open or get exclusive use of another. All databases opened for non-exclusive use must be closed by the program before another database can be opened for exclusive use.

A PLEX program can always use as many as 16 stacks. In addition, it can use up to 16 Locate Files, depending on the number given in the START S2K command. These stacks and Locate Files can be used as needed by program logic. For example, your program could use stacks (and Locate Files) 0 through 3 for one database and stacks (and Locate Files) 4 through 7 for another. This is useful, for example, in a cascade of linked retrievals, which is discussed next.

A command that usually uses several open databases is LINK. The retrieval conditions specified in a series of LINK commands can include items from several databases. A cascade of LINKs results in the retrieval of the specified components. The linking of retrievals from different databases establishes a number of Locate Files and stacks that contain entries from each of those databases.

A Locate File used for one database can later be used for another, and the same applies to stacks. A Locate File and a stack with the same subscript are paired as follows. If a stack and Locate File of the same number have the same database, a GET1 command reuses the stack but has no effect on the Locate File. Also, a LOCATE command has no effect on the stack. However, when a GET1 command refers to some database other than the Locate File's database, it not only reuses the stack but also clears the Locate File. Similarly, a LOCATE command for a database other than the stack's database clears the stack.

The next section discusses the commands used in the closing of databases and the termination of PLEX processing.

# TERMINATION

This section discusses the two kinds of termination in the PLEX portion of a program: closing a database and ending a SYSTEM 2000 session. A program might also have other termination steps of its own to carry out.

### Closing a Database

When your PLEX program finishes processing a database, issue the CLOSE command to close the database. This command clears the updated pages of that database and makes it unavailable to the program. If the program opens the database for exclusive use in a Multi-User environment, a CLOSE command frees it for use by other SYSTEM 2000 users. To get exclusive use of a database, a program must give CLOSE commands for any other databases currently open to it.

A CLOSE command should be issued for each database still open at the end of PLEX processing, especially if the program updated those databases.

### STOP S2K

At the end of all PLEX operations and after closing all databases that were used, give a STOP S2K command. In a Multi-User environment, this command clears the program's job from the Multi-User tables. It is the last PLEX command to be executed in a program.

The next section covers the structure of PLEX programs and the job steps in PLEX processing.

# PROGRAM STRUCTURE AND PROCESSING

This section describes what a COBOL PLEX program looks like and how it is processed. For PL/I, FORTRAN, and Assembler PLEX program structures and processing, see the appropriate appendix.

### COBOL PLEX Program Structure

The structure of a COBOL PLEX program is similar to that of any COBOL program. It includes, among other things, a DATA DIVISION that describes COMMBLOCKs and subschema records and a PROCEDURE DIVISION containing executable PLEX commands. COMMBLOCKs and subschema records can be included in the WORKING-STORAGE SECTION or the LINKAGE SECTION of the DATA DIVISION. In the PROCEDURE DIVISION, executable PLEX commands can appear as simple statements, within paragraphs, and within IF statements, that is, wherever executable COBOL statements can appear.

You can pass COMMBLOCKs and subschema records from a program to a subprogram. Subschema items can take initial values. The fields in a COMMBLOCK or a subschema record can be used in any COBOL statement in the program, as can any other COBOL variables.

Since PLEX supports modular and structured programming, it is a proper extension of COBOL.

## PLEX Processor Directives

A COBOL PLEX program is preprocessed by the COBOL PLEX processor before the program is compiled. The PLEX processor generates COBOL statements from the PLEX commands.

In addition to the PLEX commands and the COBOL statements, a PLEX program can contain another class of statements called PLEX processor directives. Unlike the PLEX commands, the PLEX processor directives do not produce any COBOL code. For example, the $BLOCK directive, which is required in every PLEX program, instructs the PLEX processor to generate the description of the S2KDUM control block. In general, the directives provide a means for you to communicate with the PLEX processor and direct its actions in certain ways (see **COBOL PLEX Processor Directives** on page 19-2).

## Job Steps

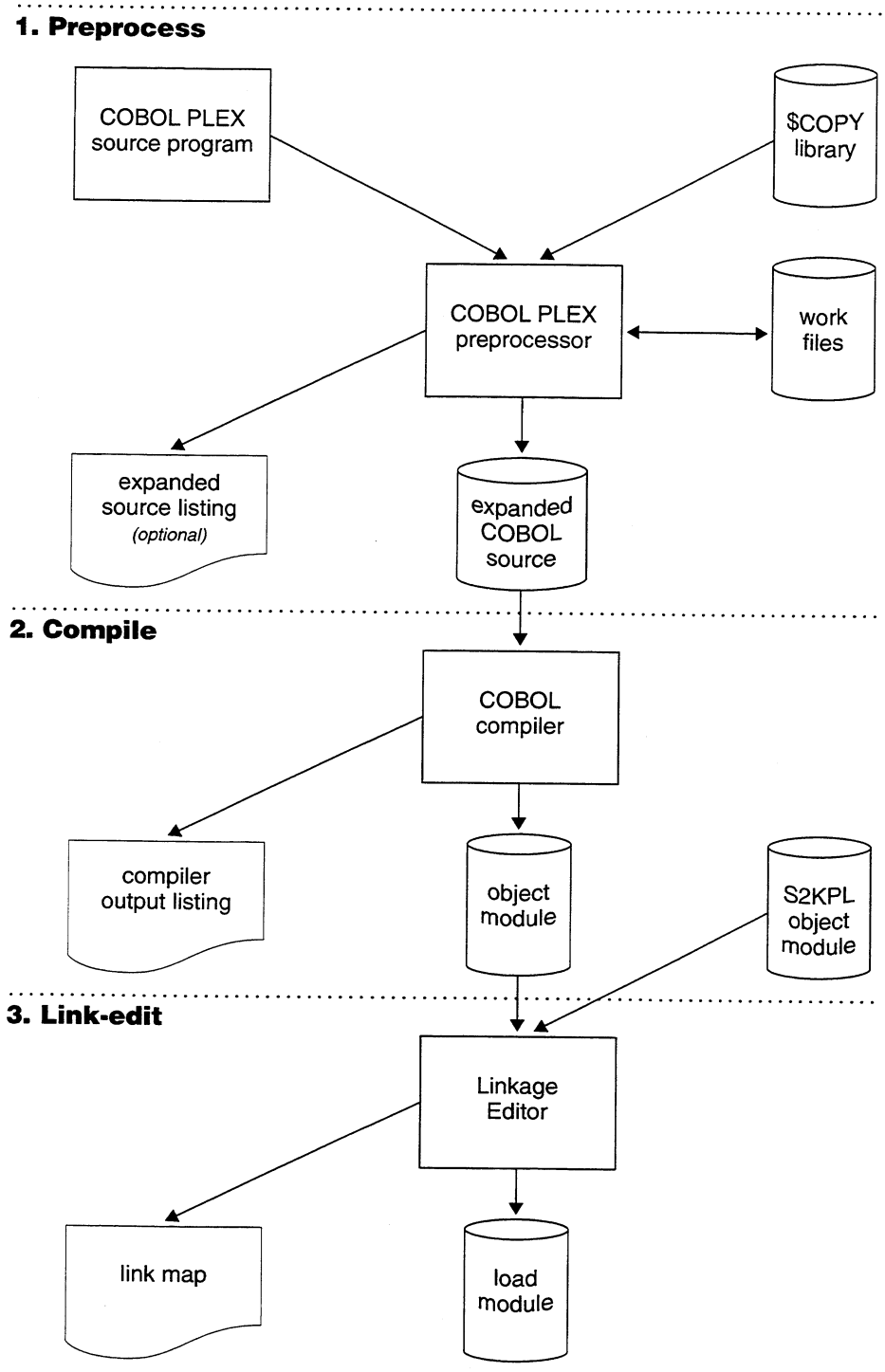
Every COBOL program must be compiled and link-edited before it can be executed. PLEX adds one more job step at the front end, namely, executing the PLEX processor. Illus. 2.47 on page 90 shows the job steps before execution: a program must be processed by the PLEX processor, compiled, and link-edited with the SYSTEM 2000 interface (S2KPL). $COPY is a PLEX processor directive that you can use to copy COMMBLOCK and subschema record descriptions into a source program from members in a source library (instead of including them in the source code).

Illus. 2.48 on page 91 shows a PLEX program in memory in a single-user environment. The module S2KPL, which you link, brings in the run-time interface S2KPLR, which loads the module S2KPLI, the single-user version of SYSTEM 2000 software. In a Multi-User environment, S2KPLR loads a module called MUPLINT and one copy of SYSTEM 2000 software, common areas, and buffers that are all shared by two or more users, as shown in Illus. 2.49 on page 92.

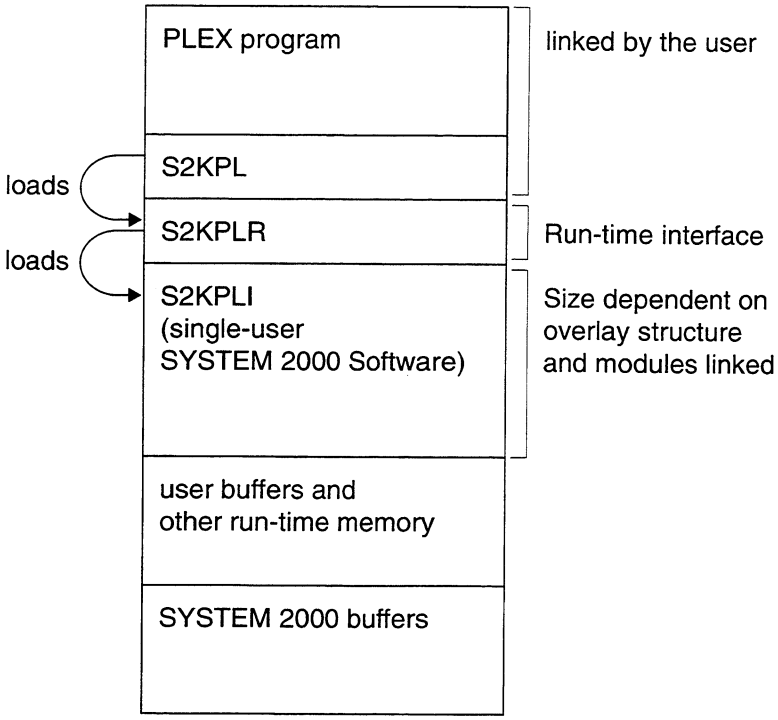
PLEX program modules can be processed by the PLEX processor separately or in a batch. COBOL modules without PLEX commands can be mixed with COBOL PLEX modules in a program, but those modules should be compiled directly without the PLEX processing step.

Since there are different steps involved in a PLEX job, an error might occur at any step. The PLEX processor ignores all statements except the PLEX commands. Only PLEX syntax errors can be detected at the PLEX processing step. The PLEX processor generates syntactically correct COBOL code. However, any COBOL syntax error in the source code is detected only at the compilation step. All errors in referencing SYSTEM 2000 data records and items are not detected until execution of the program when the database is actually accessed.

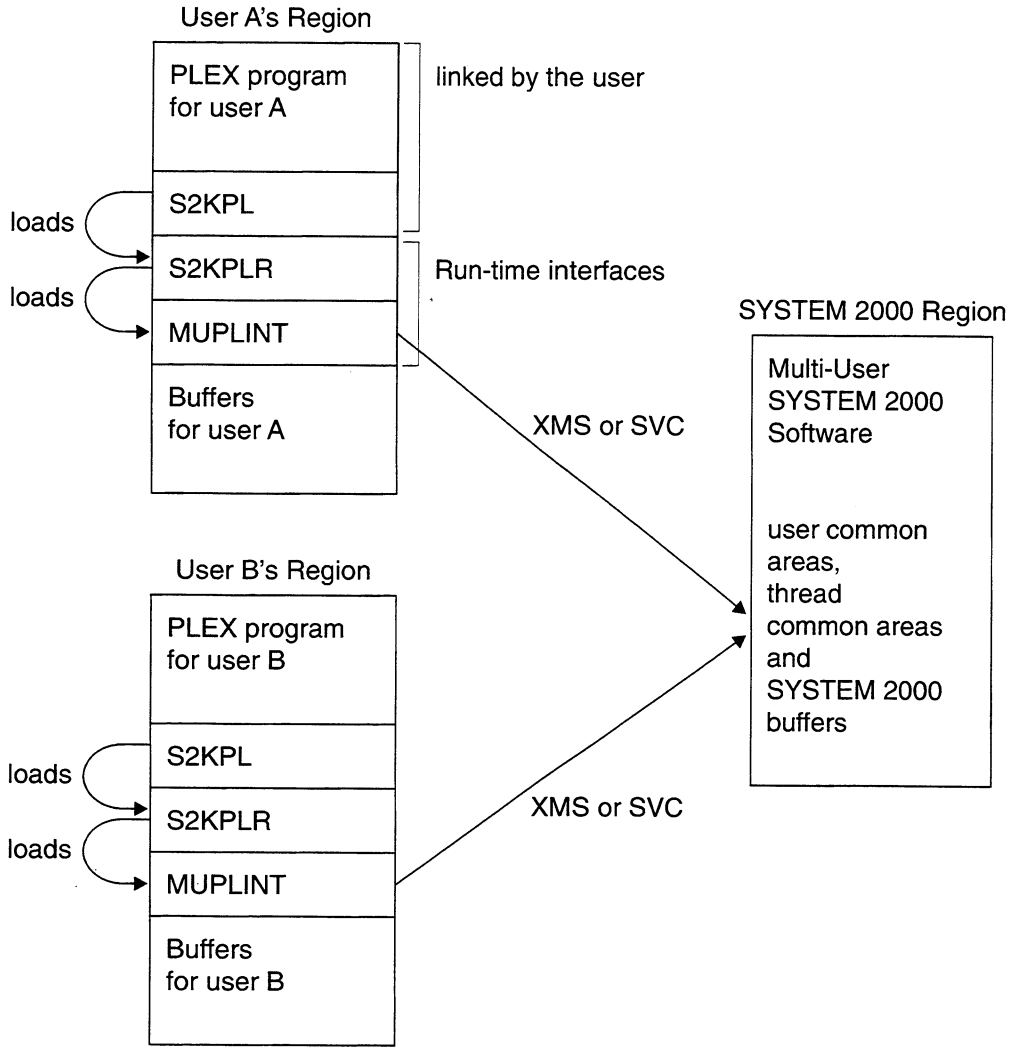
**Illus. 2.47** Job Steps Before Execution



**Illus. 2.48** PLEX Program in a Single-User Environment



**Illus. 2.49** PLEX Program in a Multi-User Environment



# REFERENCE

# PLEX COMMAND REFERENCE

Introduction

SYSTEM 2000 Item Types and
COBOL Data Descriptions

COMMBLOCK and SUBSCHEMA RECORD Descriptions

Initialization

Where-Clauses

Retrievals

Execution Modes

Updates

Saving and Restoring

FOR

ENABLE/DISABLE VALUES PADDING

FULL PASSES

$TIME

ENABLE ROUTINE and DISABLE ROUTINE

Termination

COBOL PLEX Processing

Sample COBOL PLEX Programs

Populating a Database

Usage and Guidelines

Chapter 3

# Introduction

*FORMAT OF COBOL PLEX COMMANDS 3-1*
*FORM and NOFORM 3-2*
*Continuation of a Command 3-3*
*Comments 3-3*
*One Command per Line 3-3*
*Reserved Words 3-3*

The material in these reference chapters gives the complete syntax of each PLEX command, along with examples and illustrations. Before using this reference material, become familiar with the concepts for these commands, which are introduced in the first part of this manual.

The reference material for each command includes a list of considerations, which contain usage rules and suggestions. They might also include unusual characteristics or other important information about the command.

Following the general discussion of each command, each important syntactic unit of the command is described in detail. For example, following the general discussion for the GET1 command, separate topics discuss the options FIRST, NEXT, LAST, PRESENT, and S2KCOUNT in detail, and GET1 with a where-clause. Also, separate sections are devoted to major syntactic units (for example, the where-clause, the inclusion list, and the HOLD option) that are common to several commands.

## FORMAT OF COBOL PLEX COMMANDS

This section describes general rules and conventions for coding COBOL PLEX programs, which look similar to ordinary COBOL programs. For details about coding FORTRAN, PL/I, or Assembler programs, see the appropriate appendix.

The PLEX processor scans a program for PLEX statements and commands, turns some statements and commands into COBOL comments, and generates COBOL statements to carry out the requested processing. As the PLEX processor scans, it expects to find PLEX statements and commands in a prescribed format.

In general, each executable PLEX command consists of an initial keyword, a database identifier or subschema record name, and one or more other specifications or program variable names. A PLEX command can extend over more than one line (see **Continuation of a Command** on page 3-3). You cannot divide the keyword, identifiers, and other names across adjacent lines of the command. If you divide a command word, the PLEX processor considers it to be two words.

The placement of a PLEX command on a line depends on the specifications in the SORMGIN execution option (see **SORMGIN Execution Option** on page 19-19), if you used that option. If you do not use SORMGIN, the default placement columns are as follows:

- If you use the NOFORM execution option, a PLEX command can appear anywhere between columns 7 and 73. (See the next topic.)
- If you use the FORM execution option, a PLEX command must begin in column 7.

Columns 73-80 can contain program identification information. The PLEX processor does not use this portion. Depending on specifications in the SEQID and SORMGIN execution options, the PLEX processor can generate sequence numbers for source statements. (See **SEQID Execution Option** on page 19-18 and **SORMGIN Execution Option** on page 19-19.) If you use the defaults, the PLEX processor counts the source lines and puts the sequence numbers in columns 73-80 on the expanded source listing.

An executable COBOL PLEX command must end with either a period or a semicolon. The PLEX processor places the period or semicolon at the end of the sequence of statements it generates for the command. For example, you can use a semicolon to terminate a PLEX command embedded in an IF statement. The generated code from the PLEX processor will end with a semicolon and will not interrupt the IF statement.

**Example**

```
IF DAMAGE-FLAG IS EQUAL TO 0
MODIFY SALARY-REC;
ELSE
    PERFORM DATA-BASE-DAMAGE-REPORT.
```

## FORM and NOFORM

Because you mix PLEX commands with COBOL statements in a program, the PLEX processor must recognize these two kinds of input and process only the PLEX commands. The PLEX processor offers a choice of two processing methods.

One method utilizes the NOFORM execution option (the default). (See **COBOL PLEX Processor Execution Options** on page 19-10 for a description of COBOL PLEX processor execution options.) With the NOFORM option, the PLEX processor examines the first syntactic unit of each line. If it is not a PLEX keyword, the PLEX processor assumes the line contains only COBOL statements. If the first word is a PLEX keyword, the PLEX processor looks further for other PLEX keywords making up a command. The NOFORM format is the most commonly used format.

The other processing method, using the FORM execution option, is somewhat more restrictive. With the FORM option, each PLEX command (including the COMMBLOCK and SUBSCHEMA RECORD statements) must begin in column 7, unless otherwise specified in the SORMGIN execution option. After the PLEX processor finds the beginning of a PLEX command, it scans the rest of the line (and possibly subsequent lines) until it finds the command terminator (a period or semicolon, as mentioned above). PLEX commands that do not begin in the correct column are not scanned by the PLEX processor. Lines not scanned are assumed to be COBOL statements and are passed directly to the COBOL compiler. The FORM execution option is supported only for compatibility with earlier releases of PLEX.

**Examples**

```
GET SSR1 NEXT.            (Col. 7 is acceptable for FORM or NOFORM)
COMMBLOCK OF EMPLOYEE.    (Col. 7 is acceptable for FORM or NOFORM)
  GET1 SSR3 FIRST.        (Col. 8 is acceptable for NOFORM only)
```

## Continuation of a Command

You can continue a PLEX command on several lines. Most PLEX commands refer to a database either by the database name, by a COMMBLOCK name (for example, in the OPEN command), or by a subschema record name (as in the retrieval and update commands). One of these three names must appear on the first line of a command that has continuation lines.

When you use the FORM execution option, start each PLEX command in the correct column (column 7 by default); continuation lines do not need to begin in that column.

**Examples**

```
GET SSR1      (starts in Col 7)
 FIRST.       (starts in Col 8; acceptable)
GET SSR1      (starts in Col 7)
  FIRST.      (starts in Col 9; acceptable)
```

## Comments

You can intersperse comments among PLEX commands. Also, comments can appear among the item description lines in a COMMBLOCK or SUBSCHEMA RECORD description.

## One Command per Line

Each PLEX statement and command must appear on a separate line. This rule also applies to item descriptions in COMMBLOCKs and subschema records and to EQ and FORM statements that appear with subschema item descriptions. Also, PLEX commands and COBOL statements cannot be mixed on the same line.

## Reserved Words

The PLEX processor generates and uses the following names as variable names. Therefore, do not use these names as variable names in your PLEX program.

```
S2K
S2KCOUNT
S2KDUM
S2KDU1, S2KDU2, . . , S2KDU12
S2KLKH
S2KRTC
S2KSRT
S2A1, S2A2, . . . , S2A999
S2F1, S2F2, . . . , S2F999
S2R1, S2R2, . . . , S2R999
```

**Note**: S2KPL is the name of the routine that is called to process PLEX commands. Therefore, do not use it as the name of a PLEX program, subprogram, or entry point.

If a PLEX command or PLEX processor directive begins with any of the following keywords, the PLEX processor recognizes it only if the syntactic unit following the keyword is correct:

```
CANCEL          OPEN
CLOSE           RELEASE
DISABLE         RESET
ENABLE          START
END             STOP
NO              TERMINATE
```

If there is an error in one of these commands, the PLEX processor passes the command directly to the compiler. For example, a spelling error in the database name in an OPEN or a CLOSE command results in a COBOL error because the PLEX processor ignores the command as it would for a COBOL OPEN or CLOSE statement. The PLEX processor uses the following PLEX keywords to identify PLEX commands or directives. Therefore, these words cannot be used as the first word in any programming statement:

```
APPLY                          $ALLOCATE
AUTOMATICALLY                  $BLL
CLEAR                          $BLOCKS
COMMBLOCK                      $CICS
COMMIT                         $COPY
COPYL                          $DEBUG
COPYS                          $DELAY
DROP                           $ICOM
FOR                            $NAME
FRAME                          $SNAP
FULL                           $STORAGE
GET                            $S2KDUM
GETA                           $TIME
GETD
GETP
GET1
KEEP
LINK0, LINK1, . . . , LINK9
LOAD
LOCATE
MODIFY
ORDER
QUEUE
REMOVE
RESET
RESTORE
ROLLBACK
SAVE
SCHEMA
SUBSCHEMA
SUSPEND
```

See the appendices for restrictions on other programming languages.

Chapter 4

# SYSTEM 2000® Item Types and COBOL Data Descriptions

*INTRODUCTION 4-1*

*ITEM TYPE TEXT 4-2*

*ITEM TYPE CHARACTER 4-2*

*ITEM TYPE DATE 4-3*

*ITEM TYPE INTEGER 4-3*
*COMP-3 Usage 4-3*
*COMP Usage 4-4*
*DISPLAY Usage 4-5*

*ITEM TYPES DECIMAL AND MONEY 4-5*
*COMP-3 Usage 4-5*
*COMP Usage 4-6*
*DISPLAY Usage 4-6*

*ITEM TYPES REAL AND DOUBLE 4-7*

*ITEM TYPE UNDEFINED 4-7*

*ITEM TYPES AND COBOL DATA DESCRIPTIONS OF ITEMS IN WHERE-CLAUSES 4-7*
*WORKING-STORAGE Variables as Right Operands 4-8*
*Subschema Items as Right Operands 4-8*

*CONSIDERATIONS 4-9*

*SUMMARY CHART 4-10*

## INTRODUCTION

SYSTEM 2000 software offers nine different item types: CHAR, TEXT, UNDEFINED, DATE, INTEGER, DECIMAL, MONEY, REAL, and DOUBLE. Since all data transfer between a PLEX program and a database takes place through a subschema record, each subschema item must have a usage and picture compatible with the item type of the schema item it represents. SYSTEM 2000 software offers data independence by converting the data stored in an internal form to a program-understandable form for retrievals, updates, and comparisons in a where-clause. This chapter gives a brief description of the SYSTEM 2000

item types and their correspondences with COBOL usages and pictures. Illus. 4.1 on page 4-10 is a summary of all rules for COBOL data descriptions. See the appropriate appendix for FORTRAN, PL/I, and Assembler data types.

When you define a SYSTEM 2000 database, you specify the picture and the size of a schema item, along with its type. This example lists some component numbers and names with their item types and pictures as they might appear in a database definition.

**Example**

```
2*     LAST NAME (CHAR X(10))
5*     BIRTHDAY (DATE)
6*     SECURITY CLEARANCE (INTEGER NUMBER 999)
11*    ACCRUED VACATION (DECIMAL NUMBER 999.9)
111*   PAY RATE (MONEY $9999.99)
132*   COMMENT (TEXT X(7))
```

# ITEM TYPE TEXT

A value of item type TEXT is a string of characters including alphabetic, numeric, special characters, and blanks. In the example, COMMENT is of item type TEXT, and seven bytes are reserved for each value of this item in the Data Table. When a TEXT value is shorter than its picture size, the data are stored left-justified and filled with trailing binary zeros. When the value is longer than its picture size, the number of characters stored in the Data Table is four less than the picture size. The last four bytes contain a pointer to the Extended Field Table, where the rest of the characters are stored. The picture size must be at least four for the value to be able to use the Extended Field Table.

The COBOL usage for a TEXT item is DISPLAY. When SYSTEM 2000 software retrieves in the subschema record a TEXT value from a database, the value is left-justified and either blank-filled or truncated on the right. If the item is null, the field is filled with binary zeros.

When the software takes a TEXT value from a subschema item field for an update, it scans from left to right for the value. The scan is terminated either by the end-of-field specified in the SUBSCHEMA RECORD description or by a byte of binary zeros. Therefore, if the first byte contains binary zeros, the item is considered null.

# ITEM TYPE CHARACTER

Item type CHARACTER (CHAR) is like type TEXT, except that any extraneous blanks (leading, trailing, or multiple imbedded) are removed from the values. That is, ' JOHN SMITH ' is stored as 'JOHN SMITH'. Short and long CHAR values are stored like those of type TEXT.

# ITEM TYPE DATE

DATE values are special numeric data; only dates written in conventional forms are qualified to be of this type. DATE type values must be in the range January 1, 1500, to December 31, 2499. SYSTEM 2000 software allows arithmetic calculations on dates as on other numbers. For COBOL, a subschema item corresponding to a DATE item should have DISPLAY for its usage and either Xs or 9s for its picture. It can be either six or eight bytes long. The DATE value can be placed in this field in six possible formats:

```
MMDDYYYY     (default for 8-character dates)
MMDDYY       (default for 6-character dates)
DDMMYYYY
DDMMYY
YYYYMMDD
YYMMDD
```

The format for a DATE item can be specified with the keyword FORM in the description of the corresponding subschema item or is selected by default. MMDDYYYY is the default for a subschema item that is eight bytes long. The default is MMDDYY for a six-byte field. If a date format is specified and the field length is other than six or eight bytes, the PLEX processor issues an error message.

The value representing a date must be placed in its field in the specified format. SYSTEM 2000 software issues return code 10 if the value for an update or a where-clause does not conform to the date format. For retrievals, the software places the DATE value in the specified format in the corresponding field in the subschema record. Binary zeros represent null DATE items.

# ITEM TYPE INTEGER

In SYSTEM 2000 software, signed integer numbers are of item type INTEGER. They are stored in packed decimal representation in the database. For COBOL, the corresponding field in the program can have COMP-3, COMP, or DISPLAY usage. During a retrieval, SYSTEM 2000 software converts the data item, if necessary, from the internal packed decimal representation to the representation for the declared usage and moves the converted value to the appropriate field. For an update or a where-clause comparison, it converts the value in the subschema record, if necessary, to packed decimal representation. (See **Item Types and COBOL Data Descriptions of Items in Where-clauses** on page 4-7.)

## COMP-3 Usage

When an INTEGER type item has COMP-3 usage in a subschema record, no conversion is necessary since the representation of the number is packed decimal in both situations. Therefore, data transfer in this situation would be considerably faster than with the other usages.

The size of the COMP-3 field (total number of bytes) must be the same as the size of the corresponding INTEGER field in the database. There must not be any digits to the right of the decimal point in the COMP-3 field. A valid data description for an item defined as SECURITY CLEARANCE (INTEGER NUMBER 999) would be PICTURE 999 COMP-3, PICTURE 999V COMP-3, or PIC 9(3) COMP-3.

When a subschema record is referenced for the first time, all the items in the subschema record are validated. This validation is called the opening of a subschema record. If the usage and picture (byte count) of a subschema item do not agree with the item type and byte countof the corresponding item in the database, return code 17 is placed in the second COMMBLOCK field for the database, and the ordinal position of that item within the subschema record is placed in the eighth COMMBLOCK field. For example, if the description for the subschema item SECURITY-CLEARANCE is PICTURE 9(5) COMP-3 or PICTURE 9V99 COMP-3, the software issues return code 17 when the program references the subschema record.

With packed decimal representation, the number of bytes required to store an even number of significant digits is the same as that required to store the next higher odd number of significant digits. Therefore, storage for the field described as PICTURE 9(3) COMP-3 is the same as that for a database item defined as INTEGER 9(2). In this situation, although return code 00 is issued when the SSR is opened, return code 10 results if an attempt is made to update a two-digit item in the database with a COMP-3 value of three significant digits. Thus, for updates it is good practice to specify a subschema item picture no larger than the picture of the corresponding schema item. For retrievals, if the subschema item has PICTURE 9(2) and the database item is defined as INTEGER 9(3), the database value (up to three significant digits) is moved to the subschema item field and return code 00 results.

In retrievals as well as in updates, a short value, complete with leading zeros, is moved without editing to the receiving field (the subschema record in retrievals, the database in updates). The value appears in the receiving field right-justified with packed decimal zero filling.

SYSTEM 2000 software places binary zeros in a subschema item when retrieving a null. The software also interprets a field of binary zeros as a null for an update or a comparison in a where-clause.

## COMP Usage

When an INTEGER item is retrieved and placed in a COMP field, the value is converted into its binary representation and is moved into the field right-justified. If the value to be retrieved is too large (or negative and too small) to be represented in a fullword, an abnormal termination of the job occurs with a completion code of OC9 and a SYSTEM 2000 Error Code. Truncation occurs when the value can be placed in a fullword, but the receiving field is shorter than the length of the value. During updates, if the converted value is shorter than its corresponding size in the database definition, the value is stored right-justified and filled with packed decimal zeros. If the converted value is longer than the corresponding field in the database, the software issues return code 10.

Null items are represented by the characters NULL if the field is a fullword, and by the characters NL if the field is a halfword.

### DISPLAY Usage

For COBOL DISPLAY usage with an INTEGER item, the picture must be numeric and without edit characters in order to be accepted by the PLEX processor. For retrievals, the value is converted to character representation and is moved into the field right-justified on the left side of the decimal point and truncated or zero-filled (with character 0) as necessary. The right side of the decimal point is zero-filled. For updates, the value is converted into packed decimal form and is moved right-justified and zero-filled (with packed decimal zeros) or truncated as necessary. Because of truncation on the left, the most significant digits might be lost during retrievals and updates. Binary zeros in the subschema record represent null items in retrievals as well as in updates and where-clauses.

## ITEM TYPES DECIMAL AND MONEY

Decimal numbers can be item type DECIMAL or MONEY. Both DECIMAL and MONEY type items are stored in packed decimal representation. MONEY items are similar to DECIMAL items except that they can be displayed with insertion characters, for example, $ and CR, using the Self-Contained Facility (SCF).

The corresponding field in the program can have COMP-3, COMP, or DISPLAY usage. For example, for a database item defined as ACCRUED VACATION (DECIMAL NUMBER 999.99), the following data descriptions are valid for a subschema item:

```
PICTURE 999V99  |COMP-3
                |COMP
                |DISPLAY
```

Other pictures are available, for example,

```
PICTURE 999V DISPLAY

PICTURE 99V99 COMP-3     (See next topic for COMP-3 usage.)
```

Rules governing the data transfer between a subschema record and a database are similar to those for INTEGER type items. The representation of nulls is also the same.

### COMP-3 Usage

When a DECIMAL or MONEY type item has COMP-3 usage in a subschema record, no conversion is necessary.

When the program references a subschema record for the first time, the software validates all items in the subschema record. If the subschema item field does not have the same field length in bytes and the same number of digits to the right of the decimal point as in the schema item picture, the software issues return code 17 when the subschema record is opened. The software places this return code in the second COMMBLOCK field for the database, and it places the ordinal position of that subschema item in the eighth COMMBLOCK field.

With packed decimal representation, the number of bytes required to store an even number of significant digits (on both sides of the decimal point) is the same as that required to store the next higher odd number of significant digits. Consider, for example, the database item ACCRUED VACATION (DECIMAL NUMBER 999.9) and the corresponding subschema item VACATION. If VACATION is described in the subschema record as PIC 9999V9, it requires the same number of bytes as the database item does, so return code 00 results when the SSR is opened. In this situation, although a zero return code is issued when the SSR is opened, return code 17 results from an update if the program attempts to send a value with more significant digits to the left of the decimal point than the database picture specifies.

For retrievals, if the subschema item has PICTURE 9V9 and the database item is defined as DECIMAL 99.9, the database value (of up to two significant digits to the left of the decimal point) is moved to the subschema item field, and return code 00 results.

In retrievals as well as in updates, a short value, complete with leading zeros, is moved without editing to the receiving field (the subschema record in retrievals, the database in updates). The value appears in the receiving field with packed decimal zero filling. Nulls are represented as binary zeros.

## COMP Usage

A DECIMAL or MONEY type item without a decimal point, or with the assumed decimal point at the right end of its picture, can have COMP usage in a subschema record. For retrievals, the value is converted to its fixed binary representation and is moved into the field right-justified; the higher order bits are truncated if necessary. If the value to be retrieved is too large (or negative and too small) to be represented in a fullword, the job is terminated abnormally with a completion code of OC9 and a SYSTEM 2000 Error Code. Truncation occurs when the value can be represented in 32 bits, but is longer than the receiving field.

For updates, the value is converted to packed decimal. It is stored right-justified with zero-fill on the left side of the decimal point if necessary. The right side of the decimal point is filled with zeros. When the converted value is longer than the number of digits on the left of the decimal point in the receiving field, the software issues return code 17 to avoid truncating the most significant digits.

Null items are represented by the characters NULL in a fullword and by the characters NL in a halfword.

## DISPLAY Usage

The rules for DECIMAL and MONEY items with DISPLAY usage in a subschema record are similar to those for INTEGER items with DISPLAY usage in a subschema record. For retrievals, the value is converted to character representation; for updates, the value is converted to packed decimal. For both updates and retrievals, the value after conversion is moved into the receiving field with the decimal points aligned. Truncation or zero-filling might occur on either side of the decimal point. Loss of the most significant bits might occur when a long value is retrieved into a short field. Binary zeros represent nulls.

# ITEM TYPES REAL AND DOUBLE

REAL and DOUBLE items are acceptable in all PLEX syntax, including the DYNAMIC where-clause.

For REAL and DOUBLE items, internal normalized floating point numbers must be passed by the PLEX program to SYSTEM 2000 software. The software performs no conversions. REAL and DOUBLE values returned to the PLEX program are also internal normalized floating point numbers. External floating point numbers are not allowed in a subschema. NULL in the first word signifies a null value.

You can use COBOL COMP-1 or COMP-2 for REAL and DOUBLE items. The values must be properly aligned in the subschema record definition. SYSTEM 2000 software does not require word alignment when moving data to or from a PLEX subschema record, but subsequent program statements that reference REAL and DOUBLE subschema items do require proper alignment. Therefore, the PLEX processor issues message 361 if you have not aligned the subschema items correctly.

Two SYSTEM 2000 execution parameters, OPT039 and OPT040, allow you to specify several alternatives that check for padding and truncation of floating point numbers. When you transfer single and double precision numbers to and from the database, you can request complete data integrity or you can check return codes 98 and 99 to see whether padding or truncation occurred. For details about OPT039 and OPT040, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

# ITEM TYPE UNDEFINED

All 256 EBCDIC characters are allowed for an UNDEFINED item. The format for UNDEFINED input and output is character format, which means each byte represents one byte of data. Data conversions from hexadecimal to display are not performed. Also, UNDEFINED values cannot overflow; if a value exceeds its picture, return code 10 is issued.

You can use DISPLAY PIC Xs for UNDEFINED type subschema items. Nulls are represented by all binary zeros.

# ITEM TYPES AND COBOL DATA DESCRIPTIONS OF ITEMS IN WHERE-CLAUSES

Among other things, the where-clause provides two kinds of comparison of data values: one data value in the database to another and a value in the database to a value in a program variable, which can belong to a subschema record.

In the first situation, both database values should be from the same group of compatible item types. Two groups exist: CHAR, TEXT, and UNDEFINED items belong to one group, and INTEGER, DECIMAL, MONEY, REAL, DOUBLE, and DATE type items belong to the other. Before the software compares the values, it reformats the value of the right operand according to the item type and picture of the left operand. Thus, if the left operand is an INTEGER value and the right operand is a DECIMAL value, the DECIMAL or MONEY value is truncated before the comparison. Therefore, place the INTEGER value as the right operand to avoid truncation of the DECIMAL value. When a DATE value is the right operand and the

left operand is an INTEGER value, the DATE value is converted to an INTEGER value, which is the difference in number of days between the date and October 15, 1582. If the left operand is a DATE value and the right operand is an INTEGER value *n*, the latter is treated as the date that is *n* days after October 15, 1582. CHAR, TEXT, and UNDEFINED type items are compared as they appear. That is, extraneous blanks are not suppressed from TEXT and UNDEFINED values for comparison with CHAR values.

In the second kind of comparison, the left operand is always a subschema item. With this kind of comparison, the right operand of a binary operator or one or both right operands of a ternary operator can be

- WORKING-STORAGE variables not in subschema records
- subschema items.

Each right operand of a ternary operator can be a variable either within or outside a subschema record.

## WORKING-STORAGE Variables as Right Operands

When a WORKING-STORAGE variable not in a subschema record serves as a right operand, the value for the left operand is taken from the database. SYSTEM 2000 software uses the data description of the left operand to interpret the value of the right operand.

That is, the usage and length of the left operand, as specified for a subschema item, are understood by the software as the usage and length of the program variable(s) in the right operand. If the value of the right operand is incompatible with that of the left operand, return code 17 results. If the subschema item has COMP-3 usage, but the right operand does not contain a packed-decimal value, the job terminates abnormally with a SYSTEM 2000 Error Code and a completion code of OC7.

For character data, the left and right operands should be of equal length. If, however, the specified length of a character variable in the right operand or the length of the value within the variable field is shorter than the item length in the left operand, the right operand value must be followed by a byte of zero bits; otherwise, results of the comparison of operands are unpredictable.

## Subschema Items as Right Operands

When a subschema item is the right operand, the value for the left operand is taken from the database; SYSTEM 2000 software uses the data description and value of the right operand as they appear in a subschema record. In setting up the values of the left and right operands for comparison, the software does not adjust the right operand's value to conform to the data description of the left operand. If the right operand contains character data and its value has fewer characters than its picture allows, the right value must be followed by a byte of zero bits; otherwise, results of the comparison of operands are unpredictable.

# CONSIDERATIONS

1. The use of TEXT values requires special care. SYSTEM 2000 software fills TEXT fields with binary zeros on the right since blanks are significant characters in TEXT type items. Therefore, when a TEXT value in a subschema item is shorter than its field length and you use it in a where-clause or for an update, place a byte of binary zeros at the end of the value. Otherwise, the where-clause comparison will fail, or the updated value will have extra trailing blanks in the database.

2. Data values defined as INTEGER, DECIMAL, MONEY, or DATE are stored in the database as packed decimal. INTEGER, DECIMAL, and MONEY type items can be described in COBOL PLEX with three different usages, as mentioned earlier. It is significantly more efficient, where possible, to let the COBOL compiler generate the necessary code for conversion through an assignment statement than to leave the conversion up to SYSTEM 2000 software. The COBOL compiler can examine the pictures of the sending and receiving fields at compile time and generate appropriate machine instructions, whereas the software must interpret both fields at execution time (whenever the component is referenced) and do some relatively complex work to perform the conversion. A conversion by SYSTEM 2000 software is not necessary if subschema items for INTEGER, MONEY, and DECIMAL type schema items are described as COMP-3 in the COBOL PLEX program.

3. When an INTEGER item has COMP usage in a subschema record, the characters NULL (or NL) are placed in the field when a null item is retrieved. Similarly, for updates and where-clauses, if the characters NULL (or NL) appear in a field, SYSTEM 2000 software interprets that as a null item. When the field is declared as COMP, the binary representation of NULL and NL are the same as that of the numbers -706,423,853 and -10,797, respectively. Therefore, if those numbers are stored in the database, declare them as COMP-3 or DISPLAY, so they will not be misinterpreted by the program or the software.

4. Two return codes are associated with item types. When a program refers to a subschema record or any of its items for the first time, SYSTEM 2000 software validates all items in that subschema record for acceptable pictures and usages. Return code 17 is issued if the picture or usage of a subschema item does not conform to the item type, picture, and size of the corresponding schema item. The software opens the subschema record if all its items have valid pictures and usages. If you place an unacceptable data value in a correctly described field and the data value cannot be moved into the requested field during an operation, SYSTEM 2000 software issues return code 10. (However, if a value in a COMP-3 subschema item field is not in packed-decimal format, an INSERT or MODIFY command referring to that subschema item results in completion code OC7.) When the software issues return codes 10 and 17, it places the ordinal position of the erroneous subschema item in the eighth field of the appropriate COMMBLOCK. (See the discussion of using the ordinal position for rejected conditions in a where-clause, under **Considerations** on page 7-8.)

# SUMMARY CHART

Illus. 4.1 summarizes the correspondence between SYSTEM 2000 item types and COBOL data descriptions. This table shows you how to describe a field corresponding to a schema item in a program and how to place a value in the field so it can be interpreted correctly by the software. It also shows how a value retrieved from a SYSTEM 2000 database is placed in the corresponding field in the program. Representations of null items of all types are given.

**Illus. 4.1** Correspondence between SYSTEM 2000 Item Types and COBOL Data Description

| Item Type | COBOL USAGE and PIC | Requirements/ Restrictions | From Database | To Database |
|---|---|---|---|---|
| TEXT | DISPLAY PIC Xs | If the value is shorter than its subschema item field, a byte of binary zeros must follow the end of the value in the SSR to avoid storage of trailing blanks for updates and to ensure successful comparisons in a where-clause.<br><br>There is no way of knowing the exact number of trailing blanks with a retrieval. | Stored left-justified, blank-filled or truncated on right as necessary.<br><br>Null = binary zeros. | Placed in the database left-justified, with binary zero-fill, or carried over to the Extended Field Table as necessary.<br><br>Place the value left-justified with a byte of binary zeros at the end, if necessary.<br><br>Null = binary zeros in the first byte. |
| CHARACTER | DISPLAY PIC Xs | | Same as TEXT. | Placement same as TEXT.<br><br>Null = binary zeros in the first byte or all blanks. |
| DATE | DISPLAY PIC Xs or 9s | Must be 6 or 8 bytes long. | Placed according to chosen format.<br><br>Null = binary zeros. | Value should conform to chosen data format.<br><br>Null = binary zeros. |
| INTEGER | COMP-3 PIC 9s | Total length in bytes must equal that of the schema item. The number of digits to the right of the decimal point must be zero. | Stored right-justified and zero-filled. Truncation is not permitted.<br><br>Null = binary zeros. | Placed in the database right-justified and zero-filled. Truncation is not permitted.<br><br>Null = binary zeros. |
| INTEGER | COMP PIC 9s | Abend if the value cannot be represented in 32 bits. | Binary representation is placed. Truncation might occur on the left.<br><br>Null = NULL or NL. | Same as above.<br><br>Null = NULL or NL. |
| INTEGER | DISPLAY PIC 9s | Only numeric pictures without edit characters are permitted. | Character representation is placed with aligned decimal point and zero-fill. High-order digits might be truncated if necessary.<br><br>Null = binary zeros. | Placed in the database right-justified, zero-filled. The least significant digits might be truncated if necessary.<br><br>Null = binary zeros. |

*continued on next page*

**Illus. 4.1** *continued*

| Item Type | COBOL USAGE and PIC | Requirements/ Restrictions | From Database | To Database |
|---|---|---|---|---|
| DECIMAL or MONEY | COMP-3 PIC 9s | Must match schema item picture size and the number of digits to the right of the decimal point. | Stored decimal point aligned. Truncation is not permitted.<br><br>Null = binary zeros. | Placed in the database decimal-point-aligned and zero-filled. Truncation of the most significant digits is not permitted.<br><br>Null = binary zeros. |
| DECIMAL or MONEY | COMP PIC 9s | Decimal point, if present, should be at the right end in the schema item picture.<br><br>Abend if binary representation exceeds 32 bits. | Binary representation is placed right-justified. Truncation of the most significant bits might occur.<br><br>Null = NULL or NL. | Same as above.<br><br>Null = NULL or NL. |
| DECIMAL or MONEY | DISPLAY PIC 9s | Only numeric pictures without edit characters are permitted. | Character representation is placed in subschema decimal-point-aligned and zero-filled. Truncation might occur on either side as necessary.<br><br>Null = binary zeros. | Placed in the database decimal-point-aligned, zero-filled if necessary. Truncation might occur on either side.<br><br>Null = binary zeros. |
| REAL | COMP-1 or COMP-2 | Requires fullword alignment or doubleword alignment. The OPT039 and OPT040 execution parameters determine how the system handles padding and truncation. (See the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*)<br><br>Zero means a positive zero. Negative zero is not allowed; causes return code 10. | Internal normalized floating point number is placed in the subschema. External floating point numbers are not allowed in the subschema.<br><br>If you use COMP-2, read about OPT039 execution parameter in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*<br><br>Null = NULL. | Must be internal normalized floating point number. The software performs no conversions. External floating point numbers not allowed.<br><br>If you use COMP-2, read about OPT040 execution parameter in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*<br><br>Null = NULL. |
| DOUBLE | COMP-2 or COMP-1 | Requires doubleword alignment or fullword alignment. The OPT039 and OPT040 execution parameters determine how the system handles padding and truncation. (See the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*) Other rules are the same as REAL. | Internal normalized floating point number is placed in the subschema. External floating point numbers are not allowed in the subschema. If you use COMP-1, read about OPT040 execution parameter in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*<br><br>Null = NULL in the first word. | Must be internal normalized floating point number. The software performs no conversions. External floating point numbers are not allowed. If you use COMP-1, read about OPT039 execution parameter in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*<br><br>Null = NULL in the first word. |

*continued on next page*

**Illus. 4.1** *continued*

| Item Type | COBOL USAGE and PIC | Requirements/ Restrictions | From Database | To Database |
|---|---|---|---|---|
| UNDEFINED | DISPLAY PIC Xs | Each byte represents one byte of data. Data conversions from hexadecimal to display are not performed. If an UNDEFINED value overflows its picture, return code 10 is issued. | Value placed left-justified in subschema with trailing zero-fill.<br><br>Null = all binary zeros. | All 256 EBCDIC characters are allowed. Placed in database left-justified with trailing zero-fill.<br><br>Null = all binary zeros. |

Chapter 5

# COMMBLOCK and SUBSCHEMA RECORD Descriptions

*COMMBLOCK FOR COBOL PROGRAMS 5-1*
*Format 5-2*
*COMMBLOCK Fields 5-3*
*Examples 5-9*
*Considerations 5-19*

*SUBSCHEMA RECORDS FOR COBOL PROGRAMS 5-22*
*Format 5-23*
*Valid Formats of Picture Specifications 5-25*
*Examples 5-25*
*Guidelines for SUBSCHEMA RECORD Descriptions 5-31*
*Considerations for Subschema Items 5-32*

This chapter discusses COMMBLOCK and SUBSCHEMA RECORD descriptions in a COBOL PLEX program. The first section covers the COMMBLOCK, how you describe it and how you use it as a communication area between the program and the database. The second section shows how I/O areas are set up by SUBSCHEMA RECORD descriptions. The sections include examples of COMMBLOCK and SUBSCHEMA RECORD descriptions before and after they have been processed by the PLEX processor.

## COMMBLOCK FOR COBOL PROGRAMS

The COMMBLOCK description sets up an area for the database in the DATA DIVISION, and keeps the status flag, return code, and other identification information. The material in this discussion pertains to COBOL COMMBLOCK descriptions. For FORTRAN, PL/I, and Assembler COMMBLOCK declarations, see the appropriate appendix.

- You can give a COMMBLOCK a name that is different from the database name.
- You can pass a COMMBLOCK to a called program.

A separate COMMBLOCK description is required for each database to be accessed by the program. If a program (independent, calling, or called) is to process a database independently of any other program in the job step, the COMMBLOCK must appear in WORKING-STORAGE. If subprograms are being used, a COMMBLOCK description can appear in either the WORKING-STORAGE SECTION or the LINKAGE SECTION of the DATA DIVISION, as necessary for a program to send or receive the COMMBLOCK as a parameter.

A COMMBLOCK description consists of two main parts. The first part, a one-line statement, gives the name of the database. The second part resembles a COBOL record description, giving the name of the COMMBLOCK and its fields. The COMMBLOCK name can be either

the database name or an alternate name (see Example 2). The ability to use an alternate name is useful when the database name is not a valid COBOL data name. For the COMMBLOCK fields, the program specifies only the names; the PLEX processor generates appropriate picture clauses. You can specify fourteen fields, along with an optional fifteenth field.

Each COMMBLOCK field to be used in SYSTEM 2000 processing must have the same level number within any one COMMBLOCK. But a COMMBLOCK can also include one or more REDEFINES and RENAMES clauses to allow for additional subdivision or grouping of COMMBLOCK fields as the PLEX program might require for its own processing. See Example 5 later in this chapter, as well as #4 in the Considerations that follow Example 5.

The PLEX processor translates the COMMBLOCK description into a COBOL record description. It generates the name of the COBOL record by adding the suffix -0 to the COMMBLOCK name. The PLEX processor also generates a 28-byte control block with FILLER statements at the beginning of the COBOL record description. If the COMMBLOCK is part of WORKING-STORAGE, the PLEX processor specifies initial values for the control block to provide internal identification information used in executing PLEX commands. If, on the other hand, the COMMBLOCK appears in the LINKAGE SECTION, no initial values are generated for the control block. (See Example 4.)

## Format

```
COMMBLOCK OF database-name.
01    name .
      n  var-1 .
      n  var-2 .
      n  var-3  VALUE IS 'initial-value'.
      n  var-4 .
      n  var-5  VALUE IS 'initial-value'.
      n  var-6 .
       .
       .
       .
      n  var-14 .
     [n  var-15 .]
```

*database-name* is the name of a SYSTEM 2000 database.

*name* is any valid COBOL data name, up to 28 characters.

*n* is any valid COBOL level number greater than 01.

*var-n* is any valid COBOL data name, up to 30 characters, naming a field of the COMMBLOCK. Each COMMBLOCK field is discussed in the next section.

*initial-value* for *var-3*, is a five-character literal; for *var-5*, is a four-character literal that is the password for the database.

## COMMBLOCK Fields

*var-1* (subschema record name)

The PLEX processor describes this field as PIC X(30) and initializes it to blanks. If a PLEX command refers to an SSR, SYSTEM 2000 software places the name of the SSR in this field. If the command does not reference an SSR, SYSTEM 2000 software puts blanks in this field. When a command is rejected because of an error in a subschema item, SYSTEM 2000 software stores the name or alias of that subschema item in this field. Thus, this field can be used as a debugging aid.

*var-2* (return code)

This field is specified as PIC 9(4) and is initialized to zero by the PLEX processor. This is a very important field returned to the PLEX program, since it contains the return code from SYSTEM 2000 software after the execution of each command referring to that COMMBLOCK. This field must be checked after each command. *SYSTEM 2000 Messages and Codes, Version 12, First Edition* discusses the meaning of each return code. Every PLEX program should contain code that checks the value in this field and takes any action necessary. Failure to check the return code immediately might multiply errors and cause problems that would be difficult to solve.

**Note**: The variable S2KRTC in the S2KDUM control block contains the return code after every command, regardless of whether the command refers to a database. Since some commands, such as START S2K, do not refer to a database, their return codes are placed only in S2KRTC, not in any COMMBLOCK.

*var-3* (for the user)

The PLEX processor describes this field as PIC X(5) and initializes it to blanks. This field is not used by SYSTEM 2000 software. You can store information about the database in this field, and you can initialize it using a value clause.

*var-4* (last data record indicator)

The PLEX processor describes this field as PIC 9 and initializes it to zero. SYSTEM 2000 software sets this field to 1 when the data record for the last address in a Locate File is retrieved by a GET command. Otherwise, this field is set to 0. You can use this field after a LOCATE command to control the retrieval of data records from a located subset, as shown in the following example.

Suppose LAST-REC is the name of the fourth COMMBLOCK field.

```
LOCATE SSR WHERE . . .
(Check return code)
       .
       .
       .
PERFORM GETLOOP UNTIL LAST-REC = 1.
       .
       .
       .
GETLOOP.
GET SSR NEXT.
(Check return code)
       .
       .
       .
```

For the GETP command only, this field has a different function. SYSTEM 2000 software sets this field to 1 if the database contains a specified subschema item value that equals the value in the where-clause condition. Otherwise, for GETP, this field is set to 0.

*var-5* (password)

The PLEX processor describes this field as PIC X(4). This field must contain a valid password before the program opens the database associated with the COMMBLOCK. That is, the password (master, secondary, or DBA) must be present in the COMMBLOCK before the program issues an OPEN command for the database and before it refers again to the database after working with another database. Until a CLOSE command is issued for the database, SYSTEM 2000 software automatically keeps track of the access authorities allowed for the secondary passwords. It also keeps track of the commands available to the DBA password. The password in the COMMBLOCK is not referred to again until the database is opened again (either explicitly by an OPEN command or implicitly by a command referencing the database after the program has used another database).

The password can be initialized using a value clause, or it can be moved or read into this field before the database is opened.

*var-6* (number of records selected by LOCATE or qualified by GET1)

The PLEX processor describes this field as PIC 9(10) and initializes it to 0. SYSTEM 2000 software sets this field after each successful LOCATE command or GET1 command with a where-clause for the database. The value of this field indicates the number of data records selected as a result of a LOCATE command or the number of qualified records resulting from a GET1 command with a where-clause. You can use this field after a LOCATE command to control the retrieval of data records from a selected subset of the database.

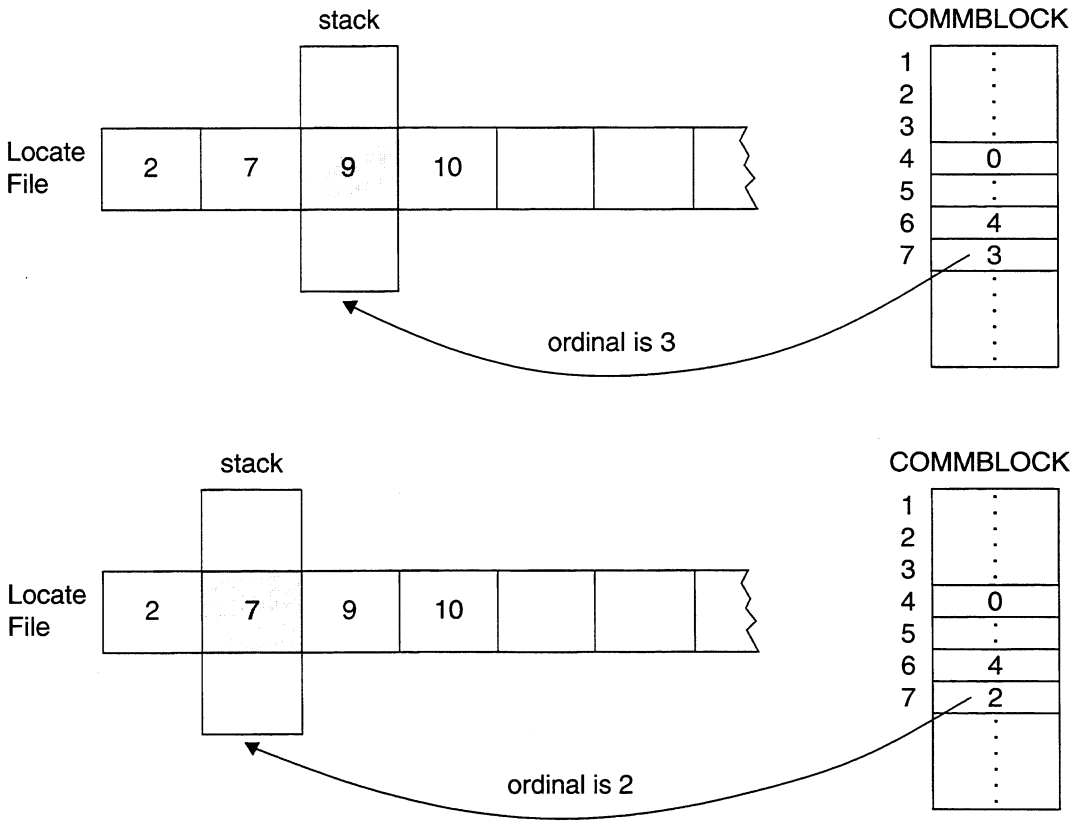
Suppose the name of the sixth COMMBLOCK field is NUMREC.

```
      .
      .
      .
LOCATE SSR WHERE . . .
(Check return code)
      .
      .
      .
PERFORM GETLOOP
   VARYING LOOP-COUNT FROM 1 BY 1
   UNTIL LOOP-COUNT > NUMREC.
         .
         .
         .
GETLOOP.
   GET SSR NEXT.
   (Check return code)
         .
         .
         .
```

For the GETP command only, this field has a different function. SYSTEM 2000 software sets the field to one if the where-clause value is unique in the database for the specified subschema item. Otherwise, for GETP, the field is set to zero.

*var-7* (Locate File pointer: position of retrieved data record in Locate File)

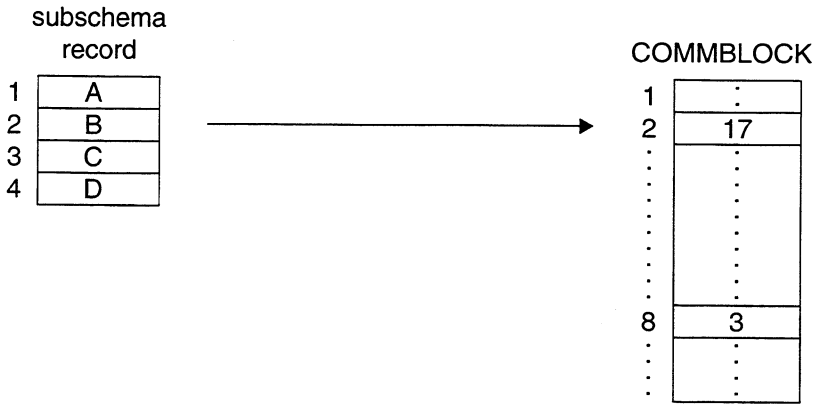
The PLEX processor describes this field as PIC 9(10) and initializes it to zero. This field contains the ordinal of the data record address in the Locate File when a data record is retrieved with a GET command (which sets the primary stack entry), as shown here. This field can be used for debugging when an error return code occurs.



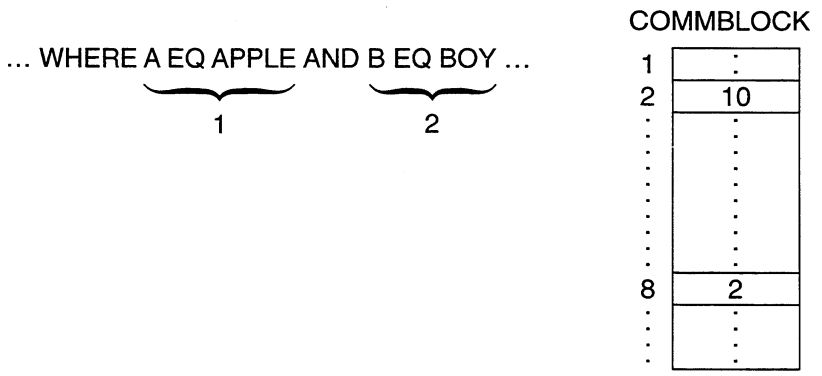

*var-8* (level)

The PLEX processor describes this field as PIC 99 and initializes it to zero. During the normal execution of a command, SYSTEM 2000 software uses this field to store the hierarchical level of the data record accessed. However, under error conditions this field contains information helpful for debugging, such as the ordinal of a rejected item in its subschema record, the ordinal of a component in a where-clause responsible for the rejection of the command, or the level of a data record removed from the database. The examples on the next page show cases in which the eighth COMMBLOCK field contains the ordinals of rejected items.

Case 1: The subschema record is referenced, but item C is an unacceptable type, so the subschema record cannot be opened. Since C is the third item in the subschema record, the value 3 is placed in the eighth COMMBLOCK field. Return code 17 is also placed in the second COMMBLOCK field.




Case 2: The where-clause is executed and the variable BOY contains an unacceptable value, so the second condition cannot be processed. The value 2 is placed in the eighth COMMBLOCK field. Also, return code 10 is placed in the second COMMBLOCK field.



*var-9* (cycle time)

The PLEX processor describes this field as PIC X(6) and initializes it to blanks. SYSTEM 2000 software places here the time of day of the last update to the database. The time format is HHMMSS (hours, minutes, seconds). The time is entered in this field whenever the database is referenced.

*var-10* (cycle date)

The PLEX processor describes this field as PIC X(6) and initializes it to blanks. SYSTEM 2000 software places here the date of the last update to the database. The format is YYMMDD (year, month, day). The date is entered in this field whenever the database is referenced.

*var-11* (cycle number)

The PLEX processor describes this field as PIC 9(8) and initializes it to zero. SYSTEM 2000 software places the cycle number of the database in this field whenever the database is referenced. The software maintains a cycle number for each database internally in the database files. The cycle number indicates how many times the data in the database was changed. The internal cycle number is moved into this COMMBLOCK field so that you can associate the cycle number with the updates to the database. The maximum value for the cycle number is 2,147,483,647.

*var-12* (system separator)

The PLEX processor describes this field as PIC X and initializes it to blank. When the database is opened, SYSTEM 2000 software places the system separator in this field. The default system separator is an asterisk. **Note**: the program must use an asterisk as the system separator with ternary operators in a where-clause unless it specifies a different character with the SYSSEP execution option (see **SYSSEP Execution Option** on page 19-20).

*var-13* (entry terminator)

The PLEX processor describes this field as PIC X(4) and initializes it to blanks. When the database is opened, SYSTEM 2000 software places the entry terminator word in this field. The default entry terminator word is END, left-justified and blank-filled. The entry terminator word is not used in PLEX.

*var-14* (database damage flag)

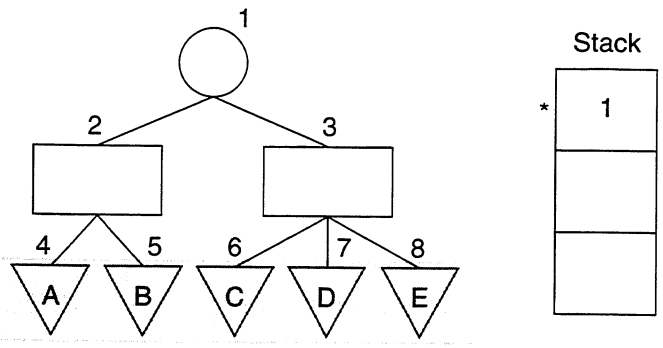
The PLEX processor describes this field as PIC 9 and initializes it to zero. When the database is opened, SYSTEM 2000 software moves the internal damage flag for the database into this field. A zero value indicates an undamaged database, and a value of one indicates that the database is damaged. This field should be checked after the OPEN command is given. Updates are not permitted for a damaged database, although data retrievals are allowed.

*var-15* (parent; optional field)

The PLEX processor describes this optional field as PIC S9(9) USAGE COMP and initializes it to zero. When any retrieval command is executed, SYSTEM 2000 software uses this field to store the value of the Hierarchical Table pointer for the parent of the data record retrieved. (See **Physical and Logical Order of Data Records** on page 16 for a description of the Hierarchical Table.)

You can use this field to distinguish between similar data records under different parents. The value of this field, being an internal address, is not meaningful to the user. But you can save it after a retrieval and compare it with the value of the parent in the next retrieval.

In the following illustration, the domain of GETD ▽ with the tree and stack as shown consists of all the ▽ 's . If you want to retrieve A and B only (not C, D, or E) with GETD commands without retrieving their parent, you can check the parent field of the COMMBLOCK after each GETD ▽ and stop when the parent field changes value.



## Examples

Five examples of COMMBLOCKs are presented here. Each example shows how the COMMBLOCK actually appears in the source code, followed by the expansion produced by the PLEX processor as input to the COBOL compiler.

### Example 1

A COMMBLOCK for the EMPLOYEE database is shown here. Note that no picture clause or initial value is specified for any field.

```
COMMBLOCK OF EMPLOYEE.
01  EMPLOYEE.
    05  RECORD-NAME.
    05  RETURN-KODE
    05  FREE.
    05  LAST-RECORD-FLAG.
    05  PASSWD.
    05  NO-OF-RECORDS.
    05  LF-POINTER.
    05  LEVEL.
    05  CTIME.
    05  CDATE.
    05  CYCLE.
    05  SEPARATOR.
    05  TERMINATOR.
    05  DAMAGE.
```

**Example 1 (continued)**

Here is the same COMMBLOCK description, augmented by the PLEX processor for input to the COBOL compiler.

Note that the PLEX processor made a comment out of the first two lines of the program's COMMBLOCK description. Also, the processor inserted the control block (level 02) and added picture clauses and initial values to the descriptions of the COMMBLOCK fields.

```
*COMMBLOCK OF EMPLOYEE.
*01  EMPLOYEE.
 01  EMPLOYEE-0.
     02  FILLER PIC X(16) VALUE 'EMPLOYEE'.
     02  FILLER PIC S9(9) USAGE COMP VALUE +0.
     02  FILLER PIC S9(9) USAGE COMP VALUE +120.
     02  FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 EMPLOYEE.
     05 RECORD-NAME              PIC X(30) VALUE ' '.
     05 RETURN-KODE              PIC 9(4) VALUE 0.
     05 FREE                     PIC X(5) VALUE ' '.
     05 LAST-RECORD-FLAG         PIC 9 VALUE 0.
     05 PASSWD                   PIC X(4) VALUE ' '.
     05 NO-OF-RECORDS            PIC 9(10) VALUE 0.
     05 LF-POINTER               PIC 9(10) VALUE 0.
     05 LEVEL                    PIC 99 VALUE 0.
     05 CTIME                    PIC X(6) VALUE ' '.
     05 CDATE                    PIC X(6) VALUE ' '.
     05 CYCLE                    PIC 9(8) VALUE 0.
     05 SEPARATOR                PIC X VALUE ' '.
     05 TERMINATOR               PIC X(4) VALUE ' '.
     05 DAMAGE                   PIC 9 VALUE 0.
```

### Example 2

A COMMBLOCK for a database called INVENTORY can be described as shown in this example.

This COMMBLOCK includes the optional fifteenth field (INV-PARENT-RECORD) as an indicator of the parent of a record retrieved by the program. The program initializes both the third and the fifth fields. No picture clause is specified for any field. Note that the declared name of the COMMBLOCK (INVENTCOMM) can be different from the database name (INVENTORY).

```
COMMBLOCK OF INVENTORY.
01  INVENTCOMM.
    05  INV-RECORD-NAME.
    05  INV-RETURN-KODE.
    05  INV-FREE           VALUE 'CHEK'.
    05  INV-LAST-RECORD-FLAG.
    05  INV-PASSWORD       VALUE 'ABCD'.
    05  INV-NO-OF-RECORDS.
    05  INV-LF-POINTER.
    05  INV-LEVEL.
    05  INV-CTIME.
    05  INV-CDATE.
    05  INV-CYCLE.
    05  INV-SEPARATOR.
    05  INV-TERMINATOR.
    05  INV-DAMAGE.
    05  INV-PARENT-RECORD.
```

**Example 2 (continued)**

In this expansion of the description by the PLEX processor, note that the item descriptions now include picture clauses as well as the initial values chosen by the program. In addition, the fifteenth field, INV-PARENT-RECORD, is specified USAGE COMP.

```
*COMMBLOCK OF INVENTORY.
*01  INVENTCOMM.
 01  INVENTCOMM-0.
     02  FILLER PIC X(16) VALUE 'INVENTORY'.
     02  FILLER PIC S9(9) USAGE COMP VALUE +0.
     02  FILLER PIC S9(9) USAGE COMP VALUE +124.
     02  FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 INVENTCOMM.
     05 INV-RECORD-NAME          PIC X(30) VALUE ' '.
     05 INV-RETURN-KODE          PIC 9(4) VALUE 0.
     05 INV-FREE                 PIC X95) VALUE 'CHEK'.
     05 INV-LAST-RECORD-FLAG     PIC 9 VALUE 0.
     05 INV-PASSWORD             PIC X(4) VALUE 'ABCD'.
     05 INV-NO-OF-RECORDS        PIC 9(10) VALUE 0.
     05 INV-LF-POINTER           PIC 9(10) VALUE 0.
     05 INV-LEVEL                PIC 99 VALUE 0.
     05 INV-CTIME                PIC X(6) VALUE ' '.
     05 INV-CDATE                PIC X(6) VALUE ' '.
     05 INV-CYCLE                PIC 9(8) VALUE 0.
     05 INV-SEPARATOR            PIC X VALUE ' '.
     05 INV-TERMINATOR           PIC X(4) VALUE ' '.
     05 INV-DAMAGE               PIC 9 VALUE 0.
     05 INV-PARENT-RECORD        PIC S9(9) USAGE COMP VALUE +0.
```

### Example 3

A sample COMMBLOCK description for the COMPANY database is shown here. This COMMBLOCK includes the optional fifteenth field for the parent of a retrieved record. Note that the program uses level-number 02 in the item descriptions for the COMMBLOCK fields.

```
COMMBLOCK OF COMPANY.
01  COMPANY.
    02  REC-NAME.
    02  RETURNCODE.
    02  FREE.
    02  LAST-REC-FLAG.
    02  PASSWD.
    02  NUM-OF-RECS.
    02  LOC-FILE-POINTER.
    02  ACCESS-LEVEL.
    02  CYCLE-TIME.
    02  CYCLE-DATE.
    02  CYCLE-NUMBER.
    02  SEPTOR.
    02  TERMIN.
    02  DAMAGE-FLAG.
    02  PARENT-RECORD.
```

In the following output from the PLEX processor, a warning message indicates that the level numbers in the item descriptions were increased to 03, so the higher-level control block can be inserted in the correct place in the COMMBLOCK.

```
   *COMMBLOCK OF COMPANY.
   *01  COMPANY.
 .........C
****     PLX202 ALL LOWER LEVEL NUMBERS HAVE BEEN INCREASED BY ONE
         03  REC-NAME           PIC X(30) VALUE ' '.
         03  RETURNCODE         PIC 9(4) VALUE 0.
         03  FREE               PIC X(5) VALUE ' '.
         03  LAST-REC-FLAG      PIC 9 VALUE 0.
         03  PASSWD             PIC X(4) VALUE ' '.
         03  NUM-OF-RECS        PIC 9(10) VALUE 0.
         03  LOC-FILE-POINTER   PIC 9(10) VALUE 0.
         03  ACCESS-LEVEL       PIC 99 VALUE 0.
         03  CYCLE-TIME         PIC X(6) VALUE ' '.
         03  CYCLE-DATE         PIC X(6) VALUE ' '.
         03  CYCLE-NUMBER       PIC 9(8) VALUE 0.
         03  SEPTOR             PIC X VALUE ' '.
         03  TERMIN             PIC X(4) VALUE ' '.
         03  DAMAGE-FLAG        PIC 9 VALUE 0.
         03  PARENT-RECORD      PIC S9(9) USAGE COMP VALUE +0.
```

**Example 3 (continued)**

The following PLEX processor output, expanded for input to the COBOL compiler, shows that the control block was inserted (level 02):

```
*COMMBLOCK OF COMPANY.
*01  COMPANY.
 01  COMPANY-0.
     02  FILLER PIC X(16) VALUE 'COMPANY'.
     02  FILLER PIC S9(9) USAGE COMP VALUE +0.
     02  FILLER PIC S9(9) USAGE COMP VALUE +124.
     02  FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 COMPANY.
     03 REC-NAME           PIC X(30) VALUE ' '.
     03 RETURNCODE         PIC 9(4) VALUE 0.
     03 FREE               PIC X(5) VALUE ' '.
     03 LAST-REC-FLAG      PIC 9 VALUE 0.
     03 PASSWD             PIC X(4) VALUE ' '.
     03 NUM-OF-RECS        PIC 9(10) VALUE 0.
     03 LOC-FILE-POINTER   PIC 9(10) VALUE 0.
     03 ACCESS-LEVEL       PIC 99 VALUE 0.
     03 CYCLE-TIME         PIC X(6) VALUE ' '.
     03 CYCLE-DATE         PIC X(6) VALUE ' '.
     03 CYCLE-NUMBER       PIC 9(8) VALUE 0.
     03 SEPTOR             PIC X VALUE ' '.
     03 TERMIN             PIC X(4) VALUE ' '.
     03 DAMAGE-FLAG        PIC 9 VALUE 0.
     03 PARENT-RECORD      PIC S9(9) USAGE COMP VALUE +0.
```

### Example 4

If a COMMBLOCK description appears in the LINKAGE SECTION of a program, the PLEX processor generates picture clauses and inserts the control block, but it does not generate initial values. Here is the COMMBLOCK as it appears in the source code:

```
LINKAGE SECTION.
COMMBLOCK OF DEPARTMENTS.
01  DEPARTMENTS.
    05  RECORD-NAME.
    05  RETURN-KODE.
    05  FREE.
    05  LAST-RECORD-FLAG.
    05  PASSWD.
    05  NO-OF-RECS.
    05  LF-POINTER.
    05  LEVEL.
    05  CTIME.
    05  CDATE.
    05  CYCLE.
    05  SEPARATOR.
    05  TERMINATOR.
    05  DAMAGE-FLAG.
```

**Example 4 (continued)**

As expanded by the PLEX processor, the COMMBLOCK description includes no initial values.

```
LINKAGE SECTION.
*COMMBLOCK OF DEPARTMENTS.
*01  DEPARTMENTS.
 01  DEPARTMENTS-0.
     02  FILLER PIC X(16).
     02  FILLER PIC S9(9) USAGE COMP.
     02  FILLER PIC S9(9) USAGE COMP.
     02  FILLER PIC S9(9) USAGE COMP.
     02 DEPARTMENTS.
     05 RECORD-NAME.
     05 RETURN-KODE.
     05 FREE.
     05 LAST-RECORD-FLAG.
     05 PASSWD.
     05 NO-OF-RECS.
     05 LF-POINTER.
     05 LEVEL.
     05 CTIME.
     05 CDATE.
     05 CYCLE.
     05 SEPARATOR.
     05 TERMINATOR.
     05 DAMAGE-FLAG.
```

### Example 5

REDEFINES and RENAMES clauses appear in this COMMBLOCK description.

REDEFINES is used twice. It provides an alternate description of RETURN-KODE, and it subdivides the CTIME field. A level-66 item at the end of the COMMBLOCK groups together (RENAMES) the three COMMBLOCK fields related to the database cycle.

```
COMMBLOCK OF PROGRAMS.
01  PROGRAMS.
    05  RECORD-NAME.
    05  RETURN-KODE.
    05  RETURN-KODE-X REDEFINES RETURN-KODE PIC XXXX.
    05  FREE.
    05  LAST-RECORD-FLAG.
    05  PASSWD.
    05  NO-OF-RECS.
    05  LF-POINTER.
    05  LEVEL.
    05  CTIME.
    05  TIME-BREAKDOWN REDEFINES CTIME.
        10  CYCLE-HOUR    PIC XX.
        10  CYCLE-MINUTE  PIC XX.
        10  CYCLE-SECOND  PIC XX.
    05  CDATE.
    05  CYCLE.
    05  SEPARATOR.
    05  TERMINATOR.
    05  DAMAGE-FLAG.
    66  CYCLE-INFO RENAMES CTIME THRU CYCLE.
```

The PLEX processor generates the normal code, skipping over the RENAMES clause and the REDEFINES clauses and subordinate items, as shown on the next page.

**Example 5 (continued)**

```
*COMMBLOCK OF PROGRAMS.
*01  PROGRAMS.
 01  PROGRAMS-0.
     02  FILLER PIC X(16) VALUE 'PROGRAMS'.
     02  FILLER PIC S9(9) USAGE COMP VALUE +0.
     02  FILLER PIC S9(9) USAGE COMP VALUE +120.
     02  FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 PROGRAMS.
     05 RECORD-NAME          PIC X(30) VALUE ' '.
     05 RETURN-KODE          PIC 9(4) VALUE 0.
     05  RETURN-KODE-X REDEFINES RETURN-KODE PIC XXXX.
     05 FREE                 PIC X(5) VALUE ' '.
     05 LAST-RECORD-FLAG     PIC 9 VALUE 0.
     05 PASSWD               PIC X(4) VALUE ' '.
     05 NO-OF-RECS           PIC 9(10) VALUE 0.
     05 LF-POINTER           PIC 9(10) VALUE 0.
     05 LEVEL                PIC 99 VALUE 0.
     05 CTIME                PIC X(6) VALUE ' '.
     05  TIME-BREAKDOWN REDEFINES CTIME.
         10  CYCLE-HOUR      PIC XX.
         10  CYCLE-MINUTE    PIC XX.
         10  CYCLE-SECOND    PIC XX.
     05 CDATE                PIC X(6) VALUE ' '.
     05 CYCLE                PIC 9(8) VALUE 0.
     05 SEPARATOR            PIC X VALUE ' '.
     05 TERMINATOR           PIC X(4) VALUE ' '.
     05 DAMAGE-FLAG          PIC 9 VALUE 0.
     66  CYCLE-INFO RENAMES CTIME THRU CYCLE.
```

## Considerations

1. The COMMBLOCK contains important debugging information for checking program logic. It should be printed in an understandable format when an unexpected return code is issued.

2. The PLEX processor does not generate initial values for fields in a COMMBLOCK description in the LINKAGE SECTION. The calling program that passes the COMMBLOCK as a parameter must use the suffixed name of the COMMBLOCK (such as EMPLOYEE-0) in its CALL statement, and the called program must include the suffixed name in its PROCEDURE DIVISION header or ENTRY statement.

3. A COMMBLOCK must contain 14 fields, along with an optional fifteenth field (excluding REDEFINES and RENAMES items). If it does not, the PLEX processor gives an error message.

4. Each of the 14 or 15 COMMBLOCK fields to be used in SYSTEM 2000 software processing must have the same level number within any one COMMBLOCK. If this level number is less than 3, the PLEX processor substitutes a level number that is one level greater than the given level number (see Example 3). If the level numbers for all of these COMMBLOCK fields are not identical, the PLEX processor issues an error message.

   But a COMMBLOCK description can include one or more items with REDEFINES or RENAMES clauses. The level number of an item with a REDEFINES clause must be the same as the level number of the COMMBLOCK fields. When the PLEX processor encounters the keyword REDEFINES immediately following a data name, it skips item descriptions until it either finds another item with the level number for COMMBLOCK fields or finds the end of the COMMBLOCK description. The PLEX processor does not check the validity of a REDEFINES clause or of items subordinate to a REDEFINES clause item.

   Group items besides REDEFINES clause items cannot appear within a COMMBLOCK description. However, one or more level-66 items with RENAMES clauses can appear in a COMMBLOCK to specify item grouping that the PLEX program might require for its own processing. If an item with a RENAMES clause is used in a COMMBLOCK, it must appear at the end of the COMMBLOCK description. The PLEX processor does not check the validity of the RENAMES clause.

5. A program can supply initial values for only the third and fifth fields in the COMMBLOCK. If it assigns an initial value to some other COMMBLOCK field, the PLEX processor issues an error message.

6. If the COMMBLOCK name is not unique within the program, the PLEX processor issues an error message.

7. When a Locate File is currently in use by the program, the fourth, sixth, and seventh COMMBLOCK fields are set for the Locate File. When a program uses one Locate File, then switches to a second one, those fields are set with the appropriate information for the second Locate File. If the program resumes retrievals with the first Locate File, the information for that Locate File is brought back into the COMMBLOCK. Thus, for example, a program can give GET commands to retrieve records for the first five addresses in the first Locate File, then work with another Locate File for a while, then resume retrievals at the point where it had left off in the first Locate File.

8. The sixth COMMBLOCK field is set after a program issues a GET1 command with a where-clause. The value of this field is the number of data records qualified by the where-clause.

9. For the GETP command only, the fourth and sixth COMMBLOCK fields function differently than for other retrieval commands. (See **COMMBLOCK Fields** on page 5-3.) When GETP is not the command, the fourth COMMBLOCK field indicates whether the Locate File pointer is at the last record. When GETP is not the command, the sixth COMMBLOCK field indicates the Locate File number of records.

10. Comments can be interspersed among the lines of a COMMBLOCK description.

11. The database name associated with a COMMBLOCK can be changed during program execution. The technique for changing the database name in the COMMBLOCK is useful for processing two or more databases in which the components to be accessed have identical definitions.

    The name of the database associated with the COMMBLOCK is kept in the first 16 bytes of the COMMBLOCK control block. The PLEX processor first associates the COMMBLOCK with a database by initializing this 16-byte FILLER field to the name specified after OF in the COMMBLOCK statement. A COBOL PLEX PROGRAM can change this name at execution time by using an item with a REDEFINES clause. Each time the PLEX program moves a different database name to the redefined field, it must reissue the OPEN command specifying the COMMBLOCK name (the level-01 data name), regardless of whether the database has already been opened.

    The changed database name in the COMMBLOCK, together with the OPEN command for that database, associates each SSR for that COMMBLOCK with the alternate database. Subschema item descriptions must be compatible with component definitions in each database with which an SSR is associated. If they are not, the software issues return code 17 when the SSR is referenced.

    An example of redefining the database name field appears below. Notice the following:

    a) The database name DEPT is initialized in the first level-02 field of the COMMBLOCK description generated by the PLEX processor.

    b) The program can redefine the field, as shown in the last two lines, to make it available for storing an alternate database name.

c) The REDEFINES clause must include the level-01 COMMBLOCK name (with the suffix -0) generated by the PLEX processor.

```
*COMMBLOCK OF DEPT.
*01  DEPT
 01  DEPT-0.
     02  FILLER PIC X(16) VALUE 'DEPT'.
     02  FILLER PIC S9(9) USAGE COMP VALUE +0.
     02  FILLER PIC S9(9) USAGE COMP VALUE +120.
     02  FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 DEPT.
     05 RECORD-NAME              PIC X(30) VALUE ' '.
     05 RETURN-KODE              PIC 9(4) VALUE 0.
     05 FREE                     PIC X(5) VALUE ' '.
     05 LAST-RECORD-FLAG         PIC 9 VALUE 0.
     05 PASSWD                   PIC X(4) VALUE ' '.
     05 NO-OF-RECORDS            PIC 9(10) VALUE 0.
     05 LF-POINTER               PIC 9(10) VALUE 0.
     05 LEVEL                    PIC 99 VALUE 0.
     05 CTIME                    PIC X(6) VALUE ' '.
     05 CDATE                    PIC X(6) VALUE ' '.
     05 CYCLE                    PIC 9(8) VALUE 0.
     05 SEPARATOR                PIC X VALUE ' '.
     05 TERMINATOR               PIC X(4) VALUE ' '.
     05 DAMAGE                   PIC 9 VALUE 0.
*
01  ALTERNATE-COMMBLOCK REDEFINES DEPT-0.
     05  ALTERNATE-DB-NAME       PIC X(16).
```

12. Instead of the technique described in consideration 11, you can use subprograms to process two or more databases in which the components to be accessed have identical definitions. You place S2KDUM, the SUBSCHEMA RECORD descriptions, and a single COMMBLOCK description in the LINKAGE SECTION of a called program. The calling program contains S2KDUM with values and a separate COMMBLOCK in WORKING-STORAGE for each database to be accessed.

    The calling program decides which database is to be accessed and passes S2KDUM and the appropriate COMMBLOCK to the called program, which then issues one or more commands referring to the COMMBLOCK or the SSRs. In this situation, when the called program is invoked, SYSTEM 2000 software automatically takes care of opening the database and opening (validating) the SSRs as necessary.

    Using this technique, the called program does not need to include an OPEN command for the database to be accessed, unless the damage flag needs to be checked before update commands are issued.

# SUBSCHEMA RECORDS FOR COBOL PROGRAMS

A SUBSCHEMA RECORD (SSR) description associates COBOL data names with a schema record and the items in the schema record. The SSR represents a schema record, and the items in the SSR represent the corresponding schema items. The SUBSCHEMA RECORD description also allocates a storage area for items in the corresponding data records. Update and retrieval commands use subschema records as input and output areas, respectively, for data records in a database. A database can be accessed only through a subschema record.

The material in this section pertains to COBOL SUBSCHEMA RECORD descriptions. For FORTRAN, PL/I, and Assembler SUBSCHEMA RECORD declarations, see the appropriate appendix.

A subschema record has the following features:

- You can pass a subschema record to a called program.
- The name of a subschema record containing items can be different from the name of the corresponding schema record.
- The names of subschema items can be different from the names of the corresponding database items, if you use the EQ option.
- Items can appear in any order in a subschema record, regardless of the corresponding order in a schema record.
- You can initialize subschema items in the description.
- Six different display formats are available for items of type DATE.

At least one subschema record (SSR) must be described for each schema record representing data records to be accessed by the program. A SUBSCHEMA RECORD description can appear in either the WORKING-STORAGE SECTION or the LINKAGE SECTION of the DATA DIVISION. If a program (independent, calling, or called) processes an SSR independently of any other program in the job step, the SUBSCHEMA RECORD description must appear in the WORKING-STORAGE SECTION. If you are using subprograms, the SUBSCHEMA RECORD description can appear in either WORKING-STORAGE or the LINKAGE SECTION as necessary for a program to send or receive the SSR as a parameter.

A SUBSCHEMA RECORD description consists of two main parts: the name of the SSR and database and the names of the subschema items. In the first part of the description, you specify the name of the SSR, along with either the COMMBLOCK name or the database name. The second part repeats the SSR name and includes the names, picture clauses, and usage clauses of the subschema items. These items must correspond to items in a single schema record.

The name (alias) of a subschema item can be the component name or C-number (the letter C followed by the component number) of a schema item, or it can be an alternate data name that is acceptable in COBOL. A component name can contain blanks, which are not acceptable in a COBOL data name. Each C-number is acceptable except that C01, C02, through C12 are COBOL reserved words. The use of an alternate data name is especially helpful when the component name is not acceptable in COBOL and a data name is required

that is more meaningful than the C-number. If the alias is not the component name or C-number, the component name or C-number must appear, preceded by the keyword EQ, on a line below the alias in an item description. Retrieval and update processing is most efficient when the C-number of a subschema item appears either as the item alias or on the EQ line of the item description. This applies even when the component name is used as the item alias.

Each subschema item must be declared with a picture clause and usage clause that is compatible with the item types of the corresponding items in the database. See Chapter 4, "SYSTEM 2000 Item Types and COBOL Data Descriptions" for a discussion of item types and COBOL data descriptions. A subschema item for a database item of type DATE can specify any one of six date formats with a FORM statement (see Example 2). This statement must appear on a line below the item description line and can precede or follow an EQ statement if there is one.

Each of the subschema items must have the same level number within any one SUBSCHEMA RECORD description. But a SUBSCHEMA RECORD description can also include one or more REDEFINES and RENAMES clauses to allow for additional subdivision or grouping of subschema items as the PLEX program might require for its own processing. See Example 4 and **Considerations for Subschema Items** on page 5-32.

The PLEX processor converts the SUBSCHEMA RECORD description to a COBOL record description. The record name is generated by adding a suffix of -0 to the name of the SSR. The PLEX processor creates a control block for the SSR and its items to store information, including the component name or C-number, for each item. If the SSR is part of WORKING-STORAGE, the PLEX processor specifies initial values for the control block. However, if the SSR appears in the LINKAGE SECTION, no initial values are generated for the control block.

The actual input/output area for the subschema items follows the control block. The PLEX processor uses the subschema record name, item aliases, and picture clauses specified in the SUBSCHEMA RECORD description to generate this area.

## Format

```
SUBSCHEMA [RECORD] name OF database-identifier
01 name
   [item-descriptions]
```

| | |
|---|---|
| *name* | is any valid COBOL data name, up to 28 characters. |
| *database-identifier* | is the name of a SYSTEM 2000 database or the name of a COMMBLOCK. |
| *item-descriptions* | *item-description*<br>[*item-descriptions*] |

Abbreviation: SUBSCHEMA RECORD = SSR

*item-description* — *alias picture-clause* [*usage-clause*] [*value-clause*]

```
[EQ  |component-name]
     |C-number

[FORM date-format]
```

*n* — is any valid COBOL level number greater than 01.

*alias* — is any valid COBOL data name, up to 30 characters.

*picture-clause* — **PICTURE [IS]** *picture*

*picture* — is any valid COBOL picture containing picture characters X, A, 9, V, or S. (See **Valid Formats of Picture Specifications** on page 5-25.)

*usage-clause* — **USAGE [IS]** *usage*

*usage* —

```
|DISPLAY
|COMP
|COMP-3
```

See Chapter 4, "SYSTEM 2000 Item Types and COBOL Data Descriptions" for the correspondence between usage and SYSTEM 2000 item types.

*value-clause* — **VALUE [IS]** *initial-value*

*initial-value* — is a literal compatible with both the subschema item description and the definition of the component in the database schema.

*component-name* — is the name of an item in the database schema.

*C-number* — is the letter C followed by a component number, up to four digits including leading zeros. For example, C0001 is interpreted as C1 by SYSTEM 2000 software.

*date-format* —

```
|DDMMYYYY     (default for 8-character dates)
|MMDDYYYY
|MMDDYY       (default for 6-character dates)
|DDMMYY
|YYYYMMDD
|YYMMDD
```

**EQ** |*component-name* |*C-number* — If you omit this specification, the alias is assumed to be a component name or C-number.

---

Abbreviation: PICTURE = PIC

| | |
|---|---|
| FORM *date-format* | If you omit this specification for an item of type DATE, the date format is assumed to be MMDDYYYY for an eight-character PICTURE and MMDDYY for a six-character PICTURE. |

## Valid Formats of Picture Specifications

| | |
|---|---|
| A(*i*) | where |
| X(*i*) | $1 \leq i \leq 250$ |
| 9(*n*) | $1 \leq n \leq 15$ |
| 9(*n*)V | $1 \leq m \leq 15$ |
| 9(*n*)V(*m*) | $n+m \leq 15$ |
| S9(*n*)V | |
| S9(*n*)V(*m*) | |

Repetitions of A, X, or 9 can be used in place of parenthetical expressions.

## Examples

Here are four examples of SUBSCHEMA RECORD descriptions for the EMPLOYEE database. Each example shows how the description actually appears in the source code of the program. In most situations, the source description is followed by the expanded description produced by the PLEX processor as input to the COBOL compiler.

### Example 1

The description of subschema record DIRECTORY includes four items, each of which has an appropriate PICTURE.

Note that the alias LNAME is equated with the C-number C2 and that the alias SSN is equated with the component name SOCIAL SECURITY NUMBER.

```
SUBSCHEMA RECORD DIRECTORY OF EMPLOYEE.
01  DIRECTORY.
    05  C1                  PIC 9(4).
    05  LNAME               PIC X(10).
        EQ C2.
    05  OFFICE-EXTENSION    PIC X(9).
    05  SSN                 PIC X(11).
        EQ SOCIAL SECURITY NUMBER.
```

The next page shows the same SUBSCHEMA RECORD description, augmented by the PLEX processor.

**Example 1 (continued)**

Note that the PLEX processor makes a comment out of the first two lines of the source description and renames the level-01 group item DIRECTORY-0. It inserts the level-02 control block. Also, the EQ lines giving the C-number and component name for two aliases are converted to comments.

```
*SUBSCHEMA RECORD DIRECTORY OF EMPLOYEE.
*01  DIRECTORY.
 01  DIRECTORY-0.
     02 DIRECTORY-1 PIC X(32) VALUE 'DIRECTORY'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +34.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +4.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +134217728.
     02 DIRECTORY-2 PIC X(32) VALUE 'C1'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +224.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +131076.
     02 DIRECTORY-3 PIC X(32) VALUE 'C2'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +172.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +65546.
     02 DIRECTORY-4 PIC X(32) VALUE 'OFFICE-EXTENSION'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +126
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +65545.
     02 DIRECTORY-5 PIC X(32) VALUE 'SOCIAL SECURITY NUMBER'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +79.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +65547.
     02 DIRECTORY.
     05  C1                      PIC 9(4).
     05  LNAME                   PIC X(10).
*        EQ C2
     05  OFFICE-EXTENSION        PIC X(9).
     05  SSN                     PIC X(11).
*        EQ SOCIAL SECURITY NUMBER.
```

### Example 2

In the description of subschema record PAYINFO, each subschema item has an alias equated with the component name or C-number. The program uses level number 02 for the subschema items. Also, note that a date format is specified for the item HIREDATE.

```
SUBSCHEMA PAYINFO OF EMPLOYEE.
01  PAYINFO.
    02  HIREDATE                PIC X(8).
        EQ HIRE DATE.
        FORM YYYYMMDD.
    02  VAC-TOTAL               PIC S999V99.
        EQ C11.
    02  SICK-TOTAL              PIC S999V99.
        EQ C12.
    02  SOC-SEC                 PIC X(11).
        EQ SOCIAL SECURITY NUMBER.
```

In the following output from the PLEX processor, note that the level numbers in the item descriptions have been increased to 03 so that the level-02 control block can be correctly inserted in the SUBSCHEMA RECORD description. A warning message is given.

```
*SUBSCHEMA PAYINFO OF EMPLOYEE.
*01  PAYINFO.
     03  HIREDATE                PIC X(8).
 ..........C
****      PLX202 ALL LOWER LEVEL NUMBERS HAVE BEEN INCREASED BY ONE
     *        EQ HIRE DATE.
     *        FORM YYYYMMDD.
         03  VAC-TOTAL            PIC X999V99.
     *        EQ C11.
         03  SICK-TOTAL           PIC S999V99.
     *        EQ C12.
         03  SOC-SEC              PIC X(11).
     *        EQ SOCIAL SECURITY NUMBER.
```

**Example 2 (continued)**

The following input to the COBOL compiler shows that the control block was inserted and that the FORM line and EQ lines were all turned into comments:

```
*SUBSCHEMA PAYINFO OF EMPLOYEE.
*01  PAYINFO.
 01  PAYINFO-0.
     02 PAYINFO-1 PIC X(32) VALUE 'PAYINFO'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +29.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +4.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +134217728.
     02 PAYINFO-2 PIC X(32) VALUE 'HIRE DATE'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +224.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +983048.
     02 PAYINFO-3 PIC X(32) VALUE 'C11'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +176.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +197125.
     02 PAYINFO-4 PIC X(32) VALUE 'C12'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +125
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +197125.
     02 PAYINFO-5 PIC X(32) VALUE 'SOCIAL SECURITY NUMBER'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +74.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +65547.
     02 PAYINFO.
     03  HIREDATE              PIC X(8).
*        EQ HIRE DATE.
*        FORM YYYYMMDD.
     03  VAC-TOTAL             PIC S999V99.
*        EQ C11.
     03  SICK-TOTAL            PIC S999V99.
*        EQ C12.
     03  SOC-SEC               PIC X(11).
*        EQ SOCIAL SECURITY NUMBER.
```

### Example 3

A SUBSCHEMA RECORD description can appear without items, regardless of whether the corresponding schema record has items. The name of an SSR that has no items must be either the component name or the C-number of the schema record. In this example, the subschema name is the C-number C400.

```
SSR C400 OF EMPLOYEE.
01  C400.
```

As expanded by the PLEX processor, the record description includes the control block.

```
*SSR C400 OF EMPLOYEE.
*01  C400.
 01  C400-0.
     02 C400-1 PIC X(32) VALUE 'C400'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +134217728.
```

### Example 4

In this example, REDEFINES and RENAMES clauses appear in the SUBSCHEMA RECORD description. The REDEFINES clause subdivides the HIREDATE item. The level-66 RENAMES clause groups together the two parts of the employee name.

```
SSR BOOKKEEPING OF EMPLOYEE.  01  BOOKKEEPING.
    05  LASTNAME          PIC X(16).
        EQ C2.
    05  FORENAME          PIC X(16).
        EQ C3.
    05  HIREDATE          PIC X(8).
        EQ HIRE DATE.
        FORM YYYYMMDD.
    05  HIRE-DATE-SUBDIVIDED REDEFINES HIREDATE.
        10  YEAR          PIC X(4).
        10  MONTH         PIC XX.
        10  DAYX          PIC XX.
    05  VAC-TOTAL         PIC S999V99.
        EQ C11.
    66  EMPLOYNAME RENAMES LASTNAME THRU FORENAME.
```

The PLEX processor generates the normal code, skipping over the REDEFINES clause and the subordinate items as shown on the next page.

```
*SSR BOOKKEEPING OF EMPLOYEE.
*01  BOOKKEEPING.
 01  BOOKKEEPING-0.
     02 BOOKKEEPING-1 PIC X(32) VALUE 'BOOKKEEPING'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +45.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +4.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +134217728.
     02 BOOKKEEPING-2 PIC X(32) VALUE 'C2'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +224.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +65552.
     02 BOOKKEEPING-3 PIC X(32) VALUE 'C3'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +184.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +65552.
     02 BOOKKEEPING-4 PIC X(32) VALUE 'HIRE DATE'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +144
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +98308.
     02 BOOKKEEPING-5 PIC X(32) VALUE 'C11'.
     02 FILLER PIC S9(9) USAGE COMP VALUE +96.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +0.
     02 FILLER PIC S9(9) USAGE COMP VALUE +197125.
     02 BOOKKEEPING.
     05  LASTNAME        PIC X(16).
*        EQ C2.
     05  FORENAME        PIC X(16).
*        EQ C3.
     05  HIREDATE        PIC X(8).
*        EQ HIRE DATE.
*        FORM YYYYMMDD.
     05  HIRE-DATE-SUBDIVIDED REDEFINES HIREDATE.
         10  YEAR        PIC X(4).
         10  MONTH       PIC XX.
         10  DAYX        PIC XX.
     05  VAC-TOTAL       PIC S999V99.
*        EQ C11.
     66  EMPLOYNAME RENAMES LASTNAME THRU FORENAME.
```

## Guidelines for SUBSCHEMA RECORD Descriptions

1. A SUBSCHEMA RECORD description can appear in either the WORKING-STORAGE SECTION or the LINKAGE SECTION of the DATA DIVISION. In a main program or calling program, it should be declared in WORKING-STORAGE, but in a called program it should be declared in the LINKAGE SECTION.

2. The SSR name can be different from the schema record name if the SSR has at least one item in it. SYSTEM 2000 software associates an SSR containing items with the corresponding schema record by associating item names, rather than by matching record names.

3. You can declare an SSR without any items in it. Use the following syntax:

   ```
   SUBSCHEMA [RECORD]  name OF database-identifier
   01  name
   ```

   You can use an SSR without items when there is no item in the schema record, or when retrieval of the data values corresponding to schema items is not required. You can use such an SSR in a where-clause or for address retrieval before an update. Since the SSR has no items, its name must be either the component name or the C-number of the corresponding schema record. In this situation, SYSTEM 2000 software associates the two records by matching their names. If the name of such a subschema record is not the component name or C-number of the corresponding SSR, any command referring to the SSR results in return code 13.

4. A program can have more than one SSR corresponding to the same schema record. The only requirement for these SSRs is that they have unique names (not necessarily the name of the corresponding schema record). If the name of an SSR is not unique within the program, the PLEX processor issues an error message.

5. If the SSR name specified in the SUBSCHEMA RECORD line does not match the name specified in the level-01 line, the PLEX processor issues an error message.

6. SSRs can be placed in the Source Library and copied into the program using the $COPY directive. See **COBOL PLEX Processor Directives** on page 19-2.

7. If an SSR will be passed to a called program, the calling program must include the SUBSCHEMA RECORD description in its WORKING-STORAGE SECTION; the called program must include the corresponding SUBSCHEMA RECORD description in its LINKAGE SECTION. A calling program that passes an SSR as a parameter must use the suffixed name of the SSR (such as SALARY-0) in its CALL statement, and the called program must include the suffixed name in its PROCEDURE DIVISION header or ENTRY statement.

8. A SUBSCHEMA RECORD description can include an item with a REDEFINES clause to redefine a subschema item already described in the SSR. The level number of the REDEFINES clause item must be the same as the level number of the subschema items. When the PLEX processor finds the keyword REDEFINES immediately following a data name, it skips item descriptions until it finds another item with the level number for subschema items. The PLEX processor does not check the validity of a REDEFINES clause.

---

Abbreviation: SUBSCHEMA RECORD = SSR

## Considerations for Subschema Items

1. If the alias of a subschema item is different from the corresponding schema item, you must use the EQ option to establish the correspondence between that alias and the component name or C-number of the schema item. If you do not specify the EQ option, the software issues return code 13 for each command that refers to that SSR. The first COMMBLOCK field (record name) contains the name of the invalid item, and the eighth COMMBLOCK field (database level) indicates the ordinal position of the invalid item in the SUBSCHEMA RECORD description.

2. The picture and COBOL usage of a subschema item must be compatible with the item type of the corresponding schema item. If you use a VALUE clause, the value provided must be compatible with the description. The PLEX processor does not check the validity of the picture, usage, or the initial values. If the description is not valid in COBOL, errors occur during compilation. Incompatibility between subschema item descriptions and database item types results in run-time errors.

3. When a program first references an SSR at execution time, the software validates all items in the SSR. If the picture or usage of an item is not compatible with the item type, picture, or size of the corresponding item in the database, the software places return code 17 in the COMMBLOCK for that subschema record. It also places the ordinal of that item (within its SSR) in the eighth COMMBLOCK field.

4. The same item alias can appear in more than one SUBSCHEMA RECORD description. In a where-clause or item list referring to a subschema item whose alias is not unique within the program, the alias must be qualified as: *alias* OF (or IN) *SSR*. If the alias is not properly qualified, the PLEX processor issues an error message.

5. The names of items used in a PLEX command must appear exactly as they do in the subschema item descriptions. The PLEX processor issues an error message if a command includes an alias (item name) that does not appear in an SSR.

   When you specify C-numbers as aliases for subschema items, SYSTEM 2000 software allows some flexibility that COBOL and the PLEX processor do not. For example, if you specify the alias C0100 for a subschema item corresponding to C100 in the database schema, you do not need to include the number C100 with EQ in the item description. SYSTEM 2000 software understands the implicit equivalence at execution time. When a PLEX command includes a C-number such as C0100, SYSTEM 2000 software knows that it refers to component C100 in the database. If you give the number C0100 as the alias, this number must be used in a PLEX command referring to that item. In this situation, if you use C100, the PLEX processor issues an error message because the alias is actually C0100 in the subschema item description.

   A C-number as an alias can have up to four digits after the C. For example, you can give the component whose C-number is C1 an alias of C1, C001, or C0001. **Note**: The C-numbers C01 through C12 are COBOL reserved words and cannot be used.

6. If an SSR refers twice to the same database item by specifying the same alias, component name, or C-number twice, the PLEX processor issues an error message. This error can occur in the following three situations:

   a) The same alias is declared twice in the same SSR.

   b) The SSR includes an EQ option specifying a component name or C-number that is also specified as the alias for another item in the same SSR.

   c) Two EQ specifications in the SSR specify the same component name or C-number.

7. Return code 13 occurs when a command refers to an SSR that contains two references to the same database item, for example, if the component name is specified in one item description and the associated C-number is specified in another item description. The same return code occurs if different forms of the same C-number appear in different item descriptions in the same SSR. This error is not detected in preprocessing, because the PLEX processor does not access the database definition and, therefore, does not detect the double reference. This error can occur in the following four situations within the same SSR:

   a) The component name and C-number are both used as item aliases.

   b) The component name or C-number is used as an alias, while its counterpart is named with the EQ option in another item description.

   c) The component name and C-number are named with the EQ option in two different item descriptions.

   d) The C-number C1, for example, is included in one item description, while a variant such as C001 is included in another item description.

   When return code 13 occurs, the first COMMBLOCK field (record name) contains the name of the invalid item, and the eighth COMMBLOCK field (database level) indicates the ordinal position of the invalid item in the SUBSCHEMA RECORD description.

8. The level numbers of all subschema items must be identical in any one SUBSCHEMA RECORD description. If the level number for subschema items is less than 3, the PLEX processor substitutes a level number that is greater by one (see Example 1). If the level numbers for all subschema items are not identical, the PLEX processor issues an error message.

   But a SUBSCHEMA RECORD description can include one or more items with REDEFINES or RENAMES clauses. The level number of an item with a REDEFINES clause must be the same as the level number of the subschema items. When the PLEX processor encounters the keyword REDEFINES immediately following a data name, it skips item descriptions until it either finds either another item with the level number of subschema items or the end of the SUBSCHEMA RECORD description. The PLEX processor does not check the validity of a REDEFINES clause or of items subordinate to a REDEFINES clause item.

Group items other than REDEFINES clause items cannot appear within a SUBSCHEMA RECORD description. However, one or more level-66 items with RENAMES clauses can appear in a SUBSCHEMA RECORD description to specify item grouping that the PLEX program might require for its own processing. If you use an item with a RENAMES clause in an SSR, it must appear at the end of the SUBSCHEMA RECORD description. The PLEX processor does not check the validity of a RENAMES clause.

9. When you specify a date format in an item description, the length of the item (either six or eight characters) must match the length of the FORM specification. If the lengths are not the same, the PLEX processor issues an error message.

10. You can describe up to 1000 items in one SSR, provided that all of these belong to the same schema record. If a program includes more than 1000 items in an SSR, the PLEX processor issues an error message.

11. When COMP-3 is the COBOL usage, the picture and decimal alignment in the subschema item must be the same as for the corresponding item in the database.

12. You can intersperse comments among the lines of a SUBSCHEMA RECORD description.

Chapter 6

# Initialization

*START S2K 6-1*
*Format 6-1*
*Default Values 6-2*
*Examples of Syntax 6-2*
*Examples of Usage 6-3*
*Considerations 6-6*

*OPEN 6-7*
*Format 6-7*
*Examples of Syntax 6-9*
*Examples of Usage 6-10*
*Considerations 6-11*

This chapter describes the PLEX commands that make a database accessible. The first section covers the syntax and semantics of START S2K, which is the first PLEX command executed in any program. **OPEN** on page 6-7 shows how to open a database explicitly by using the OPEN command.

## START S2K

The START S2K command requests extra Locate Files to be used by the program and specifies the environment in which the program will run. With START S2K the program can

- request up to 15 extra Locate Files
- specify either a single-user or Multi-User environment.

The program can indicate the number of Locate Files it requires in addition to Locate File 0, which is assigned to every program automatically. At execution time, the specified number of extra Locate Files is reserved for the program. These Locate Files are retained by the program until it issues the STOP S2K command.

START S2K must be issued before any other executable PLEX command. However, it can appear after the COMMBLOCK and SUBSCHEMA RECORD declarations.

### Format

```
START S2K [(extra-Locate-Files)] [USE (environment-number)]
```

*extra-Locate-Files* is the number of extra Locate Files requested: a constant or a variable whose value is greater than or equal to zero and less than or equal to 15 ($0 \leq n \leq 15$). For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*environment-number* is a constant or a variable with a numeric value of 0 (zero) for single-user or 1 for Multi-User. For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit integer.

## Default Values

- When the START S2K command does not specify extra Locate Files, the software provides only Locate File 0 by default.
- For a program that does not run in a TP environment, a single-user environment is the default when you omit USE (*environment-number*) from the START S2K command.
- A Multi-User environment is the default for a program that uses a TP interface.

Short Form of Command and its Equivalents

```
START S2K.                    (is the short form, using all defaults)
START S2K (0).
START S2K (0) USE (0).
```

## Examples of Syntax

**START S2K.** is the short form; specifies that the program requires no extra Locate Files and that the program is not running in a Multi-User batch environment. Instead, the program is running in either a single-user (USE (0)) or a TP environment where USE is ignored. Locate File 0 is automatically assigned.

**START S2K (4) USE (1).** specifies that the program requires four extra Locate Files (in addition to Locate File 0) and that it is running in either a Multi-User batch (USE (1)) or a teleprocessing environment (in which USE is ignored).

**START S2K (LFNUM) USE (ENV).** uses variables LFNUM and ENV to specify the required number of extra Locate Files and the environment. The program assigns appropriate values to these variables.

## Examples of Usage

This section gives sample usage syntax for the START S2K command. (See Appendix B for examples of PL/I usage of the START S2K command.)

Program 1: Use of constants in START S2K.

```
WORKING-STORAGE SECTION.
      :
      :
   COMMBLOCK OF . . .
      :
      :
   SSR . . .
      :
      :
PROCEDURE DIVISION.
      :
      :
   START S2K (5) USE (1).          (First PLEX command executed,
                                   preceding all other PLEX
                                   commands; requests five
                                   Locate Files in addition to
                                   Locate File 0;
                                   specifies Multi-User)

(Check S2KRTC for return code)

OPEN . . .
      :
      :
(Program statements)
```

Program 2: Use of variables in START S2K.

```
WORKING-STORAGE SECTION.
      .
      .
   COMMBLOCK OF. . .
      .
      .
   SSR . . .
      .
      .
  01 LFNUM PICTURE S99 COMP.
  01 ENVIRON PICTURE S99 COMP.
      .
      .
PROCEDURE DIVISION.
      .
      .
   MOVE 13 TO LFNUM.
   MOVE 0 TO ENVIRON.
  START S2K (LFNUM) USE (ENVIRON). (First PLEX command executed;
                                    requests 13 (LFNUM=13) extra
                                    Locate Files; specifies
                                    single-user (ENVIRON=0))
(Check S2KRTC for return code)

OPEN . . .
      .
      .
(Program statements)
      .
      .
```

Program 3: Use of a parameter for environment number.

To enable your COBOL PLEX program to be run either as a single-user job or in a Multi-User environment without re-compilation, pass a parameter to the program from the JCL to specify the environment in the START S2K command. This sample EXEC job control statement specifies the parameter 1 for Multi-User. The parameter becomes available in the LINKAGE SECTION. The PLEX program can use the parameter value in the START S2K command, as shown here. This sample program logic provides for the situation in which a parameter is not specified (PARM-LENGTH IS EQUAL TO 0) in the JCL; zero is moved to ENVIRON-NUM for the START S2K command.

```
//EXEC . . . PARM.GO='1'
      :
      :
DATA DIVISION.
WORKING-STORAGE SECTION.
01  ENVIRON-NUM                PIC 9   COMP-3.
      :
      :
LINKAGE SECTION.
01 PARMS.
   05   PARM-LENGTH                PIC S9(4)  COMP.
   05   ENVIRON-NUM-IN             PIC 9.
      :
      :
PROCEDURE DIVISION USING PARMS.
      :
      :
     IF PARM-LENGTH IS EQUAL TO 0
         MOVE 0 TO ENVIRON-NUM
     ELSE
         MOVE ENVIRON-NUM-IN TO ENVIRON-NUM.

     START S2K . . . USE (ENVIRON-NUM).
     (Check S2KRTC for return code)
```

## Considerations

1. Check the return code after you issue the START S2K command. Since START S2K does not refer to a particular database, its return code is not placed in a COMMBLOCK. Instead, the return code is placed in S2KRTC, a variable in the S2KDUM control block.

2. If a called PLEX program does not include the START S2K command, the PLEX processor issues a warning message. The called program can still be processed, compiled, and executed. However, some routine in the main program must include START S2K, which must be executed before any other PLEX command in the program.

3. In order to execute the main PLEX program, it must issue one and only one START S2K command. A START S2K command issued after a STOP S2K command or after a previous START S2K command, produces unpredictable results.

4. In the START S2K command, the subscript for the requested number of Locate Files is the highest Locate File number that you can use in a LOCATE command. If a LOCATE command includes a number higher than this subscript, the software issues return code 40 when you issue the LOCATE command.

5. If the requested number of Locate Files is negative or greater than 15, the software issues return code 40 when you issue START S2K.

6. If a program issues another PLEX command before it issues START S2K, that other command is ignored, and return code 44 occurs for it.

7. A program that runs in a Multi-User batch environment must specify USE (1) in the START S2K command. However, a program that uses a teleprocessing interface can omit USE (1) from the command because the interface ignores this specification and assumes a Multi-User environment.

8. In either a single-user or a Multi-User environment, when the number of Locate Files (in addition to the default Locate File 0) is unavailable, a SYSTEM 2000 Error Code occurs. It can be caused either by a JCL error or, in a Multi-User session, by an excessively busy system in which the allocated number of Locate Files cannot accommodate all of the concurrent PLEX programs.

9. In a program running in a teleprocessing environment, the START S2K command must be given before all executable commands, including FOR and LINK. In other environments, you can issue FOR and LINK commands before START S2K.

10. The START S2K command establishes a user synchpoint. User synchpoints are used by the Coordinated Recovery feature when the Rollback Log is enabled. For details, see the ENABLE/DISABLE ROLLBACK, COMMIT, ROLLBACK, and RESET ROLLBACK commands. See also the Coordinated Recovery discussion in *SYSTEM 2000 CONTROL Language, Version 12, First Edition.*

# OPEN

The OPEN command makes a database available to the program. If rollback is enabled for a database and the database becomes damaged, an OPEN command causes rollback processing to occur.

- You can request exclusive use of a database in a Multi-User environment.
- The program can wait until the requested database becomes available, or it can perform PLEX processing on other databases.

To prepare for the opening of a database, place a valid password in the fifth COMMBLOCK field for the database. Then issue the OPEN command to make the database available to the program. Up to 63 databases can be available to a program at one time. If a program already has the maximum number of databases available to it, you must issue a CLOSE command for one of those databases before an additional database can be made available.

## Format

```
|OPEN  [/EXCLUSIVE]   database
|OPENR
```

*database* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

exclusive use

In a Multi-User environment, a database is usually accessed by several users concurrently. However, occasionally a PLEX program contains retrievals or updates for a certain database that must not interfere with (or be affected by) operations of other users of the same database. Such a program should open the database for its own exclusive use. A database opened for exclusive use by a program cannot be accessed by any other user. To request exclusive use, specify the EXCLUSIVE (or LOCK) option in the OPEN command. (This option has no significance in a single-user environment.) A program is allowed exclusive use of a database only when both of the following conditions are met: 1) no other users currently have the database open and 2) the program itself has no databases already open. After a program obtains exclusive use of a database, it retains the privilege until it issues a CLOSE command for that database.

OPEN and OPENR

If the database requested in an OPEN command is already under exclusive use by another user, two different results can occur, depending on which of the two forms of the command the program specifies. One form begins with OPEN, the other with OPENR. If the program specifies OPEN, it will enter a wait state until the database becomes available; that is, program execution will be suspended. However, if the

Abbreviation: EXCLUSIVE = EXCL, LOCK

program specifies OPENR, SYSTEM 2000 software issues return code 94, which the program can check. In this situation, after the program detects the return code, it can continue to execute and go on to other processing functions. Neither OPEN nor OPENR has any effect on stacks.

damage flag

In addition to checking the return code after the program gives the OPEN (or OPENR) command, it should also check the damage flag, which is the fourteenth COMMBLOCK field for the requested database. When the value in this field is 0, the database is not damaged so both updates and retrievals can be processed. But when the value of the damage flag is 1, the database is damaged. Retrievals are allowed, but not updates.

If an OPEN (or OPENR) command refers to a damaged database that has rollback enabled, the command causes rollback processing to occur. After rollback processing is completed, the software resets the damage flag to 0, and return code 102 informs the program that rollback processing took place.

other COMMBLOCK fields

Besides the damage flag and the return code, five other COMMBLOCK fields are set when an OPEN command is processed:

| | |
|---|---|
| Field 9 | (clock time of the last update to the database) |
| Field 10 | (date of the last update to the database) |
| Field 11 | (the current database cycle number) |
| Field 12 | (the system separator used for the database) |
| Field 13 | (the entry terminator used for the database) |

allocating database files dynamically

You can let SYSTEM 2000 software dynamically allocate the database files automatically. For dynamic allocation of the existing database files, the data set names must fit the default naming convention. If they do and you want DISP=OLD, you do not need to allocate the files in the JCL or CLIST. The software look for the existing files automatically when you issue the OPEN or OPENR command.

The default data set names and disposition are as follows:

**DSN=***prefix.database*,**DISP=OLD**

where *database* is the first seven characters of the database name (with embedded blanks replaced by Xs), followed by an integer from 1 through 8.

SYSTEM 2000 software tries to locate existing database files with the steps given below. Files 1 through 6 are required, although the software searches for File 7 and 8 also.

- Sees whether the files were allocated in the JCL or CLIST; if so, uses those allocations.
- Gives user exit EXIT03 a chance to allocate the files dynamically.
- Looks for permanent data sets, using the default data set names or the DSN specified in the S2KDBCNT table. For details about the S2KDBCNT table, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*

At this point, if the files are still not allocated, an error message appears: -766-DATA BASE OPEN FAILED FOR *database*.

## Examples of Syntax

| | |
|---|---|
| **OPEN EMPLOYEE.** | In a Multi-User environment, requests a database called EMPLOYEE for non-exclusive use, allowing other users to access it. If the database is not available, the program is put in a wait state until it becomes available. |
| **OPEN/EXCLUSIVE PARTS.** | In a Multi-User environment, requests the PARTS database and also asks for exclusive use of that database. If other users already have it open for exclusive or non-exclusive use, the program is placed in a wait state until all other programs issue CLOSE commands for the database. |
| **OPENR/EXCL SCHOOLS.** | In a Multi-User environment, requests the SCHOOLS database and also asks for exclusive use of that database. If other users already have it open either exclusively or non-exclusively, SYSTEM 2000 software issues return code 94 to that effect because OPENR is specified. The program continues to execute. |

## Examples of Usage

Program 1: Use of OPEN to request exclusive use of a database.

```
        .
        .
        .
(Program statements)
        .
        .
        .
COMMBLOCK OF EMPLOYEE.
        .
        .
        .
   03 RETURN-KODE.
        .
        .
        .
   03 DAMAGE-FLAG.
        .
        .
        .
SUBSCHEMA RECORD. . .OF EMPLOYEE.
        .
        .
        .
   START S2K.
(Check S2KRTC return code)
  OPEN/EXCLUSIVE EMPLOYEE.
(Check RETURN-KODE)
   IF DAMAGE-FLAG = 1 GO TO REPORT-DAMAGE.
                                  (If the database is damaged,
                                  go to a routine that reports
                                  the condition.)
        .
        .
        .
(Program statements)
        .
        .
        .
```

Program 2: Use of OPENR to obtain a return code if another program has exclusive use of the requested database.

```
      :
(Program statements)
      :
COMMBLOCK OF EMPLOYEE.
      :
  03 RETURN-KODE.
      :
  03 DAMAGE-FLAG.
      :
SUBSCHEMA RECORD. . .OF EMPLOYEE.
      :
 START S2K.
(Check S2KRTC return code)

OPENR EMPLOYEE.
IF RETURN-KODE =. . .GO TO OPEN-NEXT-DB.  (If RETURN-KODE shows that
                                           another program has exclusive
                                           use of the EMPLOYEE database,
                                           go to a routine that tries
                                           to open another database
                                           instead.)
      :
IF DAMAGE-FLAG = 1 GO TO REPORT-DAMAGE.   (If the database is damaged,
                                           go to a routine that
                                           reports the condition.)
```

## Considerations

1. Check the return code after you give the OPEN command. The return code is placed in the second COMMBLOCK field for the database to which the command refers. In addition, check the damage flag (fourteenth COMMBLOCK field) to determine whether the database is damaged.

2. A database can be opened implicitly, that is, without giving an OPEN command for it. If you do not issue an OPEN command for a database, the first command that refers to that database (either by name or by referring to a subschema record declaration for the database) causes an implicit opening.

   When an implicit opening occurs, the program cannot check the damage flag before it attempts to process the database. Neither can it specify that the database be opened for exclusive use. Therefore, it is preferable to give an OPEN command for each database to be used, so you can check the damage flag before you attempt any update operations, and so you can specify the EXCLUSIVE option when necessary.

3. A valid password for the database must be present in the COMMBLOCK when the database is opened. If it is not present, the software issues SYSTEM 2000 Error Code 24 and terminates the program. This password requirement applies to opening of a database both explicitly with an OPEN command and implicitly with some other command that refers to the database.

4. If your program attempts to open more than 63 databases simultaneously (by explicit or implicit opening), the software issues return code 92. In addition, the command that caused the program to request more than the maximum number is ignored.

5. Before your program can open a database for exclusive use, it must give CLOSE commands for all databases that were already opened. If all other databases are not closed, the software ignores the command specifying exclusive use and issues return code 95.

6. If your program already has a database open for exclusive use, it cannot open another database, either explicitly or implicitly. The program must close the database open for exclusive use before attempting to open another. If it does not, return code 95 results. In addition, the command that would have caused the explicit or implicit opening of the other database is ignored.

7. If your program has a database open for exclusive use and issues a FRAME command listing a database other than that one, the command is ignored and return code 96 results. The software sends this return code to the COMMBLOCK of the first listed database that is not under exclusive use.

8. While any program has a certain database open, no other program can open that database for exclusive use. If a program attempts to open it with an OPENR/EXCLUSIVE command, return code 94 results, and the command is ignored.

9. Similarly, while one program has a database open with exclusive use, no other program can open that database. If a program attempts to open it with an OPENR command, return code 94 results, and the command is ignored.

10. A database can be opened with the FRAME command.

11. An OPEN command causes rollback processing to occur for the specified database if rollback was enabled for the database and the database damage flag is set to 1. During rollback processing, execution of the program is suspended. After rollback is completed, the software issues return code 102, indicating that the rollback processing was completed. The program can then check the database cycle number in the eleventh COMMBLOCK field.

12. When you issue an OPEN or FRAME command, the eighth COMMBLOCK field for each specified database is set to show whether rollback is enabled for that database. If the eighth field equals 1, rollback is enabled; if it equals zero, rollback is not enabled. However, this field does not indicate whether the command actually caused rollback processing to occur for the database.

13. If an OPEN command specifies no database name or an invalid database name, the PLEX processor assumes it to be a COBOL OPEN statement, generates no code for the command, and issues no error message. The error is caught by the COBOL compiler.

14. If you issue an OPEN command after a RELEASE command for the same database, without an intervening RESTORE, the software ignores the OPEN command and issues return code 14.

15. An OPEN command terminates an insert sequence if it refers to the same database as that of the insert sequence.

16. An OPEN command or a command that causes an implicit opening of a database might be ignored because of incorrect JCL specifications. In this situation, return code 93 results. The region as specified might not be large enough, or a JCL DDname might be missing or incorrect.

17. After an aborted load mode session for a database, automatic error recovery takes place the next time the database is opened. As a security measure, the installation can force recovery to write binary zeros on each Data Table page. When the software writes binary zeros on the last page of the Data Table, it issues a warning message. If the message is issued after an OPEN command or a command that causes an implicit open, it indicates that error recovery was completed.

18. If the Rollback Log is enabled, an exclusive open of the database causes a database synchpoint and resets the Rollback Log for that database. The exclusive open also causes a user synchpoint, and any updates performed by that program since the last user synchpoint (that is, logical units of work) are committed and permanently recorded in the affected databases. For details see the ENABLE/DISABLE ROLLBACK, COMMIT, ROLLBACK, and RESET ROLLBACK commands. See also the Coordinated Recovery discussion in *SYSTEM 2000 CONTROL Language, Version 12, First Edition.*

Chapter 7

# Where-Clauses


*FIXED WHERE-CLAUSES 7-2*
- *Format 7-5*
- *Examples 7-7*
- *Considerations 7-8*
- *The Unary Operator EXISTS 7-10*
  - *Format 7-10*
- *The Unary Operator FAILS 7-10*
  - *Format 7-10*
- *The Binary Operators EQ, NE, GE, GT, LE, and LT 7-11*
  - *Format 7-11*
  - *Examples 7-12*
- *Item-to-Item Comparisons 7-13*
  - *Format 7-13*
- *The Ternary Operators EQ, NE, and SPANS 7-13*
  - *Format 7-13*
  - *Example 7-14*
- *The AND and OR Operators 7-15*
  - *Format 7-15*
  - *Examples 7-17*
- *The NOT Operator 7-18*
  - *Format 7-18*
- *The HAS Operator 7-19*
  - *Format 7-19*
  - *Examples 7-20*
- *The AT Operator 7-23*
  - *Format 7-23*
  - *Relative position under a parent 7-23*
  - *AT processing 7-24*
  - *Use of AT 7-24*
  - *Examples 7-24*
  - *Considerations 7-26*
- *Use of SAME LOCATE, SAME STACK and SAME POSITION 7-27*
  - *Format 7-27*
  - *Examples 7-27*
  - *Considerations 7-28*
- *Constructing a Where-clause 7-30*
  - *Precedence of operators 7-30*
  - *The processing of logical operators 7-32*
  - *Key vs. non-key 7-32*
  - *Placement of conditions 7-33*
  - *Preventing full passes of the Data Table 7-34*

*DYNAMIC WHERE-CLAUSES 7-35*
- *Format 7-35*
- *Examples 7-36*
- *Considerations 7-38*

*How To Code a DYNAMIC Where-Clause 7-39*
*Opcodes for DYNAMIC Where-Clause 7-39*
*Error Handling 7-40*

The where-clause specifies the conditions that select a subset of a database for retrieval or update. You can specify a where-clause with the two commands LOCATE and GET1. With GET1, the subset is a single data record. With LOCATE, the subset contains several data records corresponding to one schema record.

PLEX provides two types of where-clauses for data selection with the GET1 and LOCATE commands. The fixed where-clause must be hard-coded and compiled into the program. The DYNAMIC where-clause (QUEST where-clause syntax) uses selection criteria that can be supplied or changed during program execution. Although these where-clauses have many similar capabilities, each has its advantages and is implemented in different ways with different syntax requirements.

**Note**: The single key condition in a GETP command is a special instance of the fixed where-clause; GETP does not allow a DYNAMIC where-clause.

This chapter first discusses fixed where-clauses in detail. The last section discusses DYNAMIC where-clauses.

# FIXED WHERE-CLAUSES

The fixed where-clause offers the following features:

- Unary operators (EXISTS, FAILS) verify the existence or nonexistence of values.
- Binary operators (EQ, NE, LE, LT, GE, GT) compare data items with program variables or with other data items.
- Ternary operators (EQ, SPANS, NE) compare data items with a range of values.
- The HAS operator changes the database level of retrieval for a condition.
- The AT operator enables you to specify the position of a data record under its parent.
- You can use the logical operators AND, NOT,and OR to combine conditions.
- You can use both key and non-key items as operands.
- You can reuse the addresses in a Locate File or a primary stack entry in where-clause processing.
- You can use parentheses to control the order of processing.

A where-clause contains the keyword WHERE, followed by one or more expressions. Each expression is made up of one or more conditions referencing data items. Data records meeting the conditions in a where-clause are called qualified data records. A condition uses an operator, such as EXISTS, FAILS, EQ, NE, LT, LE, GT, GE, or SPANS, to compare values. The result of processing a condition is a list of qualified data records. The operators AT,

HAS, NOT, AND, and OR operate on the results (lists) produced by conditions and derive new lists of qualified data records.

SYSTEM 2000 software uses two different methods for processing conditions. Depending on its processing method, a condition is either a key condition or a non-key condition. Key conditions take advantage of the Index to produce lists of qualified data records. Non-key conditions require a search through every entry in the Data Table to determine which data records qualify. A key condition always involves a key item. A non-key condition always contains one of the following:

- a non-key item
- the keyword NON-KEY
- the FAILS operator
- a comparison of values for two database items.

A key condition is treated as a non-key condition when it is prefixed with the keyword NON-KEY. Illus. 7.1 lists this information in a table. See also **Key vs. non-key** on page 7-32 for more information about non-key expressions and processing. (Processing for the where-clause in a GETP command is different than processing where-clauses for GET1 and LOCATE. See **GETP** on page 8-151.)

**Illus. 7.1** Key Conditions and Non-key Conditions

| Condition | Is a key condition | Is a non-key condition |
|---|---|---|
| *item-alias* EXISTS | when the item for *item-alias* is key | when the item for *item-alias* is non-key |
| *item-alias* FAILS | never | always |
| *item-alias* [*binary-operator* [*variable*]] | when the item for *item-alias* is key | when the item for *item-alias* is non-key |
| *item-alias ternary-operator variable* * *variable* | when the item for *item-alias* is key | when the item for *item-alias* is non-key |
| *item-alias* * *binary-operator item-alias* * | never | always |
| NON-KEY *condition* | never | always |

The following factors determine the order in which the conditions in a where-clause are evaluated by the software:

- operator precedence
- use of parentheses
- type of the condition (key or non-key).

When a list is modified by more than one operator, the software honors the following precedence:

```
AT      (first)
HAS
AND
OR      (last)
```

You can overrule this precedence with parentheses, which take precedence over all operators. The software processes the conditions within the innermost pair of parentheses first, followed by those within the surrounding pair, thus moving outward until all parentheses are accounted for.

When all conditions are joined by the same operator, key conditions are processed first from right to left. Non-key conditions are processed later.

As mentioned before, conditions in a where-clause produce lists of qualified data records. To be more specific, the lists contain addresses of data records and not the actual values of data records.

The data records in a list are usually from one schema record. (There are some exceptions, which are pointed out in appropriate topics in this chapter.) In this chapter, the terminology schema record of a list means the schema record of the data records in the list, disregarding the exceptions. Illus. 7.2 shows the schema records of lists produced by different kinds of expressions in a where-clause.

Details of where-clause processing are presented with examples in the following sections.

**Illus. 7.2** Schema Record of a List of Qualified Data Records

| Expression | Schema Record of Resulting List |
|---|---|
| *condition* | schema record of the item specified in the condition |
| | schema record of the item at the lower level if the condition involves two items. |
| **NOT** *expression* | schema record of the list produced by *expression.* |
| *subschema-record* **HAS** *expression* | schema record corresponding to *subschema-record.* |
| *expression* **AT** *position* | schema record corresponding to *expression.* |
| *expression1* **[ , [AND]**<br>**[,] AND** *expression2*<br>**[,] OR** | schema record of the list produced by *expression1* or schema record of the list produced by *expression2*, whichever is at a deeper level. |

## Format

```
WHERE expression
```

```
expression    |condition
              |(expression)
              |NOT expression
              |expression | , [AND]  expression
                          | [,] AND
                          | [,] OR[2]
              |subschema-record HAS expression
              |expression AT[1] position
              |SAME[2]  |Locate-File
                        |stack

condition     [NON-KEY] item-alias|[unary-operator]
                                  |[binary-operator [variable]]
                                  |[* binary-operator item-alias*]
                                  |[ternary-operator variable*variable]

unary-operator  |EXISTS
                |FAILS
```

*binary-operator* is any one of the following: EQ, NE, GE, GT, LE or LT. You can also use their logical symbols as shown below.

| Operator | Symbol |
|---|---|
| EQ | = |
| GE | >= or => or ¬< or !< |
| GT | > |
| LE | <= or =< or ¬> or !> |
| LT | < |
| NE | ¬= or != |
| AND | & |
| OR | \| |
| NOT | ¬ or ! |

*ternary-operator* is any one of the following: EQ, NE, SPANS.

*variable* is a variable name. It can be in a subschema record. For COBOL, the variable is an 01-level variable, up to 30 characters.

---

[1] See **The AT Operator** on page 7-23 for restrictions on this syntax for the AT operator.

[2] See **Use of SAME LOCATE, SAME STACK and SAME POSITION** on page 7-27 for restrictions on SAME with NOT and OR.

Synonyms:

HAS = HAVE, HAVING
EXISTS = EXIST, EXISTING
FAILS = FAIL, FAILING
SPANS = SPAN, SPANNING

*item-alias* is the name used in a subschema item description. If this name is used in more than one subschema record description, it must be qualified with the SSR name. For example, if SSR POSREC and SSR SALARY both contain items called PDATE, the items must be qualified as PDATE OF (or IN) POSREC and PDATE OF (or IN) SALARY when they appear in where-clauses either separately or together.

*subschema-record* is the name of a subschema record.

*position* is a fullword variable containing a value equal to or greater than zero. For COBOL, the variable must be COMPUTATIONAL.

*Locate-File* **LOCATE** (*file-number*)

*stack* **STACK** (*stack-number*)

*file-number* is a fullword variable representing the number of the Locate File that will serve as a where-clause condition. For COBOL, the variable must be COMPUTATIONAL.

*stack-number* is a fullword variable representing the number of the stack whose primary entry will serve as a where-clause condition. For COBOL, the variable must be COMPUTATIONAL.

A long where-clause expression can be made shorter using the notation shown here. Short forms do not save any processing time; they are a convenience only.

| Short Forms | Equivalent Expanded Forms |
|---|---|
| **(a) in an expression:** | |
| *condition, condition* | *condition* **AND** *condition*<br>*condition,* **AND** *condition* |
| *condition* **OR** *condition* | *condition,* **OR** *condition* |
| **(b) in a comparison:** | |
| **C1** | **C1 EQ C1** |
| **C1** *binary-operator* | **C1** *binary-operator* **C1** |
| **(c) a combination of (a) and (b):** | |
| **C1, C2** | **C1 EQ C1 AND C2 EQ C2** |
| **C1 LE, C2** | **C1 LE C1 AND C2 EQ C2** |

Abbreviations and synonyms:
LOCATE = LOC
STACK = POSITION, POS

## Examples

In the following examples, C*n* denotes an item alias. A and B are variables. SSR*n* denotes a subschema record name.

```
C1 EXISTS
```

```
C1                                    (condition with assumed EQ and variable C1)
```

```
C1 EQ A
```

```
C1 SPANS A * B
```

```
C1* LT C2*                            (item-to-item comparison)
```

```
C1 GE D AT POS1                       (expression with AT; POS1 is a variable)
```

```
SSR1 HAS C2 FAILS                     (expression with HAS)
```

```
C1 EXISTS AND C2 FAILS                (expression with two conditions joined by AND)
```

```
C1 EXISTS AND C2 EQ A
OR C3 LE B AT POS2
```

```
C1 EQ A AND C3 FAILS AND (SSR2        (expressions with combinations of
HAS (SSR1 HAS C57 EQ B) AT POS1)      different operators)
AT POS2 AND NOT C137 EXISTS
```

```
C2 EQ A AND C3 EQ B AT POS1           (expression with two ATs; one AT
AT POS2                               must specify 0, meaning LAST)
```

## Considerations

1. The program's password must have W-authority for items in the where-clause.

2. A where-clause must not contain more than 80 operands. EQ and NE always count as having three operands, even when they have only two. If more than 80 operands are present, the PLEX processor issues an error message. Furthermore, the total storage requirement of the variables used in a where-clause must not exceed 368 words.

3. A where-clause must not contain more than 58 operators. A comma is counted once as an operator when used for AND and is counted twice when used as an EQ and an AND. If more than 58 operators are present, the PLEX processor issues an error message. Furthermore, the total storage requirements for operators must not exceed 400 words. Here are the storage requirements of the operators.

| Operators | Words |
|---|---|
| Unary operators | 10 |
| Binary operators | 10 |
| Ternary operators | 10 |
| AT | 10 |
| HAS | 10 |
| AND | 1 |
| OR | 1 |
| NOT | 1 |
| ( | 1 |
| ) | 1 |

4. The HAS operator cannot be nested more than 14 levels in an expression. Also, if a subschema record name does not precede the HAS operator, the PLEX processor issues an error message.

5. Under Security by Entry, the first (or the only) condition in the where-clause must be a binary condition with EQ. The left operand of the EQ must be the alias for the Entry Key item. The right operand must be a variable whose value is the unique value of the Entry Key item. The Entry Key condition must not be joined to another expression with the OR operator and must not appear in a non-key expression. If any one of these requirements is not met, return code 29 results.

   You can establish a primary stack entry on one or more databases with a GET1 that specifies an appropriate Entry Key value. Secondary stack entries (established with GETA or GETD) can be established only on a database for which there is a primary stack entry. SYSTEM 2000 software uses the primary stack entry to determine a program's accessibility to records that can be established as secondary stack entries. Therefore, a program can alternate the processing of commands that establish secondary stack entries from one database to another.

Primary stack entries usually require a qualification on the Entry Key. A subsequent primary stack entry within the same logical entry can be established without respecifying the Entry Key only if no retrieval commands for another database were processed after the Entry Key qualification.

6. The operands of the binary and ternary operators must be variables of compatible item types. See **Item Types and COBOL Data Descriptions of Items in Where-clauses** on page 4-7 for a discussion of the compatibility of item types for a where-clause comparison. If the operand to the right of a binary or ternary operator is a literal instead of a variable, an error results from the command during compilation.

7. When used with a GET1 command, the where-clause should qualify a single data record. If two or more data records qualify, the first data record (in physical order) is retrieved by the GET1 command.

8. The ordering of conditions and the use of parentheses strongly impact the efficiency of the where-clause. **Constructing a Where-clause** on page 7-30 provides some guidelines for placing the conditions in a where-clause to obtain better performance.

9. If the first operand in a condition was not declared in a subschema record, the PLEX processor issues an error message.

10. If a where-clause refers to a database other than the one for the specified SSR, the PLEX processor issues an error message.

11. If a where-clause refers to a subschema item whose name is not unique in the program, the item name must be qualified with the subschema record name. Suppose items named STARVAL appear in the descriptions of two different subschema records named FIRSTSSR and SECONDSSR. In where-clauses, the item names must be qualified as STARVAL OF (or IN) FIRSTSSR and STARVAL OF (or IN) SECONDSSR. The PLEX processor issues an error message if duplicate item names are not qualified.

12. In a command with a where-clause, the specified subschema record must correspond to a schema record in the same path indicated by the schema records and items referred to in the where-clause conditions. If the records and items referred to in the conditions do not fall in one path, or if the specified subschema record is not in the indicated path, the command is ignored and return code 58 results.

13. If the where-clause is in error and a non-zero return code results, the left-to-right ordinal position of the erroneous expression in the where-clause is indicated in the eighth field of the COMMBLOCK.

14. Unlike LOCATE and GET1 commands, GETP does not build and manipulate lists of qualified records. GETP uses the Index to select a data record according to a single key condition where-clause. For more information about the GETP command, see **GETP** on page 8-151.

## The Unary Operator EXISTS

### Format

*item-alias* **EXISTS**

The operand of the EXISTS operator is the alias of a schema item included in a subschema record. The EXISTS operator finds all data records that contain a value for the specified item.

For example, to find all employees in the EMPLOYEE database with some security clearance, you could give the where-clause WHERE C13 EXISTS.

If the specified item is a key item and you did not specify NON-KEY, the qualified data record addresses are determined by looking at the values in the Index. If the specified item is non-key or if you specified NON-KEY, the software determines the qualified data records by looking at the values in the Data Table.

## The Unary Operator FAILS

### Format

*item-alias* **FAILS**

The operand of the FAILS operator is the alias of a schema item included in a subschema record. The FAILS operator finds the places in the database where a value for the given item could exist but does not. This can happen in the following two situations:

- The item is null.
- A data record exists that has no descendent data records corresponding to the schema record that the item belongs to.

That is, FAILS qualifies the data records containing the null item and also the nearest ancestors of nonexisting data records. For example, in the EMPLOYEE database, WHERE C13 FAILS finds all employees without a security clearance (nulls for C13). The where-clause WHERE C421 FAILS finds those employees who have no formal training (nonexisting data records for C420).

A condition with FAILS is always a non-key condition, that is, SYSTEM 2000 software determines the qualified data records by looking at the values in the Data Table, not the Index. Therefore, FAILS can be a very expensive operation. The expression NOT *item-alias* EXISTS gives the same result as *item-alias* FAILS. See **The NOT Operator** on page 7-18 for NOT processing.

Synonyms:
FAILS = FAIL, FAILING
EXISTS = EXIST, EXISTING

## The Binary Operators EQ, NE, GE, GT, LE, and LT

### Format

*item-alias* [*binary-operator* [*variable*]]

A binary operator compares a data item with a given value or with another item in the database. The operands for the binary-operator are the item corresponding to the specified item alias and the value of the variable.

The variable can be in a subschema record, that is, the variable can be an item alias, too. If so, the value stored in that item alias would be the second operand (not the values in the database corresponding to that item). For COBOL, the variable must be an 01-level variable. The values to be compared must be compatible. See Chapter 4, "SYSTEM 2000 Item Types and COBOL Data Descriptions" for information about item types and COBOL data descriptors. For FORTRAN, PL/I, and Assembler data types, see the appropriate appendix.

EQ qualifies data records containing values for the given item that are equal to the value of the variable.

NE qualifies data records containing values for the given item that are different from the value of the variable.

GE qualifies data records containing values for the given item that are greater than or equal to the value of the variable.

GT qualifies data records containing values for the given item that are greater than the value of the variable.

LE qualifies data records containing values for the given item that are less than or equal to the value of the variable.

LT qualifies data records containing values for the given item that are less than the value of the variable.

**Note**: Binary operators can also be used in item-to-item comparisons (see **Item-to-Item Comparisons** on page 7-13).

You can also use the following logical symbols for the binary operators:

| Operator | Symbol |
|---|---|
| EQ | = |
| GE | >= or => or ¬< or !< |
| GT | > |
| LE | <= or =< or ¬> or !> |
| LT | < |
| NE | ¬= or != |
| AND | & |
| OR | \| |
| NOT | ¬ or ! |

**Examples**

Suppose a program contains the following statements:

```
DATA DIVISION.
       :
       :
COMMBLOCK OF EMPLOYEE.
       :
       :
SUBSCHEMA EMPLDATA   OF  EMPLOYEE.       (SSR for C0 data records)
01     EMPLDATA.
       03 EMPNUMBER        PIC 9(4)  COMP-3.
          EQ EMPLOYEE      NUMBER.
       03 EMPSTATUS        PIC X(9).
          EQ EMPLOYEE      STATUS.
       03 HIREDATE         PIC X(6).
          EQ HIRE DATE.
       :
       :
01     NUM                 PIC 9(4)  COMP-3.
01     STA                 PIC X(9).
01     HDATE               PIC X(6).
       :
       :
```

Then the binary operators qualify data records as shown here.

```
MOVE 1265 TO NUM.

  . . . WHERE EMPNUMBER EQ NUM.   (qualifies the C0 data record where the value of
                                  EMPLOYEE NUMBER is equal to 1265)

MOVE 'FULL TIME' TO STA.

  . . . WHERE EMPSTATUS NE STA.   (qualifies the C0 data records where the value
                                  of EMPLOYEE STATUS is not equal to FULL
                                  TIME)

MOVE '010189' TO HDATE.           (is in the format MMDDYY, that is, month, day,
                                  year)

  . . . WHERE HIREDATE GE HDATE.  (qualifies the C0 data records where HIRE DATE
                                  is greater than or equal to January 1, 1989)

  . . . WHERE HIREDATE GT HDATE.  (qualifies the C0 data records where HIRE DATE
                                  is greater than January 1, 1989)

  . . . WHERE HIREDATE LE HDATE.  (qualifies the C0 data records where HIRE DATE
                                  is less than or equal to January 1, 1989)

MOVE 2000 TO NUM.

  . . .WHERE EMPNUMBER LT NUM.    (qualifies the C0 data records where EMPLOYEE
                                  NUMBER value is less than 2000)
```

## Item-to-Item Comparisons

### Format

```
item-alias * binary-operator item-alias *
```

In this situation, the operands of the binary-operator are the two items corresponding to the two given item-aliases. These two items must belong to the same data record or related data records. For each logical entry in the database, SYSTEM 2000 software compares the values of these two items. If the comparison is successful, the data record at the lower level containing one of these item values is qualified, regardless of the order in which the item aliases appear in the condition.

An item-to-item comparison is always processed as a non-key condition. The values to be compared must be compatible. See Chapter 4, "SYSTEM 2000 Item Types and COBOL Data Descriptions" for item types and COBOL data descriptors. For FORTRAN, PL/I, and Assembler data types, see the appropriate appendix.

For example, in the EMPLOYEE database, both of the following where-clauses qualify all C100 data records where the value of HIRE DATE is the same as the value of START DATE.

```
WHERE C4* EQ C105*
WHERE C105* EQ C4 *
```

The comparison between the two operands is based on the same rules as those when one operand is a variable. (Without the asterisks, the SAME syntax would be treated as a comparison of an item to a specified value.)

## The Ternary Operators EQ, NE, and SPANS

### Format

```
item-alias |EQ  variable * variable
           |NE
```

The operators EQ, NE, and SPANS compare a data value with a given range of values. They have three operands: an item, a low value in the first variable, and a high value in the second variable. The value in the first variable must be less than or equal to the value in the second variable.

EQ (and SPANS) qualify the data records that contain a value for the item that falls within the specified range. Both the low value and the high value are included in the range.

NE qualifies the data records containing a value for the item that falls outside the range specified. Both the low and high values are included in the range.

---

Abbreviation: EQ = SPANS

For comparisons of database items and program variables, the values being compared must be compatible and correctly specified to SYSTEM 2000 software. See Chapter 4, "SYSTEM 2000 Item Types and COBOL Data Descriptions" for a discussion of the compatibility of item types and COBOL data descriptions. See the appropriate appendix for FORTRAN, PL/I, and Assembler data types.

**Example**

```
DATA DIVISION.
      :
      :
COMMBLOCK OF EMPLOYEE.
      :
      :
SUBSCHEMA  ENTRY-REC   OF   EMPLOYEE.
01 ENTRY-REC.
   03 HIREDATE   PIC X(6).
   EQ HIRE DATE.
      :
      :
01 DATE1   PIC X(6).
01 DATE2   PIC X(6).
      :
      :
PROCEDURE DIVISION.
      :
      :
   MOVE '010188' TO DATE 1.          (is in the format MMDDYY, that is,
                                      month, day, year)
   MOVE '123190' TO DATE 2.
      :
      :
 . . . WHERE HIREDATE SPANS DATE1 * DATE2.   (qualifies all data records where
                                              the value of HIRE DATE is any date
                                              in the years 88, 89, or 90)
      :
      :
 . . . WHERE HIREDATE NE DATE1 * DATE2.      (qualifies all data records where
                                              the value of HIRE DATE is any date
                                              except in the years 88, 89, and 90)
      :
      :
 . . . WHERE HIREDATE EQ DATE1 * DATE2.      (same as HIREDATE SPANS
                                              DATE1 * DATE2)
```

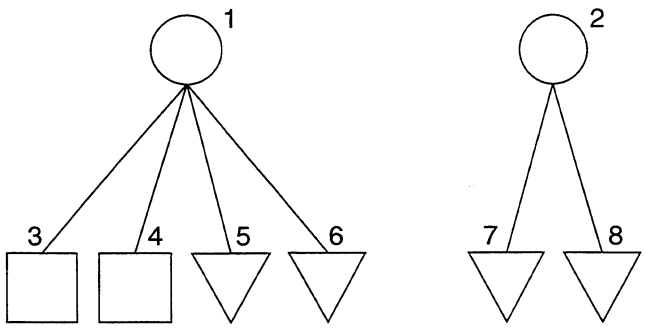
## The AND and OR Operators

### Format

```
expression   | , [AND]   expression
             | [,] AND
             | [,] OR
```

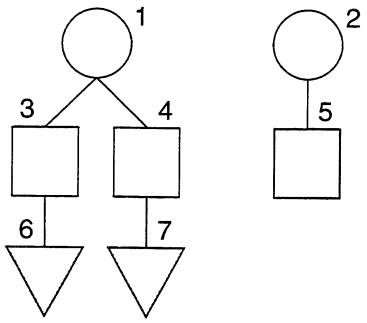
The logical operators AND and OR operate on two expressions, that is, two lists of qualified data records. The list produced from an AND operation contains data records that are qualified for both expressions. The list produced by an OR operation contains data records that are qualified for either one or both expressions.

For example, in the following illustration, suppose that data records 3, 7, and 8 qualify for *expression*1, and data records 3, 4, and 8 qualify for *expression*2. *Expression*1 AND *expression*2 produces a list consisting 3 and 8. *Expression*1 OR *expression*2 produces a list consisting of 3, 4, 7, and 8.




**Note**: AND and OR operations can be performed only on lists for the same schema record. If the lists are not for the same schema record, the software normalizes the list for a higher level schema record down to the level of the lower schema record. Then the logical operation is performed. The concept of normalization is explained in **Normalization** on page 38 with illustrations. Normalization can take place only from one schema record to another related schema record. Therefore, the logical operations AND and OR cannot be performed between lists of unrelated schema records. See **The HAS Operator** on page 7-19 for a way to combine lists of unrelated schema records using the HAS operator.

Since the normalization is always downward for AND and OR operators, the schema record of the resulting list is the same as the schema record of the list at the lower level. If an incomplete tree is encountered during the downward normalization, the data record at the lowest possible level is kept in the list. In the following trees, if normalization takes place from the circles to the triangles, the resulting list would contain the data records 5, 6, and 7.



### Examples

The following examples use the EMPLOYEE database definition. Assume that the subschema records and the variables are properly specified. The item aliases are the same as the corresponding item names except that they have hyphens instead of blanks. C-numbers are used as aliases in some places. The subschema record names are the same as the corresponding schema record names or C-numbers.

**Example 1**

```
   .
   .
   .
MOVE 'CAUCASIAN' TO ETHNIC.
MOVE 'MANAGER' TO TITLE.

LOCATE ENTRY-REC WHERE POSITION-TITLE    (locates all non-white managers)

 EQ TITLE AND ETHNIC-ORIGIN NE ETHNIC.
      .
      .
      .
```

**Example 2**

```
   .
   .
   .
MOVE 'FORTRAN' TO LANG1.
MOVE 'COBOL' TO LANG2.

LOCATE ENTRY-REC WHERE SKILL-TYPE EQ     (locates all employees with
 LANG1 OR SKILL-TYPE EQ LANG2.           programming knowledge of
                                         FORTRAN or COBOL or both)
```

**Example 3**

```
   .
   .
   .
MOVE 'MATH' TO SUB.
MOVE 'FULL TIME' TO STAT.

LOCATE ENTRY-REC WHERE (MAJOR-FIELD      (locates all full-time employees who
 EQ SUB OR MINOR-FIELD EQ SUB)           studied math either as a major or as
 AND EMPLOYEE-STATUS EQ STAT.            a minor. Note that although AND
                                         normally takes precedence over OR,
                                         the parentheses overrule this
                                         precedence.)
```

## The NOT Operator

### Format

```
NOT expression
```

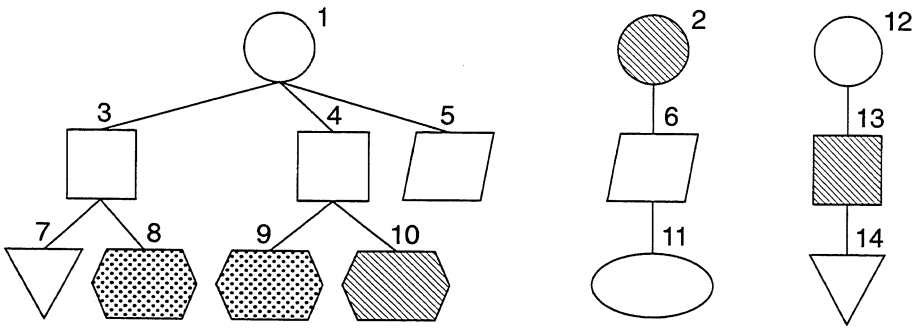
The NOT operator operates on a single operand, which is a list of data records. It replaces this list by a new list that contains

- data records that are of the same schema record as the original list but not on the list.
- data records corresponding to the nearest ancestor schema record that has corresponding data records if the schema record of the list has no corresponding data records in a logical entry. This is illustrated in Illus. 7.3.

In this example, if an *expression* produces the list of data record addresses 8 and 9, the NOT *expression* produces the list consisting of 2, 10, and 13. Data record 10 is of the same schema record as the original list and is not in the list. Data records 2 and 13 are qualified because they have no descendant data records corresponding to the schema record of the list and are the nearest ancestors. The schema record of the new list is the same as that of the old list. However, the list contains data records from other schema records.

See **Constructing a Where-clause** on page 7-30 for the usage of NOT combined with AND.

**Illus. 7.3** The NOT Operation



## The HAS Operator

### Format

*subschema-record* **HAS** *expression*

The HAS operator requires a subschema record name on the left and an expression on the right. The expression on the right produces a list of qualified data records. The subschema record name on the left represents a schema record, that is, the schema record corresponding to the subschema record. This schema record must be either an ancestor or a descendant of the schema records corresponding to data records in the list. HAS requests that the list of data records be replaced by another list consisting of those data records' ancestors or descendants that correspond to the specified schema record. In other words, HAS normalizes the list of data records on the right to the level of the schema record on the left.

You can use the HAS operator in the following situations:

- to combine two conditions specifying items in unrelated schema records
- to combine two mutually exclusive conditions
- to specify the logical placement of a record, in combination with the AT operator.

**Examples**

The following examples illustrate the purposes listed earlier, as well as other aspects of using the HAS operator.

### Example 1

The AND and OR operators can combine conditions that specify items in related schema records only. Assume that the subschema record name is the same as the corresponding schema record name and that the item aliases are the same as the corresponding item names with blanks replaced by hyphens.

```
DATA DIVISION.
      .
      .
      .
01 COBOL-PROG    PIC X(12).
01 COMPANY       PIC X(12).
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
  MOVE 'COBOL' TO COBOL-PROG.
  MOVE 'ABC' TO COMPANY.
      .
      .
      .
  LOCATE ENTRY-REC WHERE SKILL-TYPE
   EQ COBOL-PROG AND TRAIN-SOURCE
   EQ COMPANY.
```

*(produces return code 58 since the two conditions joined by AND qualify data records for the schema records JOB SKILLS and TRAINING, and these two schema records are not related)*

```
      .
      .
      .
  LOCATE ENTRY-REC WHERE ENTRY-REC
   HAS SKILL-TYPE EQ COBOL-PROG AND
   ENTRY-REC HAS TRAIN-SOURCE EQ
   COMPANY.
```

*(locates the ENTRY data records for employees who know how to program in COBOL and were trained by ABC Company)*

### Example 2

You can use HAS to combine two mutually exclusive conditions. In this example, HAS qualifies level 0 records of logical entries in which the mutually exclusive conditions are present.

```
DATA DIVISION.
      .
      .
      .
01 COBOL-PROG    PIC X(12)  VALUE 'COBOL'.
01 FORT          PIC X(12)  VALUE'FORTRAN'.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
 LOCATE ENTRY-REC WHERE SKILL-TYPE
  EQ COBOL-PROG AND SKILL-TYPE
  EQ FORT.
      .
      .
      .
      .
      .
      .
 LOCATE ENTRY-REC WHERE ENTRY-REC
  HAS SKILL-TYPE EQ COBOL-PROG AND
  ENTRY-REC HAS SKILL-TYPE EQ FORT.
```

*(produces 0 selected data records, since the intersection of the lists produced by the two conditions is null)*

*(locates the ENTRY-REC data records for the employees who know both COBOL and FORTRAN)*

You can also accomplish the same effect without using HAS. The advantages or disadvantages of using or not using HAS in this situation is discussed in **Efficiency Considerations in PLEX** on page 22-2.

### Example 3

When the database is maintained so that the most recent occurrence of some data record is first or last under its parent, you can use a HAS operator combined with an AT operator as shown in this example. (See **The AT Operator** on page 7-23 for information about the AT operator.)

Consider a logical entry in the EMPLOYEE database in which the most recent EDUCATION data record is leftmost under its parent. To obtain this most recent record (the highest degree) for the employee selected by EMPLOYEE NUMBER, you could use the following statements:

```
MOVE 1 TO ONE.
LOCATE EDUCATION
 WHERE (EDUCATION HAS EMPLOYEE-NUMBER EQ variable) AT ONE.
```

Notice the use of parentheses to force the HAS processing to precede the AT processing.

The expression using HAS would be processed a little faster than the following expression that uses an AND:

```
LOCATE EDUCATION
 WHERE EMPLOYEE-NUMBER EQ variable AND C412 EXISTS AT ONE.
```

### Example 4

The HAS operator is unnecessary and redundant in the command

```
LOCATE ENTRY-REC WHERE ENTRY-REC HAS SKILL-TYPE EQ COBOL-PROG.
```

The following command produces the same result:

```
LOCATE ENTRY-REC WHERE SKILL-TYPE EQ COBOL-PROG.
```

## The AT Operator

### Format

*expression* AT *n*

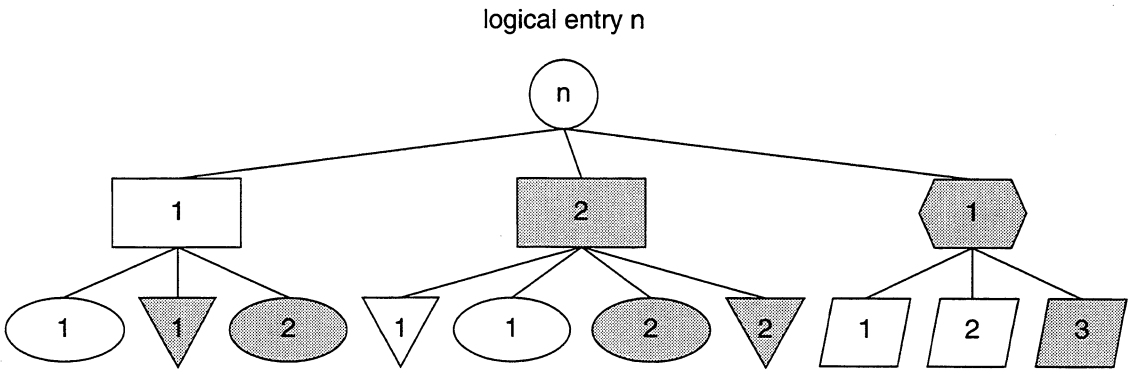
The AT operator enables you to choose a data record at a certain position under its parent, relative to its similar siblings. If the database is maintained in a certain order, the AT operator becomes useful.

**Relative position under a parent** Counting from left to right, if a data record is the *nth* one of its kind among its siblings, its position under its parent relative to its similar siblings is *n*. If a logical entry is in the *nth* position, the position of the data record at level 0 is *n*.

Also, a data record that is the last one among similar siblings is said to be in position zero. Therefore, a data record can be said to have two positions if it is the last data record of that schema record under its parent. Of these two positions, one is zero and the other is a non-zero positive integer. No data record can have two non-zero positions or more than two positions.

Illus. 7.4 shows a logical entry with the relative positions of all the data records. The data records at position zero are shaded.

**Illus. 7.4** Relative Position Under a Parent



**AT processing** The AT operator requires an expression to its left and a variable with a zero or positive integer value to its right. AT operates on the list produced by the expression on the left. You supply a zero or a positive integer in the variable to the right of AT. For each data record in the list, AT does the following processing:

- If the data record is of the same schema record as the list, it tests whether the data record's relative position among the similar siblings is the same as the integer value supplied for the AT. If the test is successful, the data record is placed in the resulting list; otherwise, it is discarded.

- If the data record is not of the same schema record as the list, it is discarded.

Thus, the resulting list contains data records that are all from the same schema record and that all have the same relative position under their parents.

**Use of AT** You can use AT to obtain data records at a specific relative position. This is useful when the database is maintained in such a way that the most recent data record is the first or the last among its similar siblings. This kind of logical placement is possible either by always inserting new data records at the first position or by always inserting them at the last position. (Similar techniques are also available in the Self-Contained Facility.)

## Examples

In the EMPLOYEE database, the data records for the schema records C100, C110, and C120 are maintained in the most-recent-first order; whereas the data records for schema records C410 and C420 are kept in the least-recent-first order.

### Example 1

The following code locates all the entries for the current managers if the subschema records and variables involved are properly described in the program:

```
DATA DIVISION.
      :
      :
01 REC-POSITION PIC 99 USAGE COMP-1.
      :
      :
PROCEDURE DIVISION.
      :
      :
 MOVE 1 TO REC-POSITION.
 MOVE 'MANAGER' TO TITLE.
 LOCATE ENTRY-REC WHERE C101 EQ TITLE AT REC-POSITION.
      :
      :
```

### Example 2

The following code locates all employees who were hired as managers:

```
      :
      :
MOVE 0 TO LAST-POSITION.
MOVE 'MANAGER' TO TITLE.
LOCATE ENTRY-REC WHERE C101 EQ TITLE AT LAST-POSITION.
      :
      :
```

### Example 3

The following code locates all current managers with a current monthly salary greater than or equal to $3200:

```
      :
      :
MOVE 3200 TO SALARY.
MOVE 1 TO REC-POSITION.
MOVE 'MANAGER' TO TITLE.
LOCATE ENTRY-REC WHERE C111 GE SALARY
 AT REC-POSITION, AND C101 EQ TITLE AT REC-POSITION.
      :
      :
```

### Example 4

This example locates all the current positions in which a salary increase did not occur at that position. The LOCATE command specifies in parentheses that the item C114 must belong to a record that is the only child of the first C100 record in the logical entry. The command does so by indicating that the record containing C114 must be both the first (REC-POSITION = 1) and the last (LAST-POSITION = 0) child. Furthermore, the C100 record with such a child must be the first (REC-POSITION = 1) C100 record in logical order in the entry, which (in the EMPLOYEE database) means the current position.

```
      :
      :
MOVE 1 TO REC-POSITION.
MOVE 0 TO LAST-POSITION.
LOCATE C100 WHERE C100 HAS (C114 EXISTS AT REC-POSITION
 AT LAST-POSITION) AT REC-POSITION.
      :
      :
```

**Example 5**

AT can also be used to determine whether a parent schema record has at least *n* occurrences of a particular child schema record. The following code can be used to find the employees who have at least two degrees/certificates:

```
    .
    .
    .
MOVE 2 TO REC-POSITION.
LOCATE ENTRY-REC WHERE C412 EXISTS AT REC-POSITION.
       .
       .
       .
```

**Considerations**

1. Suppose a where-clause includes the following AT construction:

   . . . *expression* **AT VAR1 AT VAR2** . . .

   In this situation, the AT operators have meaning only if one variable equals zero. Otherwise, if both variables have values greater than zero, the where-clause requests records that simultaneously occupy two different positions under their parents; the result is no retrieval. However, if one variable equals zero while the other equals a positive integer, the expression requests records that are at a certain position under their parents and that are last under their parents. This time, the where-clause can result in a retrieval, since a record can, for example, be both third and last one under its parent.

   When you specify two AT operators, processing is generally more efficient if VAR2 is equal to zero and VAR1 is equal to the positive integer.

2. If the variable to the right of the AT operator does not equal zero or a positive integer, return code 61 results.

3. Because of the way the pointers are maintained in the Hierarchical Table, processing for AT 1 is much faster than the processing for AT 0 or AT *a-large-number*, especially when there is a long sibling chain. However, at level 0, AT 0 can be processed as fast as AT 1.

4. A PLEX program can produce the same result by using GETD commands and program logic as it can by using AT in a where-clause. Both of these methods are worth considering if performance is a significant factor.

## Use of SAME LOCATE, SAME STACK and SAME POSITION

### Format

```
SAME   |LOCATE [Locate-File-number]
       |STACK  [stack-number]
```

*Locate-File-number* and *stack-number* are variables. For COBOL, use 01-level COMPUTATIONAL.

SAME reuses a list of data records produced in a previous command as a part of a new where-clause, without your having to build the list again. The processing of SAME is fast because the SAME condition's result is already present in either a Locate File or in a stack.

The keyword that follows the keyword SAME specifies whether the condition's result is in a Locate File or in a stack. If this keyword is LOCATE, the result is all the addresses in an existing Locate File. If this keyword is STACK, the result is a single data record address, that is, the primary entry of a stack. The number following this keyword determines which Locate File or stack contains the condition's result.

### Examples

The following examples show the use of SAME LOCATE and SAME STACK for including an existing Locate File or stack as a where-clause expression.

**Example 1**

```
DATA DIVISION.
      :
      :
01   LOCVAR     PIC S9(9)   COMP.        (variable must be fullword, COMP for
                                          COBOL)

PROCEDURE DIVISION.
      :
      :
 START S2K (6).
      :
      :
 LOCATE(1) MACHINE WHERE . . .            (establishes Locate File (1))
      :
      :
 MOVE 1 TO LOCVAR.
 LOCATE(5) EQUIPMENT WHERE
  SAME LOCATE (LOCVAR) AND . . .          (uses the COBOL 01-level COMP
                                          variable LOCVAR to indicate that
                                          Locate File 1 serves as an expression
                                          in the where-clause)
```

Synonyms:
LOCATE = LOC
STACK = POSITION, POS

**Example 2**

```
DATA DIVISION.
      .
      .
      .
01    STACKVAR      PIC S9(9)    COMP.      (variable must be fullword, COMP for
                                              COBOL)
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
  GET (10) OFFICE NEXT.                     (retrieves an office data record and
                                              establishes a primary stack entry)
      .
      .
      .
 MOVE 10 TO STACKVAR.
 GET1 (4) BUILDING WHERE
  SAME STACK (STACKVAR) AND. . .            (uses the COBOL 01-level COMP
                                              variable STACKVAR to indicate that
                                              the primary entry in stack 10 serves
                                              as an expression in the where-clause)
```

**Considerations**

1. The where-clause syntax shows that an expression resulting from SAME can be reused with any of the operators NOT, AND, OR, HAS, or AT with restrictions. The restrictions are discussed here.

   a) A SAME expression cannot be joined by an OR with an expression that requires a complete pass of the Data Table. (See **Constructing a Where-clause** on page 7-30.) For example, the following SAME expression is not allowed:

      **(SAME LOCATE (LOCVAR) OR** *non-key-condition*) **AND** *key-condition*

      However, the next SAME expression is allowed because the AND operation is performed first and a complete pass of the Data Table is not required.

      **SAME LOCATE (LOCVAR) OR** *non-key-condition* **AND** *key-condition*

   b) The NOT operation cannot be performed for a SAME expression. For example, the following expression is not acceptable:

      **NOT SAME LOCATE (LOCVAR)**

      However, the next expression is acceptable if the expression does not require a full pass of the Data Table. See **Constructing a Where-clause** on page 7-30 for AND NOT processing.

      *expression* **AND NOT SAME LOCATE (LOCVAR)**

      In general, SAME cannot be processed in those situations where, if SAME were replaced by any other key or non-key condition, a full pass of the Data Table would be required.

2. If a SAME expression is in error and a non-zero return code results, the left-to-right ordinal position of the expression in the where-clause is indicated in the eighth field of the COMMBLOCK.

3. If the Locate File or stack specified in a where-clause expression contains data record addresses for a database other than the one for the subschema record specified to the left of WHERE, the command is ignored, and return code 12 results.

4. If the Locate File or stack specified in a where-clause expression was not used previously by the program, the command is ignored, and return code 12 results.

5. If the variable for the specified Locate File or stack number is a negative value, the command is ignored, and return code 40 results.

6. If the value for the specified Locate File exceeds the subscript in the previous START S2K command, the command containing the where-clause is ignored, and return code 40 results.

7. If the value for the specified stack exceeds 15, the command is ignored, and return code 40 results.

8. When used with SAME STACK or SAME LOCATE, *Locate-File-number* and *stack-number* must be program variables. If they are literals rather than variables, an error results during compilation. For COBOL, the variable must be fullword 01-level COMPUTATIONAL. For PL/I, it must be a FIXED BINARY(31) variable.

## Constructing a Where-clause

This section gives some examples and guidelines for constructing an efficient where-clause.

**Precedence of operators** Understanding the precedence of operators is important in constructing a where-clause that does what it is intended to do. As mentioned before, when there are no parentheses to alter the precedence, AT is processed first, then HAS, followed by NOT, AND, and OR, in that order.

You can further alter the processing order by using parentheses.

### Example 1

This example illustrates the order of processing a condition preceded by a HAS operator and followed by an AT operator.

```
C0 HAS C21 EXISTS AT N
```

Step 1: evaluate C21 EXISTS
Step 2: the results of Step 1 **AT** N
Step 3: C0 **HAS** the results of Step 2

### Example 2

The following expressions illustrate the precedence of the HAS, AND, and OR operators, as well as the placement of parentheses. Conditions are represented by the letters A, B, and C.

*subschema record* **HAS A AND B OR C**

Step 1: subschema record **HAS** A
Step 2: the results of Step 1 **AND** B
Step 3: the results of Step 2 **OR** C

*subschema record* **HAS A AND (B OR C)**

Step 1: subschema record **HAS** A
Step 2: B **OR** C
Step 3: the results of Step 1 **AND** the results of Step 2

*subschema record* **HAS (A AND B OR C)**

Step 1: A **AND** B
Step 2: the results of Step 1 **OR** C
Step 3: subschema record **HAS** the results of Step 2

### Example 3

These examples illustrate the precedence of the HAS, AND, and AT operators, as well as the placement of parentheses.

*subschema record* **HAS A AND B AT N**

Step 1: subschema record **HAS** A
Step 2: B **AT** N
Step 3: the results of Step 1 **AND** the results of Step 2

*subschema record* **HAS (A AND B) AT N**

Step 1: A **AND** B
Step 2: the results of Step 1 **AT** N
Step 3: subschema record **HAS** the results of Step 2

**(***subschema record* **HAS A AND B) AT N**

Step 1: subschema record **HAS** A
Step 2: the results of Step 1 **AND** B
Step 3: the results of Step 2 **AT** N

**A AND** *subschema record* **HAS B AT N**

Step 1: B **AT** N
Step 2: subschema record **HAS** the results of Step 1
Step 3: A **AND** the results of Step 2

### Example 4

These examples illustrate the precedence of the AND, NOT, and OR operators, as well as the placement of parentheses. Assume that all components are key.

**NOT A AND B OR C**

Step 1: A
Step 2: B **AND NOT** results of Step 1
Step 3: the results of Step 2 **OR** C

**Note**: In Step 2, AND NOT is processed as a single operator. B AND NOT A is processed by forming a new list of addresses that appear in the list for B but not for A.

**NOT (A AND B) AND C**

Step 1: A **AND** B
Step 2: C **AND NOT** results of Step 1

**Example 5**

The following example shows the processing of an expression with nested parentheses:

```
(A HAS (B AND (C OR D)))
```

Step 1: C **OR** D
Step 2: B **AND** results of Step 1
Step 3: A **HAS** results of Step 2

**The processing of logical operators** When more than two key conditions are joined by the same operator, they are processed from right to left, as shown below.

```
A AND B AND C AND D
```

Step 1: C **AND** D
Step 2: B **AND** results of Step 1
Step 3: A **AND** results of Step 2

An AND operation on shorter lists takes less time than on longer lists. The resulting list is always shorter than or equal to the operand lists. Therefore, if the relative sizes of the lists are predictable, it is more efficient to put the conditions producing longer lists on the left side and those producing shorter lists on the right side of the expression.

When three or more key conditions joined by AND or OR are being processed from different levels of the data trees, normalization is necessary. Normalization to the next higher or lower level is generally less time consuming than normalization up or down several levels. Therefore, less normalization is required if the conditions are ordered right-to-left by descending level, that is, if the condition closest to level 0 is written last.

**Key vs. non-key** In an expression with connected key conditions, if you think that one condition would produce a very long list of qualified data records compared with the others, declare that condition as NON-KEY. NON-KEY causes the software to process a key condition as a non-key condition.

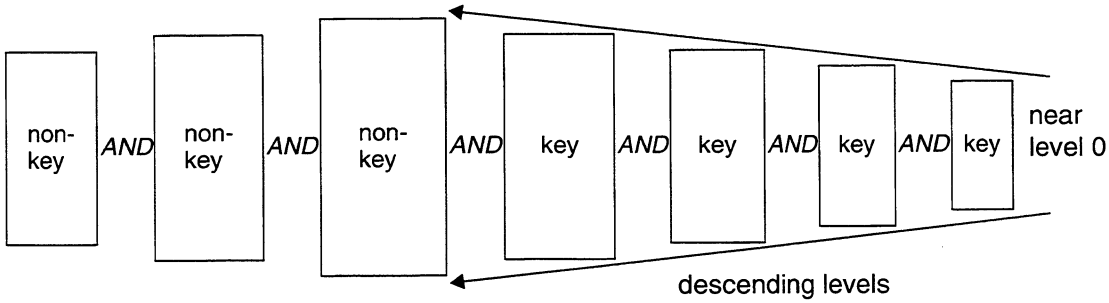
Although processing a non-key condition alone is expensive, an expression of the form A AND B where either A or B is a non-key condition can be cheaper than the same expression where both A and B are key conditions. Suppose A qualifies 400,000 data records and B qualifies only 25 data records. If both A and B are key conditions, much time would be spent in creating the long list of qualified data records for A. However, if A is a non-key condition, the list for B would be created first, so only those 25 data records would be examined in the Data Table for condition A.

**Placement of conditions** Unlike key conditions, non-key conditions are processed from left to right. Therefore, for higher efficiency when conditions are joined by AND, place the conditions that are likely to qualify small numbers of data records on the left of those that are likely to qualify large numbers of data records. Since there is no normalization, the order of the levels is arbitrary.

Illus. 7.5 shows the ideal way to place conditions joined by AND operators. A box labeled key represents a key condition, and the boxes labeled non-key represent non-key conditions. The size of the box is proportional to the number of data records qualified by the condition.

**Note**: Sometimes it might be impossible to arrange the conditions in this way. Also, it might not be worthwhile unless performance is a consideration.

**Illus. 7.5** Placing of Conditions in a Where-clause



**Preventing full passes of the Data Table** A full pass of the Data Table is necessary when a fixed (or DYNAMIC) where-clause consists of a non-key expression. A non-key expression is formed by any of the following combinations of operators, conditions, or expressions:

| | |
|---|---|
| **NOT** *key* | **NOT** *key* **OR NOT** *key* |
| *non-key* **AND** *non-key* | **NOT** *key* **OR** *non-key* |
| **NOT** *key* **AND** *non-key* | **NOT** *key* **OR** *key* |
| **NOT** *key* **AND NOT** *key* | *key* **OR** *non-key* |
| *non-key* **OR** *non-key* | |

For example, suppose the where-clause consists of the following conditions:

*non-key* **OR** *key* **AND** *non-key*

This combination of conditions constitutes a key expression.

However, the entire combination of *non-key condition* and the key expression (*non-key* AND *key*) constitutes a non-key expression.

Since a full pass of the Data Table can be very expensive in a large database, the master password holder can prohibit full passes by issuing the NO FULL PASSES command. (See Chapter 14, "FULL PASSES.") If full passes are prohibited, the software does not process a where-clause that needs a full pass, and return code 62 is issued.

# DYNAMIC WHERE-CLAUSES

A DYNAMIC where-clause is a QUEST where-clause that can be created by the PLEX program while it is executing or read in from a terminal or an external file. You place the QUEST where-clause in a program variable at run time, and it is passed directly to SYSTEM 2000 software.

You can use a DYNAMIC where-clause in a GET1 or LOCATE command. Both fixed and DYNAMIC where-clauses can be used in the same PLEX program. (You cannot use a DYNAMIC where-clause with the GETP command, which has a single key condition.)

Here are some advantages of DYNAMIC where-clauses.

- You can generate or read in any QUEST where-clause, which means you can use the familiar SCF syntax in PLEX programs. Therefore, the CONTAINS operator, for example, is available to search for character patterns, prefixes, suffixes, and so on.
- You can alter selection criteria dynamically. You do not need to modify or recompile your PLEX program every time you want to change a where-clause or use an item that is not declared in a PLEX subschema.
- You can decrease memory requirements by not having to code fixed where-clauses to cover all situations. Simply read in the selection criteria at run time.
- You do not have to worry about program aliases or inconsistent PLEX subschema item names. Simply give C-numbers and component names as they appear in the database definition.

To use a DYNAMIC where-clause, declare a program variable to hold the where-clause. Then specify that variable name in a GET1 or LOCATE command.

## Format

---

**GET1 command [/HOLD] [(*m*)]** *SSR* **[***inclusion-list***] DYNAMICALLY [WITH]** *name*.

---

**LOCATE [(*m*)]** *SSR* **DYNAMICALLY [WITH]** *name*.

---

| | |
|---|---|
| *m* | is an optional stack number for GET1 or an optional Locate File number for LOCATE. |
| *SSR* | is the name of the subschema record that corresponds to a data record to be retrieved. |
| *inclusion-list* | is an optional item list, name table, or word list, which is available for any GET1. |

---

Abbreviation: DYNAMICALLY = DYNAM

*name* is the name of the program variable that will contain the QUEST where-clause to be processed. The variable must begin on a word boundary and cannot exceed 1000 characters. It can be subdefined or redefined as allowed by the language being used.

For COBOL, the variable must be an 01-level variable with USAGE DISPLAY and a PICTURE of X in the WORKING STORAGE or LINKAGE SECTION of the program.

For FORTRAN, the variable must be INTEGER or CHARACTER in the declarative section of the program. It cannot be a substring.

For PL/I, the variable must be a DECLARED variable or a DECLARED constant.

## Examples

These three examples show COBOL, FORTRAN, and PL/I code that constructs DYNAMIC where-clauses in different situations.

### Example 1

The COBOL code below illustrates prompting for a QUEST where-clause, using it in a GET1 statement for subschema record SSRA, and testing the return codes.

```
01  MY-WHERE-CLAUSE PIC X(80)
      .
      .
      .
TRY-AGAIN
   DISPLAY "ENTER WHERE-CLAUSE."
   ACCEPT MY-WHERE-CLAUSE.
   GET1 SSRA DYNAMICALLY MY-WHERE-CLAUSE.
   IF RC = 124
      PERFORM SYNTAX-ERROR
   ELSE
      IF RC = 4
         PERFORM NO-SELECTED-DATA-SETS
   ELSE
      IF RC NOT = 0
         PERFORM BAD-PLEX-RC
   ELSE
      PERFORM EXECUTE-TRANSACTION.
   GO TO TRY-AGAIN.
```

### Example 2

The FORTRAN code below illustrates reading in 80-character lines containing a QUEST where-clause, using the where-clause to locate level 0 records (subschema ENTRY), and testing the return codes.

```
      CHARACTER WHBUF*240
         .
         .
      DO 10 I = 1, 3
      KFIRST = (I-1) *80
   10 READ 100, WHBUF (KFIRST: KFIRST + 79)
  100 FORMAT (A80)
      LOCATE ENTRY DYNAMICALLY WITH WHBUF.
      IF (RC .EQ. 124) GO TO 9000
      IF (RC .NE. 0) GO TO 9100
         .
         .
 9000 CONTINUE
      PRINT 9001, RC, FCNRG
 9001 FORMAT (1X, "DYNAMIC WHERE SYNTAX ERROR,"
     -A5, "MESSAGE NUMBER," I4)
```

### Example 3

The following PL/I code declares two DYNAMIC where-clauses. The variable names are DYWHC1 and DYWHC2, which are used in the GET1 and LOCATE statements. Notice that item C404 is used in the DYNAMIC where-clause even though it was not declared as a subschema item in the program.

```
   DCL 1 DYWHC1 CHAR(12) INIT('C301 EXISTS:');
   DCL 1 DYWHC2 CHAR(12) INIT('C404 EXISTS:');
      .
      .
SUBSCHEMA COPY OF LIBRARY;
   DCL 1 COPY,
      3 C401 PIC '9999',
      .
      .
SUBSCHEMA AUTHOR OF LIBRARY;
   DCL 1 AUTHOR,
      3 C301 CHAR(20),
      .
      .
GET1 COPY DYNAM DYWHC2;
      .
      .
LOCATE AUTHOR DYNAM DYWHC1;
IF RETURN_CODE = 124 THEN GO TO PRNT_SCFE;
IF RETURN_CODE = 0 THEN GO TO PRNT_DWCR;
RC_DISP = RETURN_CODE;
PUT EDIT (MSG2,RC_DISP) ...
      .
      .
PRNT_SCFE:
   RC_DISP = RETURN_CODE;
   SCF_RTRNCD = #_OF_DATA_SETS;
   PUT EDIT (SCFMSG,RC_DISP,SCF_RTRNCD ...
```

## Considerations

You can move any QUEST where-clause into the program variable declared for the DYNAMIC where-clause. You might want more than one variable for this purpose. Considerations regarding the contents of a DYNAMIC where-clause follow:

1. The where-clause must not begin with the keyword WHERE.

2. The where-clause must end with the SCF command terminator. Optionally, you can include SCF comments after the command terminator. Comments supply additional documentation, mostly in batch environments.

3. You can use logical symbols for the operators in the QUEST where-clause.

4. Items and records in the selection criteria must be specified with the component names or C-numbers contained in the database definition. They do not have to be declared in a PLEX subschema record. The QUEST where-clause is sent directly to SYSTEM 2000 software at run time. It is not parsed by the PLEX processor or compiled into the program.

5. The SAME operator and Collect File items are invalid for DYNAMIC where-clauses.

6. Subscripts are not allowed on the variable names, schema items, or records.

7. The contents of the variable must include the entire where-clause. Multiple calls are not allowed.

8. Password security is honored. The current password in the PLEX program must have W-authority for the components in the QUEST where-clause.

9. You can use both fixed and DYNAMIC where-clauses in the same program.

10. E-notation is valid for all numeric values in a DYNAMIC where-clause. When the item in a DYNAMIC where-clause condition is a fixed format DECIMAL, MONEY, or INTEGER item, the result of the conversion from E-format to fixed format must conform to the item's picture.

For information about the QUEST where-clause, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition*.

## How To Code a DYNAMIC Where-Clause

The source code required for the DYNAMIC where-clause is minimal. The PLEX processor takes care of the internal details. To include a DYNAMIC where-clause in your PLEX program, follow these steps:

1. Declare a DYNAMIC where-clause variable that will be used to hold the QUEST where-clause.

2. Code the statements that move the QUEST where-clause into the program variable declared in Step 1. The value for this variable can be read in directly from a user terminal or an external file, or it can be created by the PLEX program. Program logic can also provide a friendlier mechanism for the user by accepting various user responses and transforming them into acceptable QUEST where-clauses. **Note**: The system will accept hexadecimal input in a DYNAMIC where-clause.

3. Code the GET1 or LOCATE statements in the program source. Use the DYNAMICALLY WITH syntax to specify the variable name.

4. Include a test of the PLEX return code (second COMMBLOCK variable) to see if the where-clause worked successfully. Check for return code 124 to see if a syntax error occurred in the where-clause; if so, no further processing takes place. The tenth COMMBLOCK variable contains the SCF message number for the error. You can display that number to the user in some meaningful format.

5. After checking for return code 124 (Step 4), check the second COMMBLOCK variable (return code) again for the usual GET1 and LOCATE execution errors, such as improper positioning or invalid subschema record declarations.

For information about the QUEST where-clause, see *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition*. For information about messages and return codes, see *SYSTEM 2000 Messages and Codes, Version 12, First Edition*, and subsequent Changes and Enhancements technical reports.

## Opcodes for DYNAMIC Where-Clause

Here are the two operation codes for PLEX commands that contain a DYNAMIC where-clause.

| PLEX Command | Dec Opcode | Hex Opcode |
|---|---|---|
| GET1 *SSR* DYNAMICALLY | 07 | 07 |
| LOCATE *SSR* DYNAMICALLY | 12 | 0C |

## Error Handling

There are no PLEX processor error messages for the DYNAMIC where-clause feature. If you do not declare a program variable for the DYNAMIC where-clause, a COBOL, FORTRAN, or PL/I compiler error will occur.

During program execution, two types of errors could occur from DYNAMIC where-clause processing.

- Syntax errors in the QUEST where-clause that was sent to SYSTEM 2000 software. If a syntax error occurs, the following two variables are set in the COMMBLOCK:

  The second COMMBLOCK variable (RETURN CODE) is set to 124. The tenth COMMBLOCK variable (CYCLE DATE) contains the SCF error message number corresponding to the error encountered. The message number represents the same error that would result from an SCF session. (Also, remember that the SAME operator and Collect File items are not allowed in a DYNAMIC where-clause.)

  **-368- SAME NOT VALID WITH DYNAMIC WHERE -**
  is an SCF message, indicating the SAME operator is not supported in a PLEX DYNAMIC where-clause.

  If a syntax error occurs, no changes are made to the Locate Files, stacks, SSR information, and so on. They will contain the same information that they did before the error occurred. No real processing was done because the syntax error prevented the software from performing any further action.

- If the DYNAMIC where-clause processes successfully, the rest of the GET1 or LOCATE command is processed. Any errors at this point will be represented by non-zero return codes in the usual manner.

For descriptions of all SYSTEM 2000 messages and codes, see *SYSTEM 2000 Messages and Codes, Version 12, First Edition*.

Chapter 8

# Retrievals


*GET1 8-5*
- *Format 8-6*
- *Examples of Syntax 8-7*
- *Example of Usage 8-8*
- *Considerations 8-9*
- *GET1 NEXT 8-12*
- *GET1 FIRST 8-22*
- *GET1 LAST 8-26*
- *GET1 PRESENT 8-34*
- *GET1 S2KCOUNT 8-39*
- *GET1 with a Where-Clause 8-44*

*GETA 8-47*
- *Format 8-47*
- *Examples of Syntax 8-48*
- *Examples of Usage 8-49*
- *Considerations 8-50*

*GETD 8-54*
- *Format 8-55*
- *Examples of Syntax 8-56*
- *Example of Usage 8-57*
- *Considerations 8-58*
- *GETD NEXT 8-60*
- *GETD FIRST 8-64*
- *GETD LAST 8-67*
- *GETD PRESENT 8-70*
- *GETD S2KCOUNT 8-73*

*LOCATE 8-81*
- *Format 8-81*
- *Examples of Syntax 8-83*
- *Example of Usage 8-83*
- *Considerations 8-84*

*ORDER BY 8-92*
- *Format 8-92*
- *Examples of Syntax 8-93*
- *Example of Usage 8-94*
- *Considerations 8-95*

*GET 8-100*
- *Format 8-101*
- *Examples of Syntax 8-102*
- *Example of Usage 8-103*
- *Considerations 8-104*

*GET NEXT 8-107*
*GET FIRST 8-109*
*GET LAST 8-110*
*GET PREVIOUS 8-113*
*GET PRESENT 8-115*
*GET S2KCOUNT 8-117*

*INCLUSION LIST 8-121*
*Format 8-121*
*Item List 8-122*
*Name Table 8-125*
*Word Table 8-128*

*USING 0 OPTION 8-131*
*Format 8-131*
*Examples 8-131*

*HOLD OPTION 8-131*
*Format 8-132*
*Local Holds (/HOLD in Immediate Mode) 8-132*
*Global Holds (/HOLD Option in Queue Mode) 8-134*
*Considerations 8-137*
*Example of Usage in Immediate Mode 8-142*
*Example of Usage in Queue Mode 8-143*

*DROP HOLD 8-144*
*Format 8-144*
*Examples of Syntax 8-144*
*Example of Usage 8-145*
*Considerations 8-146*

*COPYS AND COPYL 8-147*
*Format 8-147*
*Examples of Syntax 8-148*
*Examples of Usage 8-149*
*Considerations 8-150*

*GETP 8-151*
*Format 8-153*
*Examples of Syntax 8-154*
*Example of Usage 8-154*
*Considerations 8-155*

*THE LINK COMMAND 8-163*
*Format 8-164*
*Examples of Syntax 8-165*
*Considerations 8-165*

*LINK PROCESSING 8-166*
*Order of Retrievals in a Cascade 8-167*
*LINK Return Codes and S2KLKH Table 8-169*
*Via-clause for LINK 8-169*

*Source and Target Control 8-170*
*Inclusion List for LINK 8-172*
*HOLD Option for LINK 8-172*

This chapter describes in detail the retrieval commands GET1, GETA, GETD, GETP, and GET. It also describes the syntax and semantics of the LOCATE and ORDER BY commands, which manipulate a Locate File, and the COPYS and COPYL commands, which copy stacks and Locate Files. You should have a general understanding of the concepts of the stack, the Locate File, and the where-clause before you read this chapter. If you have not already done so, read about these concepts in preceding chapters.

This chapter also describes how to obtain global and local holds for a database. A discussion of the commands for copying stacks and Locate Files completes the chapter.

The list on the next page provides a brief description of each command and option described in this chapter.

the commands

| | |
|---|---|
| **GET1** | retrieves a record according to logical order or with a where-clause; establishes a primary stack entry. Details in **GET1** on page 8-5. |
| **GETA** | retrieves an ancestor of a record whose address is a stack entry; establishes a secondary stack entry. Details in **GETA** on page 8-47. |
| **GETD** | retrieves a descendant of a record whose address is a stack entry; establishes a secondary stack entry. Details in **GETD** on page 8-54. |
| **GETP** | retrieves a record according to a single key condition where-clause; establishes a primary stack entry. Details in **GETP** on page 8-151. |
| **LOCATE** | uses a where-clause to establish a Locate File of addresses of selected records. Details in **LOCATE** on page 8-81. |
| **ORDER BY** | resequences the record addresses in a Locate File. Details in **ORDER BY** on page 8-92. |
| **GET** | retrieves a record whose address is in a Locate File; establishes a primary stack entry. Details in **GET** on page 8-100. |
| **DROP HOLD** | ends one or all local holds for a program. Details in **DROP HOLD** on page 8-144. |
| **COPYS** | duplicates a stack or cancels its entries. Details in **COPYS and COPYL** on page 8-147. |
| **COPYL** | duplicates a Locate File or cancels all addresses in it. Details in **COPYS and COPYL** on page 8-147. |

the options

| | |
|---|---|
| *inclusion-list* | indicates subschema items to be used in a retrieval or update operation. |
| **USING 0** | indicates that only the address of a record is to be retrieved, not the data in the record. Details in **USING 0 Option** on page 8-131. |
| **HOLD** | requests a local hold or a global hold. Details in **HOLD Option** on page 8-131. |

# GET1

The GET1 command retrieves one data record, either by following the order of data records in the logical data structure or by specifying conditions in a where-clause. The GET1 command has several options, for example, GET1 *SSR* NEXT, and GET 1 *SSR* LAST. Subsequent sections discuss each option. This section discusses the GET1 command generally and presents considerations for all GET1 commands.

The GET1 command has the following features:

- You can retrieve a data record from any point at any level in the database.
- You can retrieve data records in logical order and selectively through your choice of NEXT, FIRST, LAST, PRESENT, or your specified value of S2KCOUNT.
- Retrieval can be conditional if you use a where-clause.
- GET1 establishes the primary stack entry. You can specify any one of sixteen stacks.
- With the USING 0 option, you can retrieve a data record address without retrieving data values.
- With an inclusion list, you can retrieve values from specific (rather than all) items in a subschema record.
- You can specify the HOLD option in a Multi-User environment to protect the integrity of retrieved data.

You can code a GET1 command (with or without a where-clause) whenever necessary during retrieval or update operations. Unlike the GET command, GET1 is not based on a previous LOCATE command.

A GET1 command specifies the stack to be used and a subschema record associated with the data record to be retrieved. When GET1 is executed, data are retrieved and the address of the retrieved data record becomes the primary stack entry. Any previous entries in the stack are canceled.

If you specify a where-clause in a GET1 command, your program can retrieve a data record that meets specified conditions. If you specify FIRST with GET1, your program can select the first data record in logical order that is associated with the specified subschema record. If you specify LAST with GET1, your program can select the last such data record in logical order. If you repeatedly give a GET1 command specifying NEXT or S2KCOUNT, your program's retrieval operation can step through the database in logical order, selecting data records associated with the specified subschema record.

## Format

```
GET1 [/HOLD] [(stack-number)] SSR [inclusion-list] [NEXT]
                                  [USING 0]        [FIRST]
                                                   [LAST]
                                                   [PRESENT]
                                                   [S2KCOUNT]
                                                   [plex-where-clause]
```

*stack-number* is the number of the stack to be used in retrieval: a constant or a variable whose value is greater than or equal to zero and less than or equal to 15, that is, $0 \leq n \leq 15$. The default value for *stack-number* is 0. For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*SSR* is the name of the subschema record for the data record to be retrieved.

*inclusion-list*

```
|USING  |item-list
       |name-table
       |word-table
```

*item-list*

```
item-name [, item-list]
```

*item-name* is the alias of an item in the specified subschema record.

*name-table* is the name of a table of component names or C-numbers for items to be included in the retrieval operation. For COBOL, the entry with the OCCURS clause must specify PICTURE X(32).

*word-table* is the name of table of one-word indicators (1s and 0s), one indicator for each item in the specified subschema record. An item is included in the retrieval operation if its indicator in the table is 1. Conversely, an item is excluded from the operation if its indicator is zero. In each indicator, only the low-order bit is checked to see whether it is zero or 1. For COBOL, the entry with the OCCURS clause must specify COMP usage.

*plex-where-clause* states the conditions of retrieval, composed of the keyword WHERE and conditions or expressions. See **GET1 with a Where-Clause** on page 8-44 for details about a where-clause with GET1. See also Chapter 7, "Where-Clauses" for complete details on the where-clause.

short form of the command and its equivalents

```
GET1 SSR.                          (short form, using all defaults)
GET1 (0) SSR.
GET1 SSR NEXT.
GET1 (0) SSR NEXT.
```

## Examples of Syntax

```
GET1 (8) POSITIONS.                (specifies stack and subschema record;
                                   NEXT is implied)

GET1/HOLD (1) SKILLS PRESENT.      (specifies HOLD and PRESENT)

GET1 (9) PAYROLL NEXT.             (specifies NEXT explicitly)

GET1 (3) EDUC S2KCOUNT.            (specifies S2KCOUNT instead of NEXT;
                                   the program must supply a value for
                                   S2KCOUNT before using it in a command)

GET1 (0) POSITIONS TITLE, DIV LAST.  (specifies an item list and LAST)

GET1 (2) TRAINING USING 0.         (specifies address retrieval only)

GET1/HOLD (7) SALARY USING         (specifies the HOLD option and the
     POS-NAME-TABLE LAST.          name table called POS-NAME-TABLE)

GET1 (8) POSITIONS USING           (specifies the word table called
     POS-WORD-TABLE-LAST.          POS-WORD-TABLE)
```

## Example of Usage

```
WORKING-STORAGE SECTION.
      .
      .
      .
01  ENTRY-REC-NAME-TABLE.
    05  ENTRY-REC-ITEM OCCURS 4 TIMES PIC X(32).
01 NUM . . .
      .
      .
      .
COMMBLOCK OF EMPLOYEE.
      .
      .
      .
SSR EDUCATION OF EMPLOYEE.              (subschema record corresponds
                                        to the EDUCATION schema record)
      .
      .
      .
SSR ENTRY-REC OF EMPLOYEE.              (subschema record corresponds
                                        to the level 0 schema record)

01  ENTRY-REC.
    03  LASTNAME.
        EQ LAST NAME.
    03  FORENAME.
    03  EMPNO.
        EQ EMPLOYEE NUMBER.
    03  SSN.
        EQ SOCIAL SECURITY NUMBER.
      .
      .
      .
PROCEDURE DIVISION
      .
      .
      .
    MOVE 1 TO S2KCOUNT.                 (sets S2KCOUNT to 1 for the
                                        following GET1 command;
                                        variable was specified with
                                        PIC S9(9) COMP in the S2KDUM
                                        control block)

    GET1 EDUCATION S2KCOUNT.            (S2KCOUNT set to 1; command
                                        is the equivalent of GET1
                                        with NEXT)

    MOVE 'LAST NAME' TO ENTRY-REC-ITEM (1).
    MOVE 'FORENAME' TO ENTRY-REC-ITEM (2).
    MOVE 'SOCIAL SECURITY NUMBER' TO ENTRY-REC-ITEM (3).
    MOVE SPACES TO ENTRY-REC-ITEM (4).

      (Input value for NUM)             (obtains an employee number for
                                        retrieval and begins loop)

    GET1 ENTRY-REC USING ENTRY-REC-NAME-TABLE
    WHERE EMPNO EQ NUM.                 (retrieves the single level 0
                                        data record that contains an
                                        employee number equal to the
                                        input value of NUM and
                                        then repeats the loop)
```

## Considerations

1. Check the return code after each GET1 command. The return code is in the second COMMBLOCK field.

2. For a GET1 command specifying PRESENT, the stack must have an entry at or below the level of the specified subschema record. That entry must be for a data record in the path of the data record for the specified subschema record; otherwise, return code 03 results. The same result is obtained in this case if the GET1 command specifies S2KCOUNT, where S2KCOUNT equals 0.

3. A GET1 command with a where-clause retrieves only one data record that meets the conditions of that where-clause. However, the number of qualified records will be returned in the sixth field of the COMMBLOCK. If a GET1 command is repeated with the same subschema record and the same where-clause, the same data record will be retrieved, with return code 00. When you need to retrieve more than one data record based on the same where-clause, use the LOCATE command. Also, a GET1 command that specifies NEXT following GET1 with a where-clause retrieves the next data record associated with the subschema record, in logical order following the data record randomly selected through where-clause processing. If the where-clause in a GET1 command consists of a single key condition, you can use the no-sort capability. See Appendix A for a discussion of this capability.

4. If a GET1 PRESENT or a GET1 LAST command for a non-level-0 record causes retrieval of a data record whose address is already the primary stack entry, return code 50 results, signalling proper command execution but indicating a possible program loop.

5. A GET1 LAST command for a non-level-0 record results in return code 03 when the data record for the primary entry is beyond the last data record associated with the subschema record in logical order. The primary entry is beyond the last data record for the subschema record if either of the following is true:

   - the last data record for the subschema record is an ancestor of the data record for the primary entry.

   - the data record for the primary entry is in a path that follows the path of the last data record for the subschema record in logical order.

6. For a discussion of inclusion lists, see **Inclusion List** on page 8-121. The USING 0 option is discussed in **USING 0 Option** on page 8-131.

7. The program's current password must carry R-authority (retrieval) for each schema item or schema record referred to in a retrieval command. In addition, the password must have W-authority (where-clause) for each schema item or schema record specified in a where-clause. Otherwise, the GET1 command is ignored, and return code 124 or 46 results.

8. In a where-clause condition, the type and value of the program variable to the right of an operator must be compatible with the data type and picture of the subschema record item in the condition. If it is not, return code 124 (and SCF message -308-) results.

9. In a GET1 command with a where-clause, the specified subschema record must correspond to a schema record in the same path as indicated by the schema records and items referred to in the where-clause conditions. If the records and items referred to in the conditions do not fall in one path, or if the specified subschema record is not in the indicated path, the GET1 command is ignored, and return code 58 results.

10. A GET1 NEXT command yields four possible results:

    - If the specified stack has no primary entry, GET1 NEXT behaves like GET1 FIRST. The first record in logical order for the specified subschema record is retrieved.

    - If the schema record for the primary entry is in the same family as the schema record referred to in the command, retrieval movement through the data structure is based on the level of the primary entry and the level of the schema record to which the command refers.

    - If the schema record for the primary stack entry is not in the same family as the schema record referred to in the command, the GET1 command is ignored, and return code 03 results.

    - If a data record for the primary stack entry was deleted by a REMOVE TREE operation or moved by a MOVE TREE operation, a successful retrieval depends on whether the REMOVE TREE or MOVE TREE command was given for the same stack, for another stack, or by another user.

11. Retrieval with a GET1 command can be affected by the removal of records in a REMOVE TREE operation or by the repositioning of records in a MOVE TREE operation in the following situations: a GET1 command specifies NEXT, a GET1 command uses S2KCOUNT not equal to 0, or a GET1 command specifies LAST for a non-level-0 record. In those situations, retrieval movement through the data structure begins at the record whose address is the primary entry in the stack specified by the command.

    - If the record for the primary entry was deleted by a REMOVE TREE command or moved by a MOVE TREE command that specifies another stack or that is issued by another user in a Multi-User environment, the GET1 command is ignored, and return code 09 results. The database level of the removed or moved node is indicated in the eighth field of the COMMBLOCK.

    - If the data record for the primary stack entry is the top node of a subtree that was deleted by a REMOVE TREE command or that was moved by a MOVE TREE command specifying the same stack and if no subsequent command changed the stack, the following results can be obtained:

      - Return code 03 results, and the command (GET1 PRESENT, for example) is ignored if it specifies a subschema record at the level of the primary entry.

      - If a GET1 command requests a record from a level other than that of the primary entry, a check is made for the parent and right sibling of the removed or moved primary node. The command results in return code 03 if either the parent or the right sibling is not present. However, if both nodes are present, retrieval movement begins at the location of the removed or moved primary node.

12. SYSTEM 2000 software associates a stack and Locate File of the same number with a single database, according to the last database for which the stack or Locate File of that number was used. If, for example, a GET1 command establishes a primary entry in stack 5 for a database different from the database associated with Locate File 5, any addresses in Locate File 5 are erased, and the Locate File is no longer associated with a database.

13. If the MULTIPLE HOLDS option is enabled, the PLEX program can request any number of local holds on different records simultaneously. (The MULTIPLE HOLDS option is disabled by default. (See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for details about setting the MULTIPLE HOLDS option.) Local holds remain in effect until one of the following events occurs in the PLEX program:

    - A global hold is attempted for any database.
    - A FRAME command is issued for any database.
    - A synchpoint occurs.
    - The program terminates.
    - A DROP HOLD command occurs.
    - The QUEUE command is issued.

Once any of the above occurs, all of the PLEX program's local holds on the database(s) are dropped. Also, an update command no longer causes a local hold to be dropped.

See **HOLD Option** on page 80 for a complete discussion of local holds. Also, see **HOLD Option** on page 8-131 and **DROP HOLD** on page 8-144.

## GET1 NEXT

```
GET1 [/HOLD] [(stack-number)] SSR [inclusion-list] [NEXT]
                                                   [USING 0]
```

A GET1 command specifying NEXT retrieves data records by moving in logical order through the data structure.

The GET1 NEXT command is equivalent to GET1 *SSR* with no options, such as FIRST, LAST, PRESENT and so on. The equivalent of NEXT in a GET1 command is S2KCOUNT equal to 1. (See **GET1 S2KCOUNT** on page 8-39 for additional information.)

Several factors determine which data record is retrieved with GET1 NEXT. One is the subschema record specified in the command. Another factor is the level of the primary stack entry. Retrieval in logical order proceeds from the data record whose address is the primary stack entry.

A GET1 NEXT command has four possible results:

- If the specified stack has no primary stack entry, GET1 NEXT behaves like GET1 FIRST; that is, the first record in logical order for the specified subschema record is retrieved.
- If the schema record for the primary stack entry is in the same family as the schema record referred to in the command, retrieval movement through the data structure is based on the level of the primary stack entry and on the level of the schema record to which the command refers.
- If the schema record for the primary stack entry is not in the same family as the schema record referred to in the command, the GET1 command is ignored, and return code 03 results (see Example 9).
- If the data record for the primary stack entry was deleted by a REMOVE TREE operation or moved by a MOVE TREE operation, a successful retrieval depends on whether the REMOVE TREE or MOVE TREE command was given for the same stack, for another stack, or by another user.

Be sure to review the items listed under **Considerations** on page 8-9 for any additional information. In particular, note item 11.

The examples show different possibilities for retrieval with GET1 NEXT, depending on the level of the primary stack entry. The examples are summarized here.

**Example 1:** The primary entry is above the level of the specified SSR.

**Example 2:** The primary entry is below the level of the specified SSR.

**Example 3:** The primary entry is below the level of a specified level 0 SSR.

**Example 4:** The primary entry is at the level of the specified SSR.

**Example 5:** The primary and secondary entries are above the level of the specified SSR.

**Example 6:** A GET1 sequence reaches the end of the logical data structure.

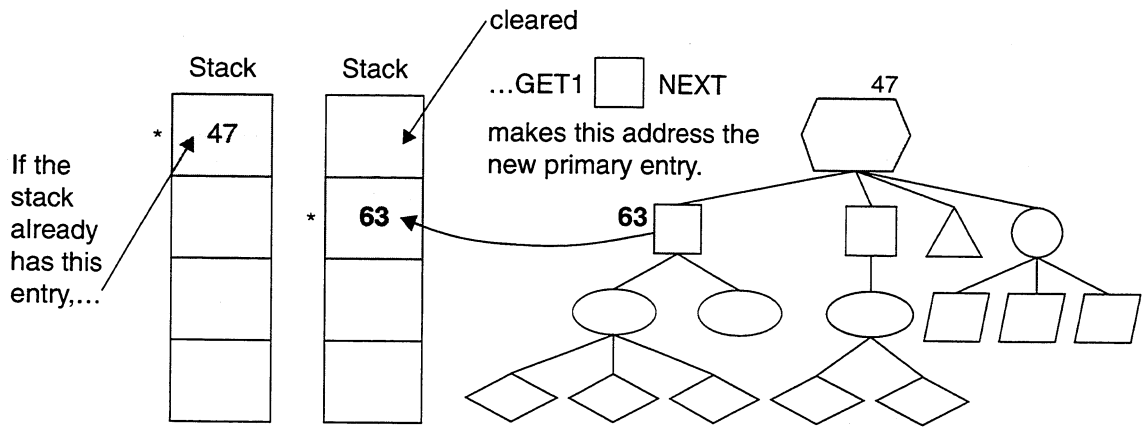
**Example 7:** A GET1 command is issued after the starting point was removed by a REMOVE TREE command for the same stack.

**Example 8:** A GET1 command is issued after the starting point was removed by a REMOVE TREE command for a different stack or by another user.

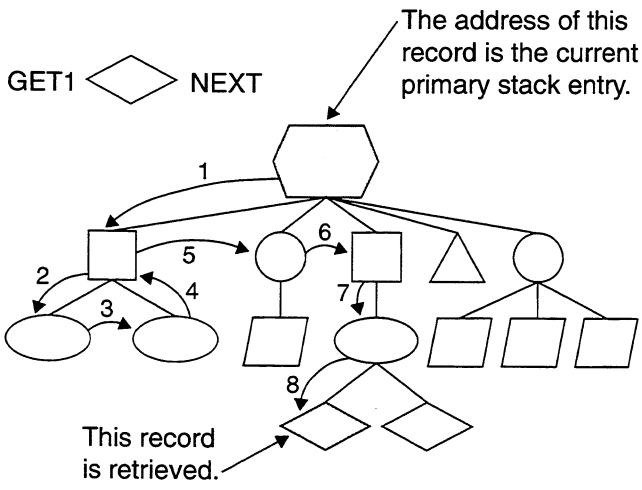
**Example 9:** A GET1 command refers to a schema record not in the family of the primary entry.

**Example 1**

The primary stack entry is above the level of the specified subschema record. When a descendant of the primary entry is retrieved, the address of the descendant becomes the primary entry. The previous primary entry is cleared from the stack.

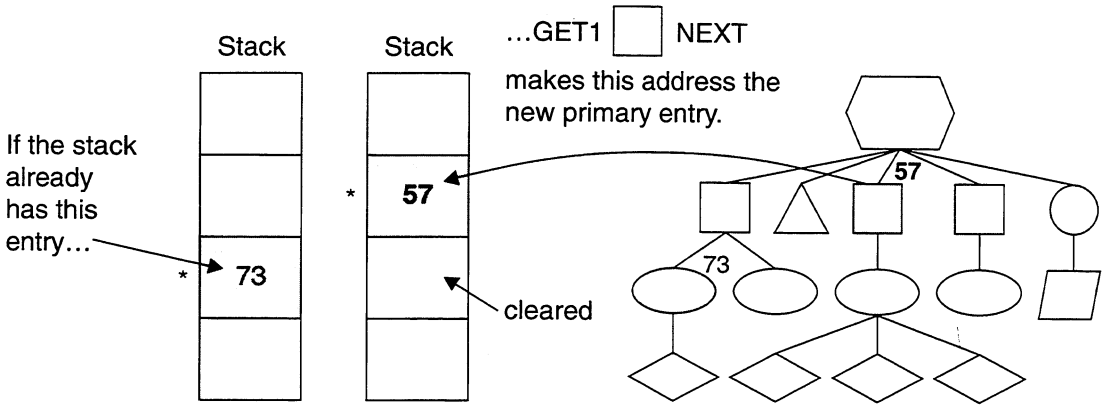



In general, when the primary entry is above the level of the specified subschema record, retrieval movement through the data structure proceeds downward from the record for the primary entry. Then, if necessary, it proceeds to the right and downward or upward, to the right, and downward, as shown below.



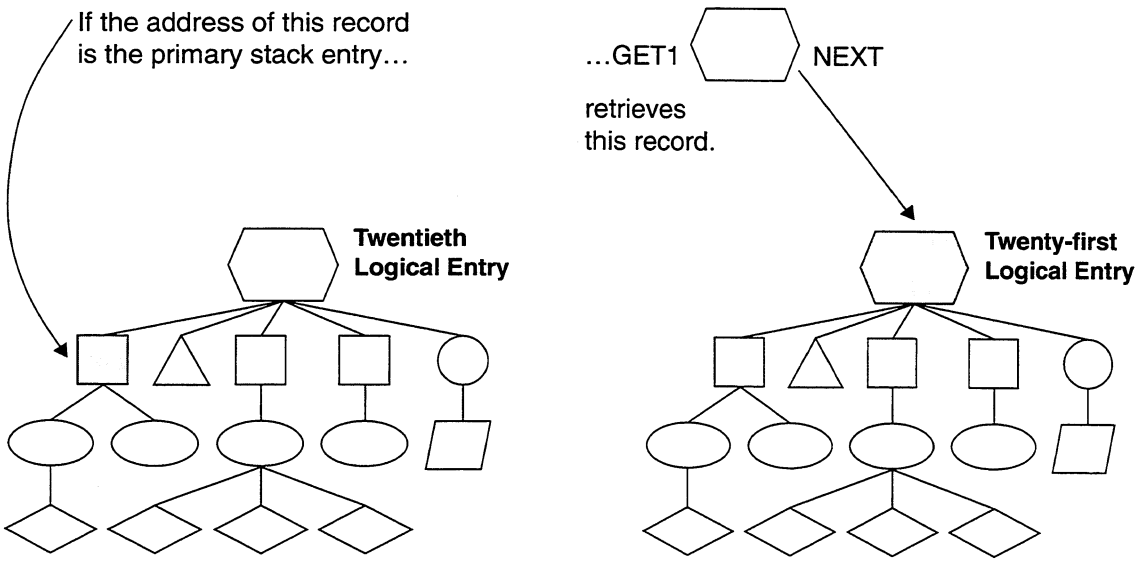
### Example 2

The primary stack entry is below the level of the specified subschema record. When the data record at the shallower level is retrieved with GET1, the address of the retrieved record becomes the primary entry. The previous primary entry is cleared from the stack.



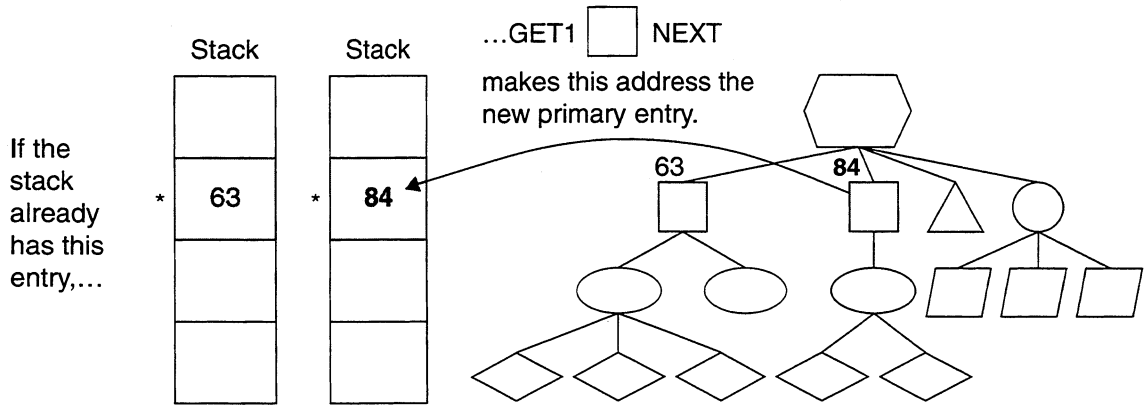

### Example 3

The primary stack entry is below the level of the specified subschema record, which is at level 0. The GET1 NEXT command retrieves the next level 0 record, that is, the root node of the next logical entry.

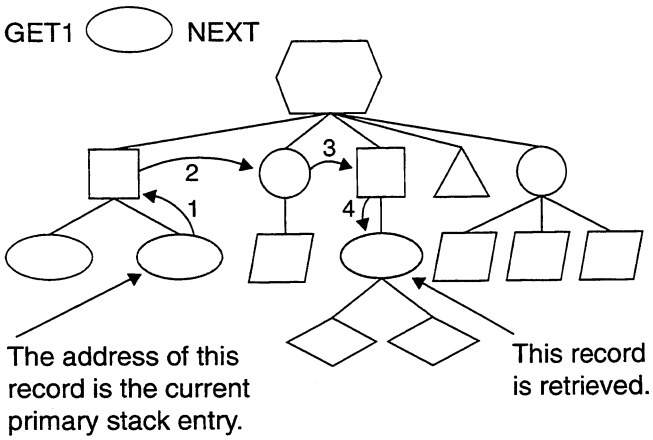


**Example 4**

The primary stack entry is at the level of the specified subschema record. When the next similar sibling record is retrieved with GET1, the address of that record becomes the new primary entry.

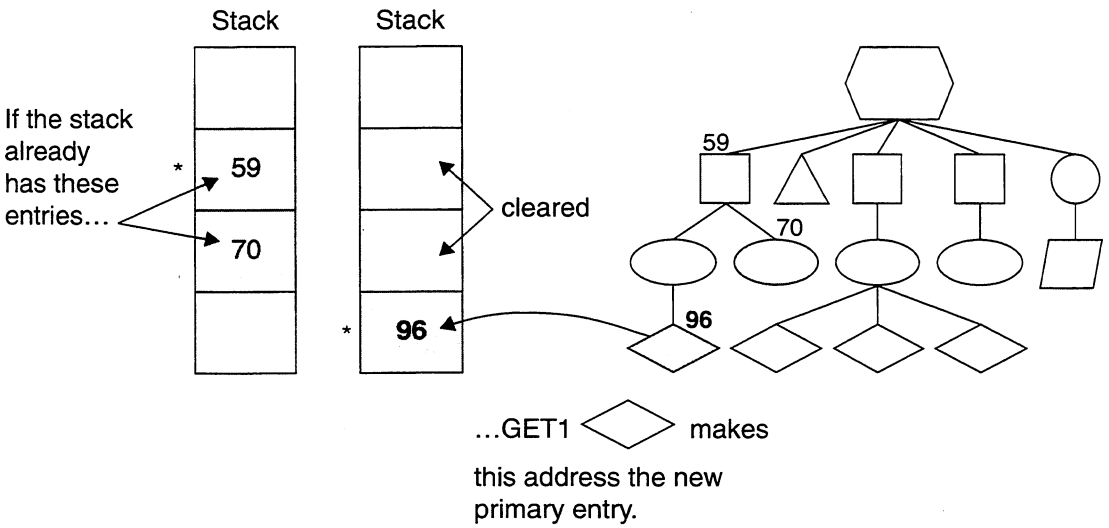



In general, when the primary entry is at the level of the specified subschema record, retrieval movement through the data structure is rightward from the primary entry, or upward from the primary entry to the right and then downward, as shown below.



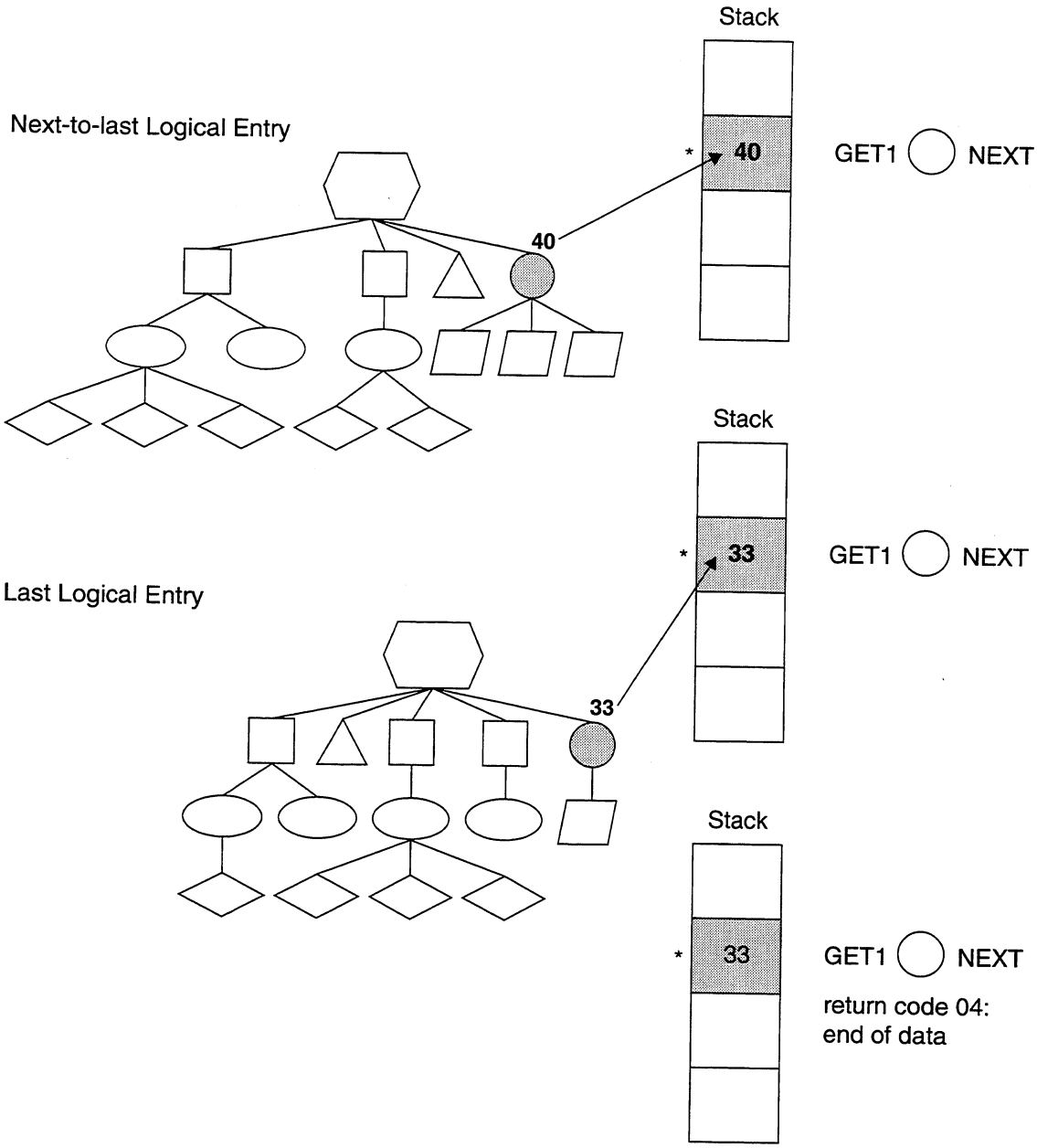
**Example 5**

Both the primary stack entry and secondary stack entry are above the level of the specified subschema record. This retrieval is based on the primary entry only; the secondary entry is ignored. Both the primary entry and the secondary entry are cleared when the new primary entry is established for the retrieved record.



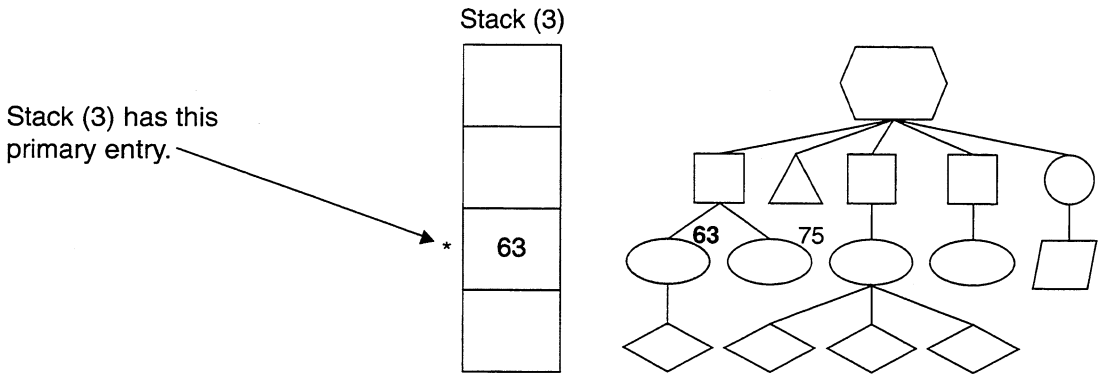
**Example 6**

Assume that the two data trees shown here represent the last two logical entries in the database. When given repeatedly, the GET1 command finally results in return code 04, signifying that the end of the logical data structure was reached.



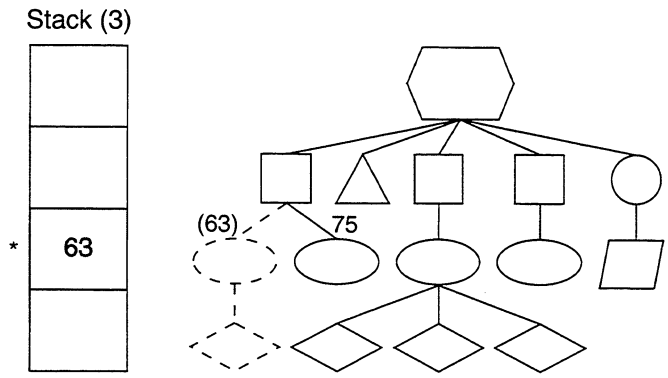
### Example 7

GET1 NEXT results in data retrieval if a REMOVE TREE operation was performed using the same stack.



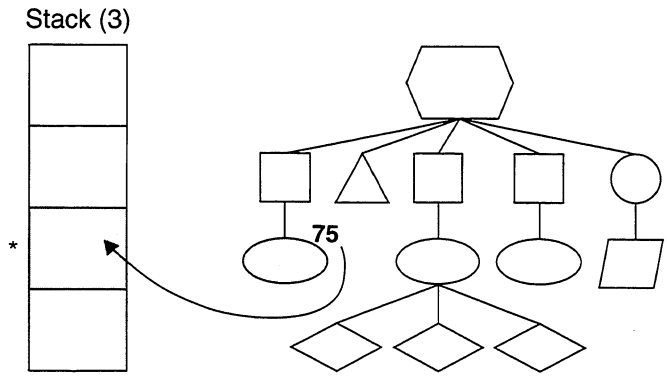

A REMOVE TREE command is issued for the primary node of this stack:

REMOVE TREE (3) ⬭



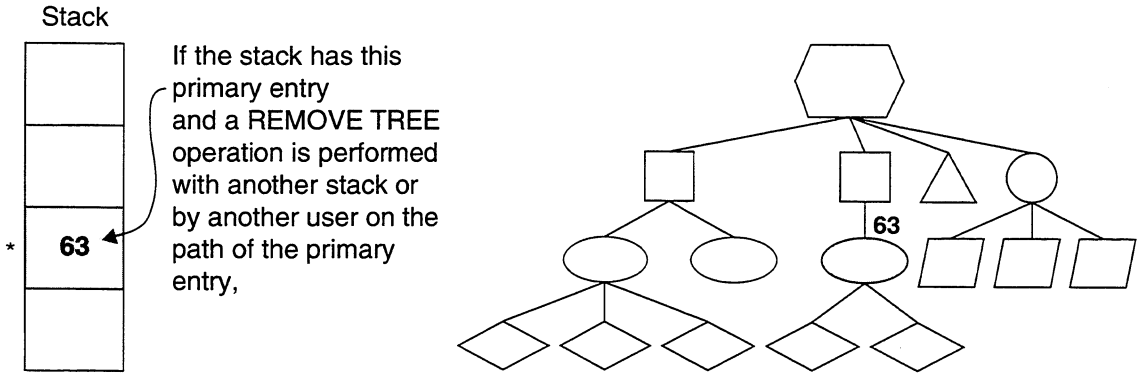

This GET1 NEXT command will still retrieve the next record for the specified SSR, even though the primary node was removed:

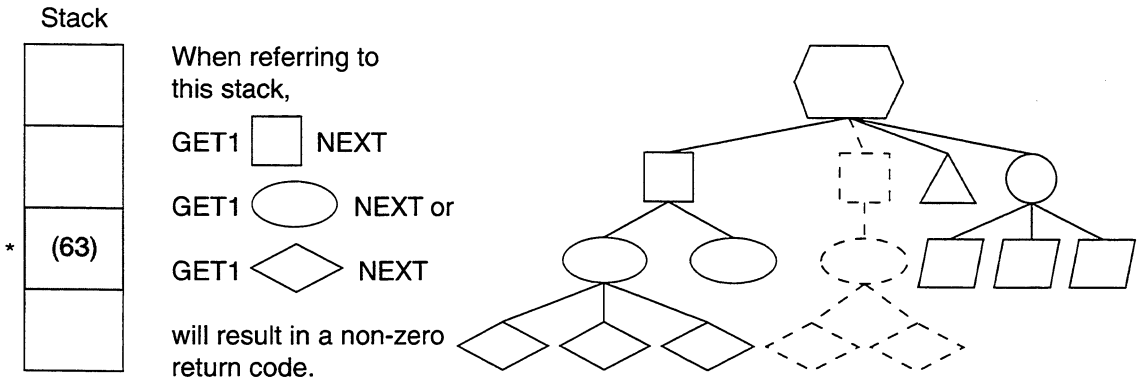
GET1 (3) ⬭ NEXT



**Example 8**

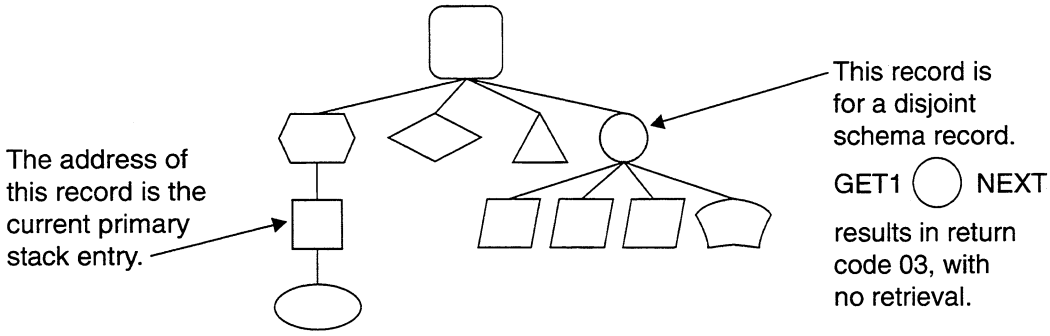
If a REMOVE TREE operation (with another stack or another user) removed the node from which GET1 NEXT processing was to proceed, the GET1 NEXT command results in return code 09.



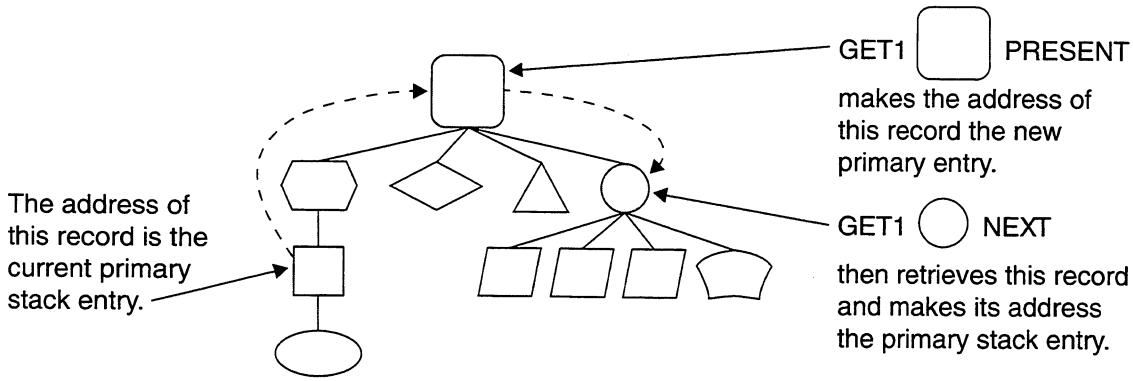



**Example 9**

The schema record referred to (through the SSR) in the command is not in the schema family of the schema record for the primary stack entry. Because the specified SSR and the SSR for the primary entry are thus disjoint, the GET1 NEXT command results in return code 03.




In order to retrieve a data record for a disjoint SSR, the program must first give a GET1 PRESENT command to retrieve a common ancestor, then issue the GET1 NEXT command.



## GET1 FIRST

```
GET1 [/HOLD] [(stack-number)] SSR [inclusion-list] [FIRST]
                                                   [USING 0]
```

A GET1 command specifying FIRST retrieves the first data record in logical order that is associated with the specified subschema record. Retrieval with GET1 FIRST does not depend on the entries already in the stack. Rather, GET1 FIRST always retrieves the first record for the specified subschema record.

Be sure to review the items listed under **Considerations** on page 8-9 for any additional information.

The examples show different retrieval situations with GET1 FIRST. They are summarized here.

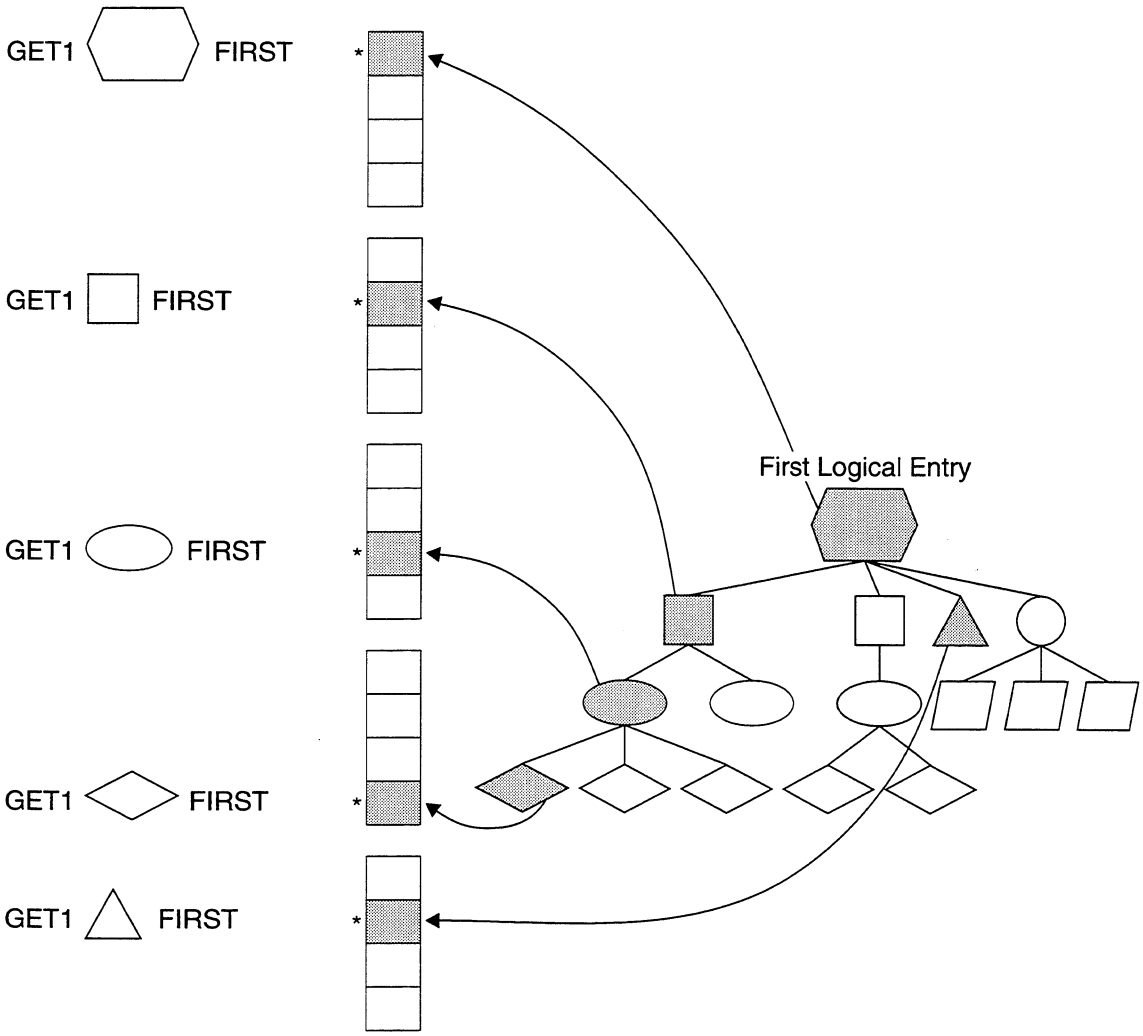
**Example 1:** a sequence of GET1 FIRST commands specifying different subschema records.

**Example 2:** a GET1 FIRST command after primary entries were already established.

**Example 3:** a GET1 FIRST command after the first logical entry was deleted by REMOVE TREE.

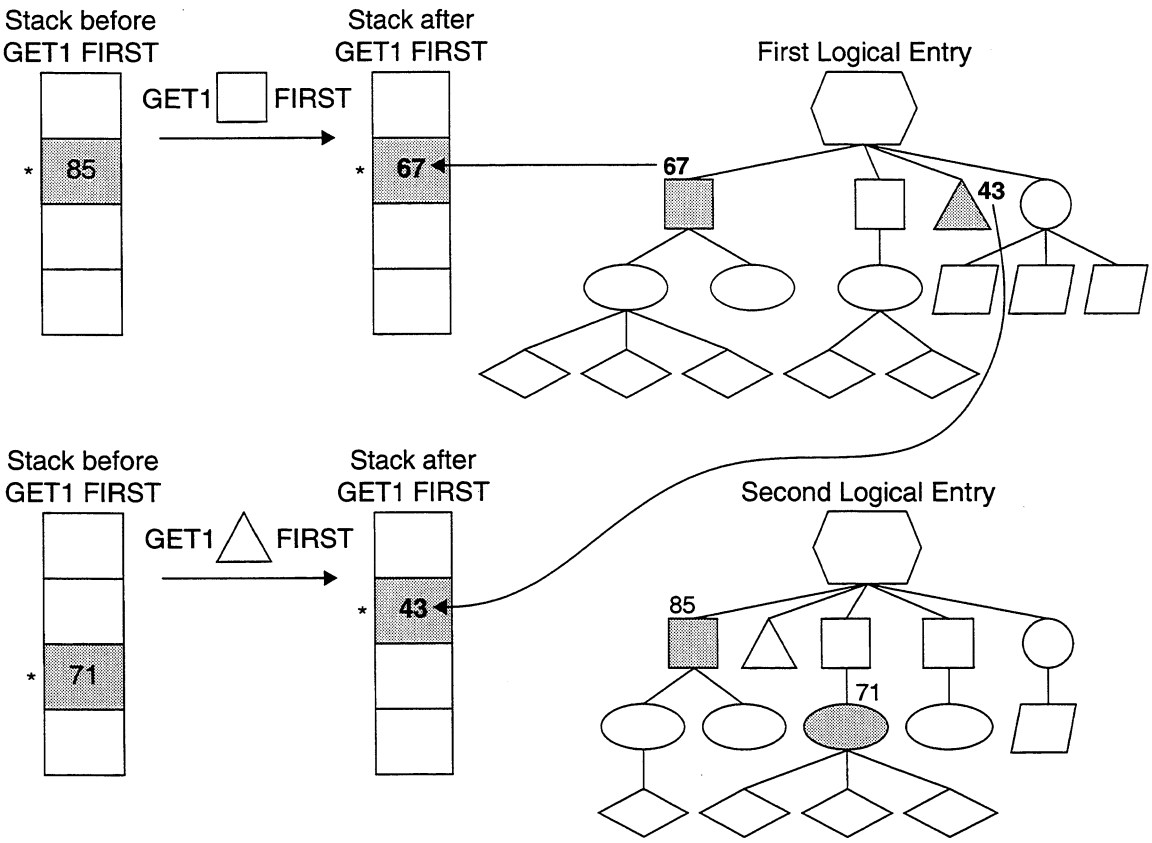
**Example 1**

GET1 FIRST commands could specify different subschema records to retrieve these data records, making the address of each record in turn the primary stack entry. Assume that the data tree shown here represents the first logical entry in logical order in the database.



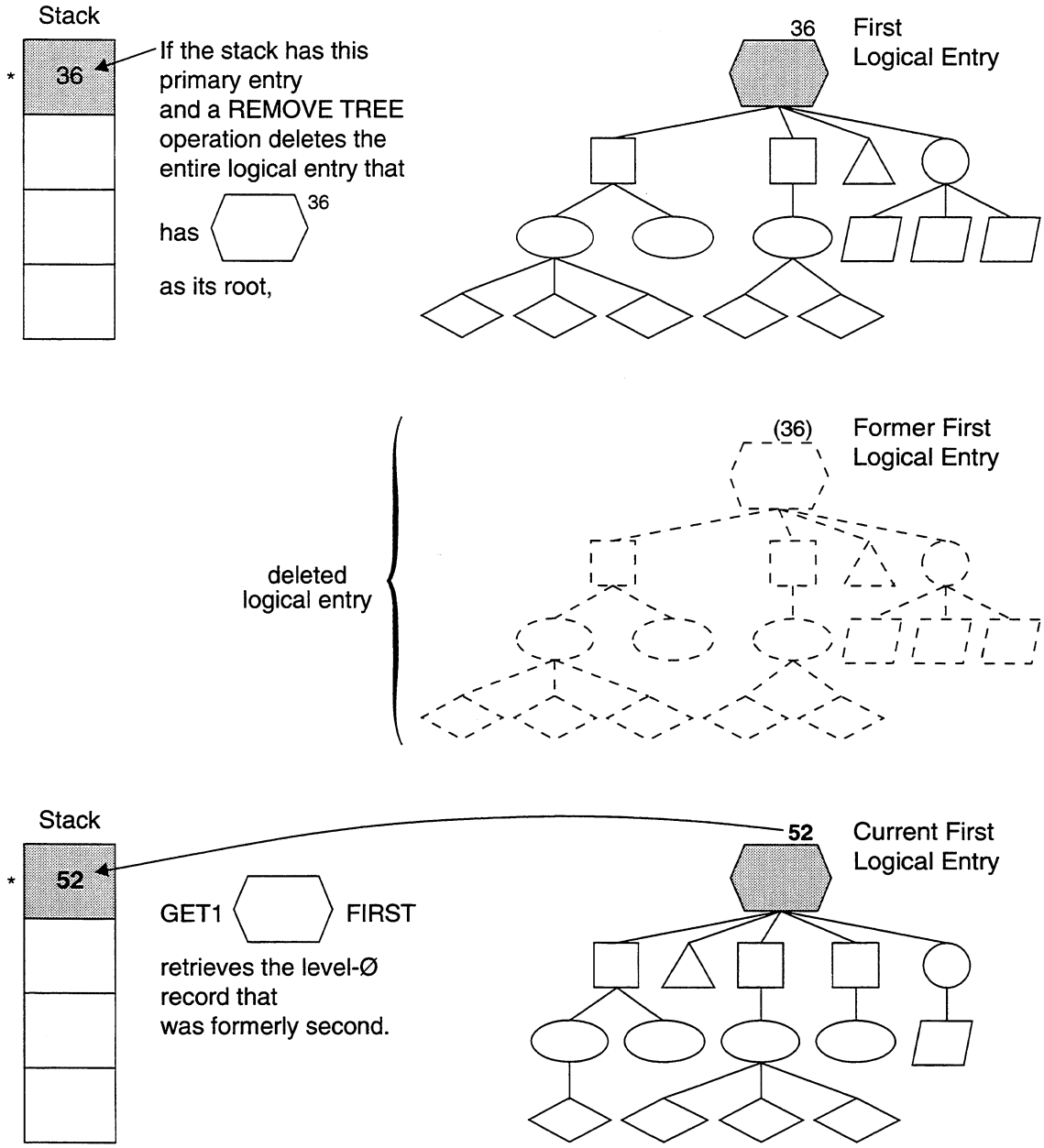
**Example 2**

Even after a primary stack entry is established for a record that comes later in logical order, the record retrieved with GET1 FIRST is always the first one in logical order for the specified SSR. Assume that the data trees shown here represent the first and second logical entries in logical order in the database.



### Example 3

At first, the primary stack entry is the address of the level 0 record of the first logical entry. Then, a REMOVE TREE command deletes that entire logical entry. A subsequent GET1 FIRST command specifying the level 0 SSR retrieves the level 0 record of the logical entry that was formerly second.



## GET1 LAST

```
GET1 [/HOLD] [(stack-number)] SSR [inclusion-list] [LAST]
                                  [USING 0]
```

A GET1 command specifying LAST retrieves the last data record in logical order that is associated with the specified subschema record. Just as with GET1 NEXT or GET 1 S2KCOUNT, retrieval with GET1 LAST for a non-level-0 record starts at the primary stack entry and is determined by the specified subschema record. For a record at a level 0, GET1 LAST simply uses a Hierarchical Table pointer, avoiding retrieval movement through the data structure.

When a GET1 LAST command specifies an SSR at a level deeper than level 0, retrieval movement starts from the beginning of the data structure if the stack has no current primary entry, or else it starts from the record whose address is the primary entry. This movement results in retrieval if the last record for the SSR comes after the starting point.

The last record might not be in the last logical entry of the database. However, if it is, you can save processing time by coding a GET1 LAST command for the level 0 record of the last logical entry. Execution of the level 0 GET1 LAST command is very rapid, avoiding retrieval movement through the entire data structure. Once the level 0 primary stack entry is established, retrieval movement for GET1 LAST for the non-level-0 record is confined to the last logical entry.

A GET1 LAST command for a level 0 record always retrieves the last such record currently in the database, producing return code 00. However, if a GET1 LAST command refers to a non-level-0 SSR and if it causes the retrieval of the record whose address is already the primary stack entry, return code 50 results.

When the primary entry is beyond the last data record for the specified SSR in logical order, return code 04 results. The primary entry is beyond the last data record for the subschema record in either of the following situations:

- The last data record for the subschema record is an ancestor of the data record for the primary entry.
- The data record for the primary entry is in a path that follows the path of the last data record for the subschema record in logical order.

Be sure to review all the considerations listed under **Considerations** on page 8-9 for additional information. In particular, see item 11 for the effect of removing nodes(s) from the database.

The examples show different situations of retrieval with GET1 LAST. They are summarized here.

**Example 1:** a sequence of GET1 LAST commands specifying different subschema records.

**Example 2:** GET1 LAST commands that can result in return code 03 or 04, depending on the primary stack entry.

**Example 3:** GET1 LAST to retrieve the same record results in return code 50.

**Example 4:** GET1 LAST to retrieve the same record results in return code 00.

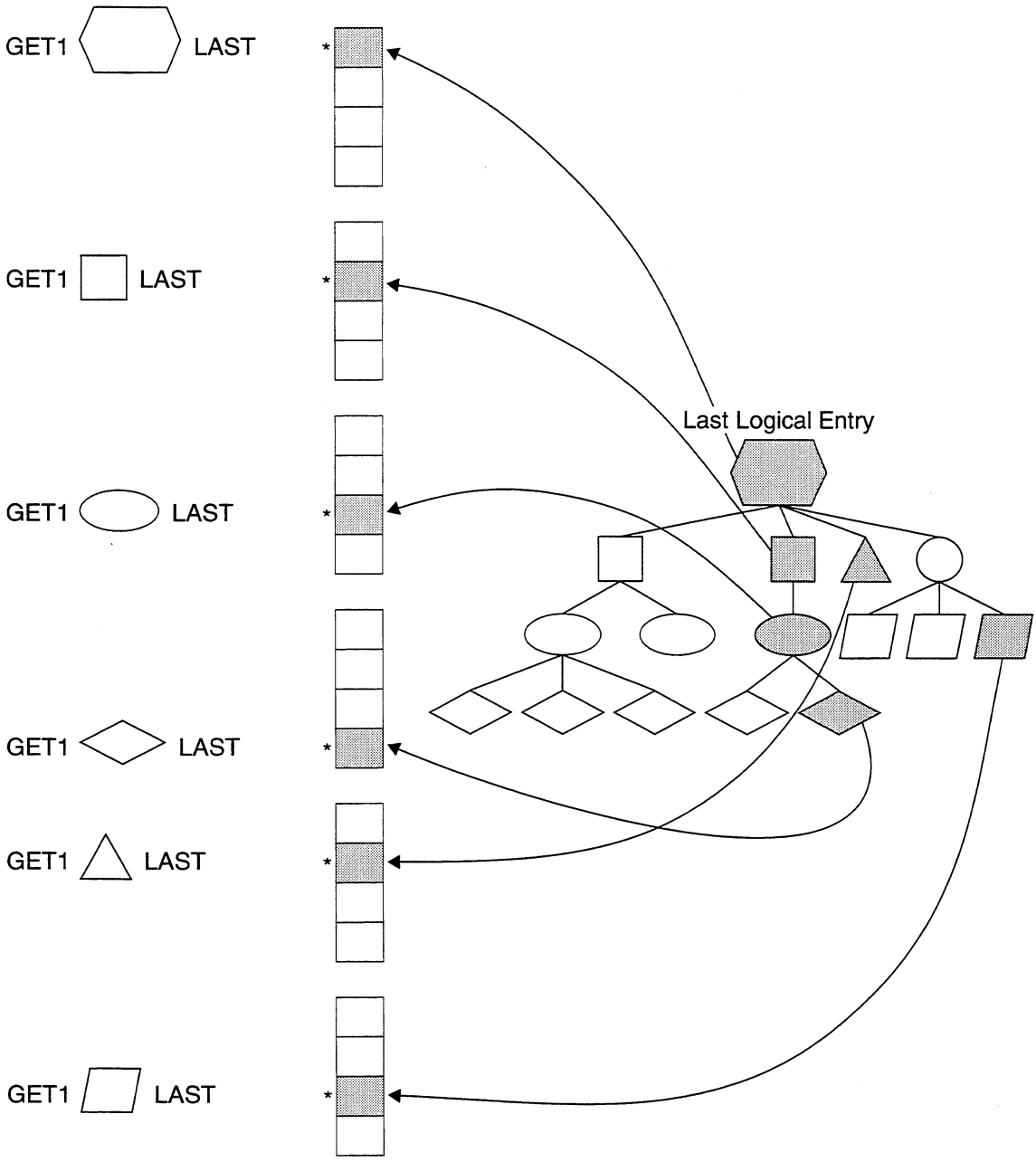
**Example 5:** GET1 LAST after the last logical entry was deleted by REMOVE TREE.

**Example 6:** GET1 LAST after a REMOVE TREE command for the same stack.

**Example 7:** GET1 LAST after a REMOVE TREE command for a different stack or from another user.

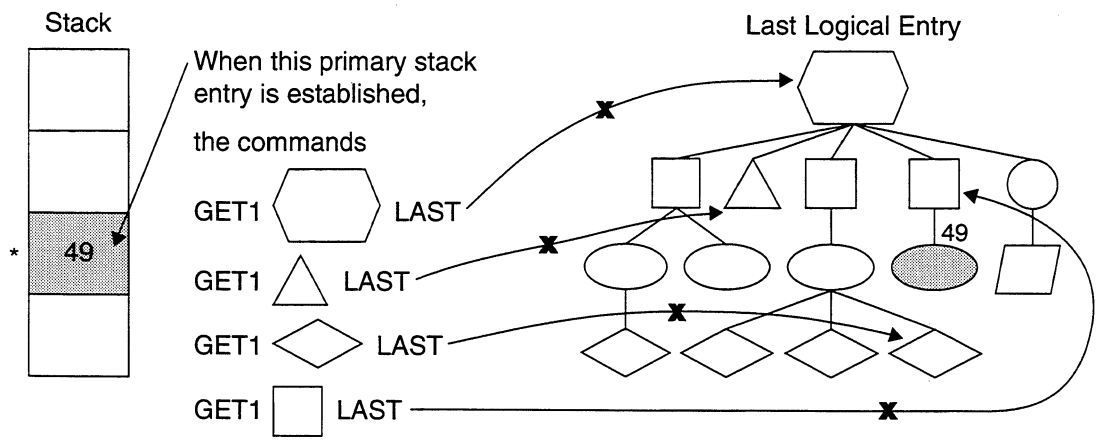
**Example 1**

GET1 LAST commands can specify different subschema records to retrieve these data records, making the address of each record the primary stack entry. Assume that the data tree shown here represents the last logical entry in logical order in the database.



### Example 2

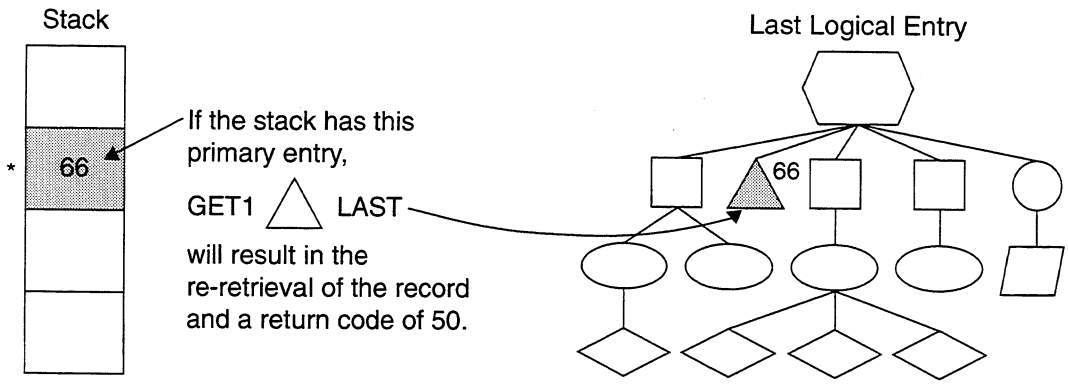
Depending on the current primary stack entry, a GET1 LAST command can result in return code 03 or 04 and no retrieval. This is the situation when the primary entry is below level 0 and the last record for the specified SSR comes earlier in logical order than the record for the primary entry.




will, if they refer to this stack, result in a non-zero return code and *will not* cause retrieval of the indicated records. The reason is that the primary stack entry is beyond these records in logical order.

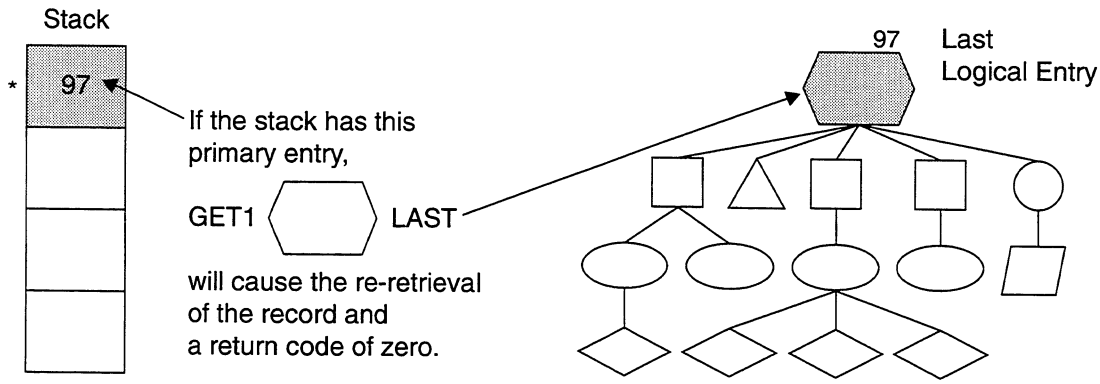
**Example 3**

Depending on the current primary stack entry, a GET1 LAST command can retrieve the same record again and issue return code 50. This happens when the primary entry is below level 0 and is the address of the last record for the specified SSR.



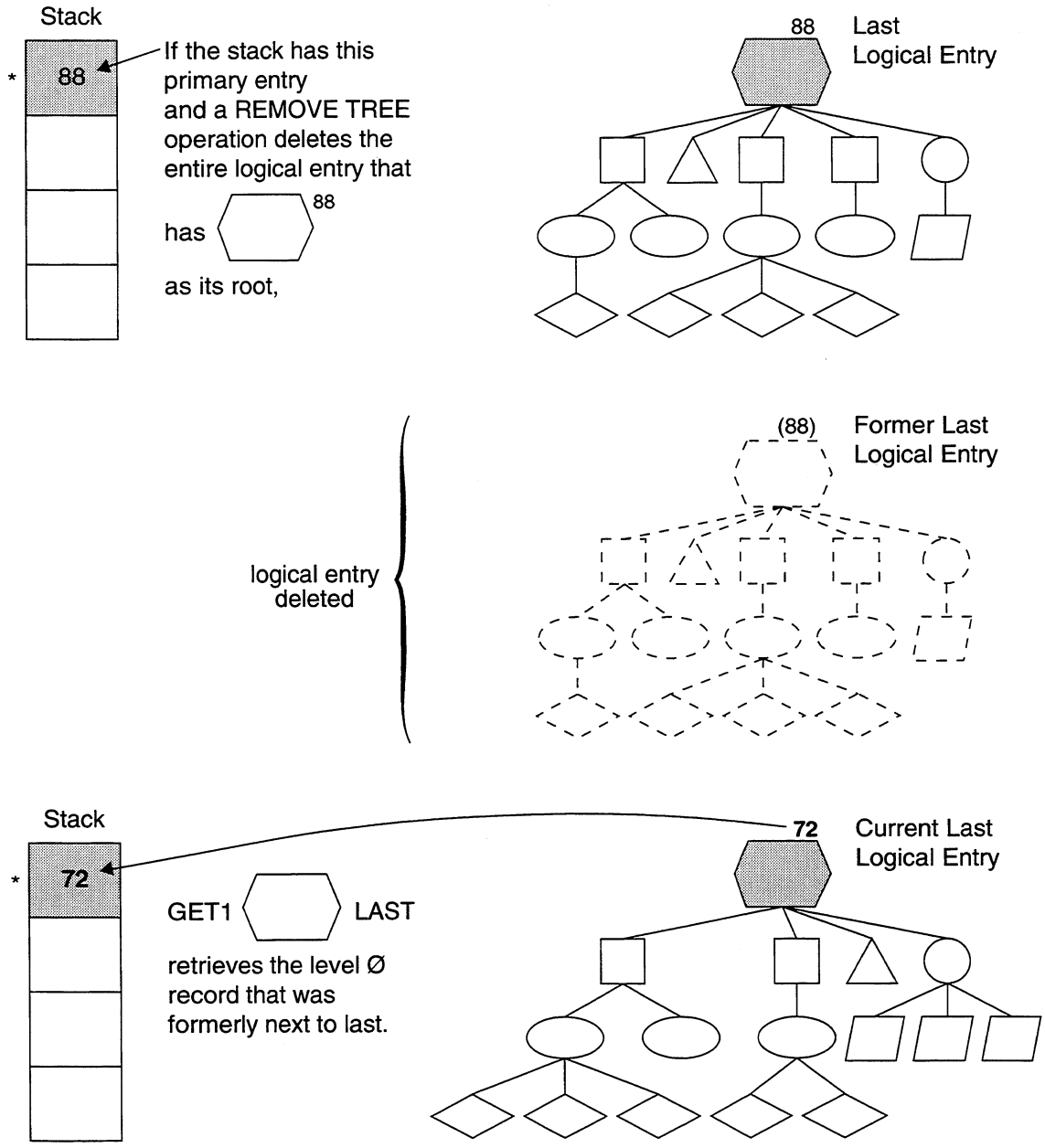

**Example 4**

If the current primary stack entry is for the last level 0 record in the database, a GET1 LAST command specifying the level 0 SSR retrieves the same record and results in return code 00.



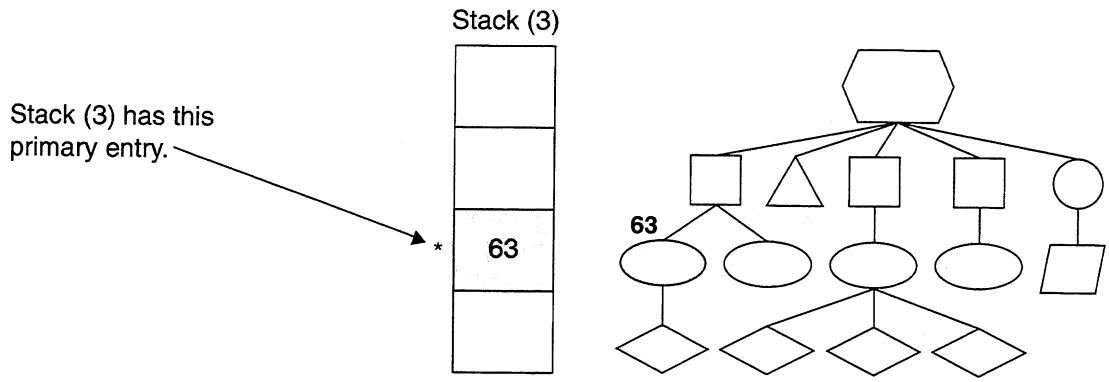
### Example 5

At first in this example, the primary stack entry is the address of the level 0 record of the last logical entry. Then a REMOVE TREE command deletes that entire logical entry. A subsequent GET1 LAST command specifying the level 0 SSR retrieves the level 0 record of the logical entry that was formerly next to last.



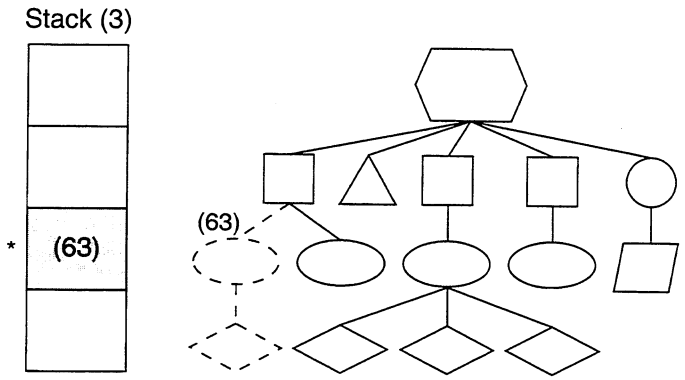
**Example 6**

GET1 LAST results in data retrieval if a REMOVE TREE operation was performed using the same stack.



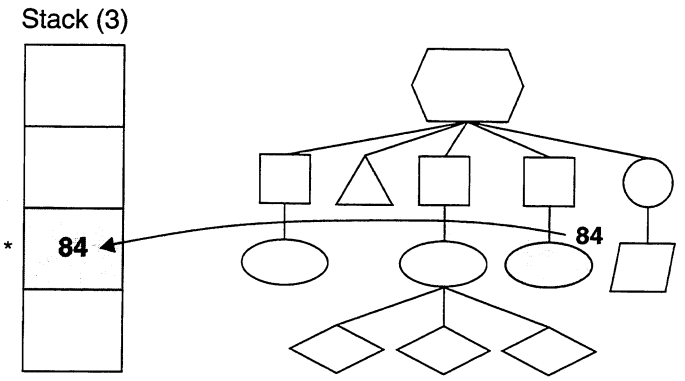

A REMOVE TREE command is issued for the primary node of this stack:

REMOVE TREE (3)



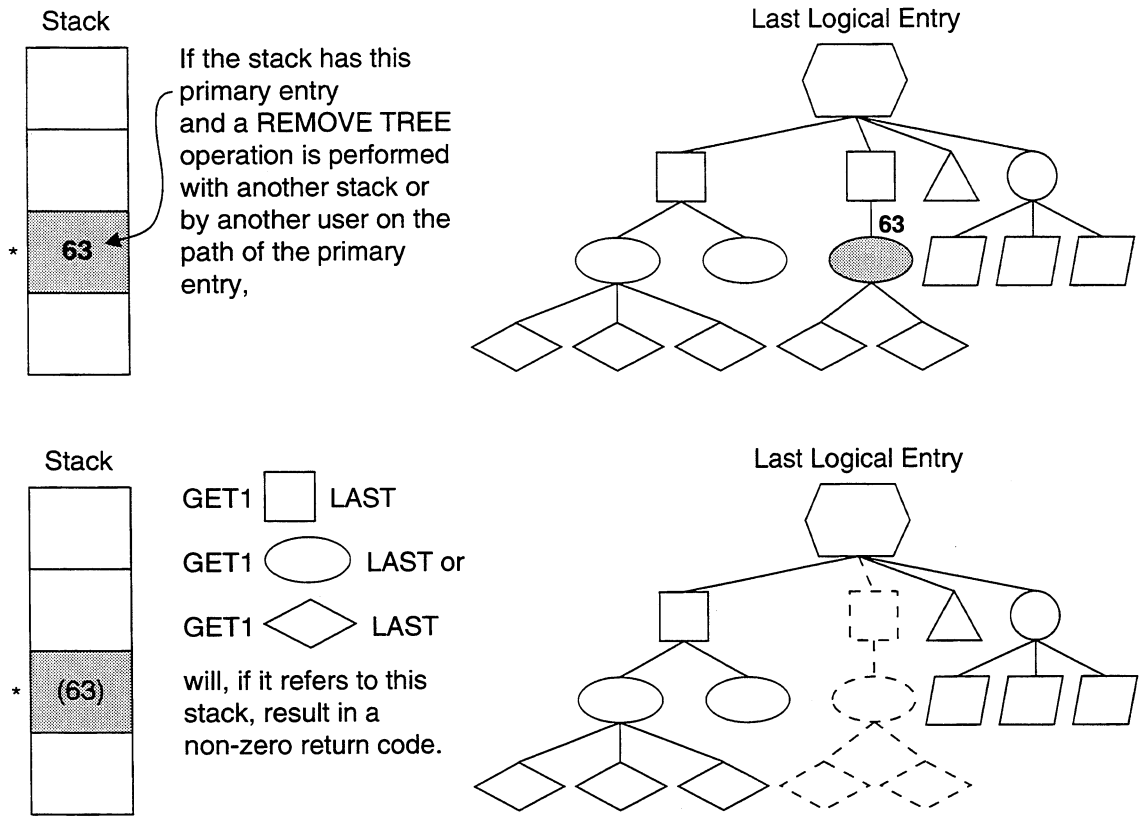

This GET1 LAST command will still retrieve the last record for the specified SSR, even though the primary node was removed:

GET1 (3) LAST



### Example 7

If a REMOVE TREE operation was performed with another stack or by another user for a record below level 0, GET1 LAST commands using a stack with one of the removed nodes can result in return code 09.



## GET1 PRESENT

```
GET1 [/HOLD] [(stack-number)] SSR  [inclusion-list] [PRESENT]
                                   [USING 0]
```

A GET1 command specifying PRESENT retrieves a data record at or above the level of a stack entry and in the path of that entry.

The PRESENT option in a GET1 command is equivalent to S2KCOUNT equal to zero. (See **GET1 S2KCOUNT** on page 8-39 for additional information.)

When a GET1 PRESENT command causes the retrieval of a data record whose address is already the primary stack entry, the result is return code 50.

If a stack entry is for a data record not in the family of a record for the specified SSR, return code 03 results.

If either a REMOVE TREE or MOVE TREE command removed or moved a record whose address was in the specified stack, the results of a subsequent GET1 PRESENT command are as follows:

- If the command specifies a subschema record at the level of the removed or moved node, return code 09 results, and the command is ignored. This is the case regardless of whether the node was removed or moved by the same program or by another user.

- If the stack entry for the removed or moved node is the highest entry below the level of the specified subschema record, the following results can occur:

  - If the REMOVE TREE or MOVE TREE command used this stack and no subsequent command changed the stack, the software checks for the parent of the removed or moved node. If it is present, the retrieval is performed.

  - If the REMOVE TREE or MOVE TREE command used another stack or was issued by another user in a Multi-User environment, return code 09 results, and the eighth field of the COMMBLOCK indicates the level of the removed or moved node.

Be sure to review the considerations listed under **Considerations** on page 8-9 for any additional information.

The examples show different situations of retrieving with GET1 PRESENT. They are summarized here.

**Example 1:** GET1 PRESENT retrieves the root node of a logical entry.

**Example 2:** GET1 PRESENT requests a record below the lowest-level stack entry, resulting in return code 03.

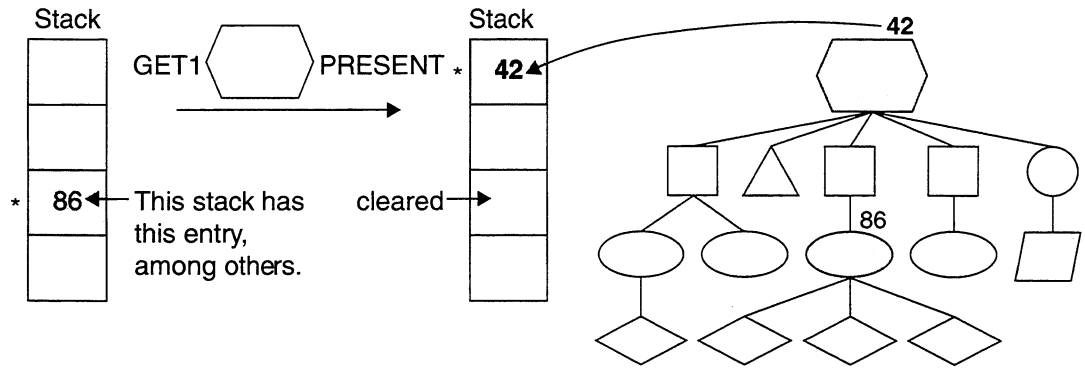
**Example 3:** GET1 PRESENT requests a record not in the same path as a stack entry, resulting in return code 03.

**Example 4:** GET1 PRESENT moves backward in logical order for retrieval.

**Example 5:** GET1 PRESENT is issued after REMOVE TREE command for a different stack or by another user.

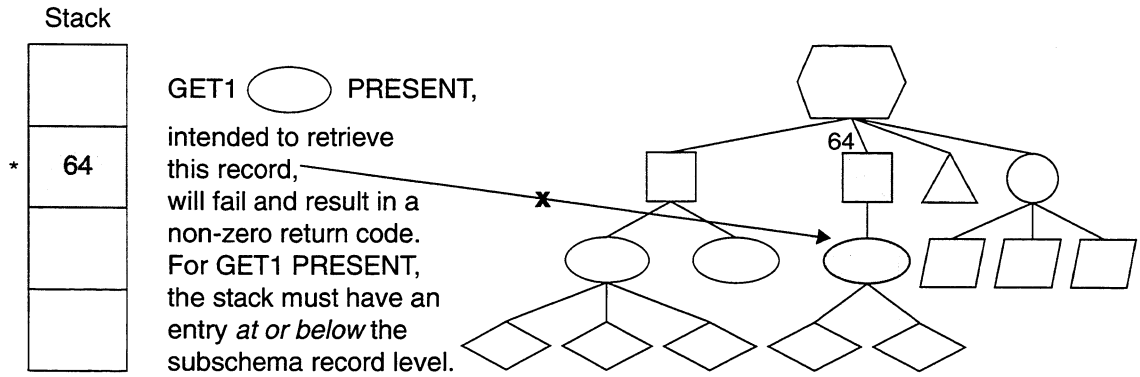
### Example 1

This GET1 PRESENT command retrieves the level 0 record of the logical entry.



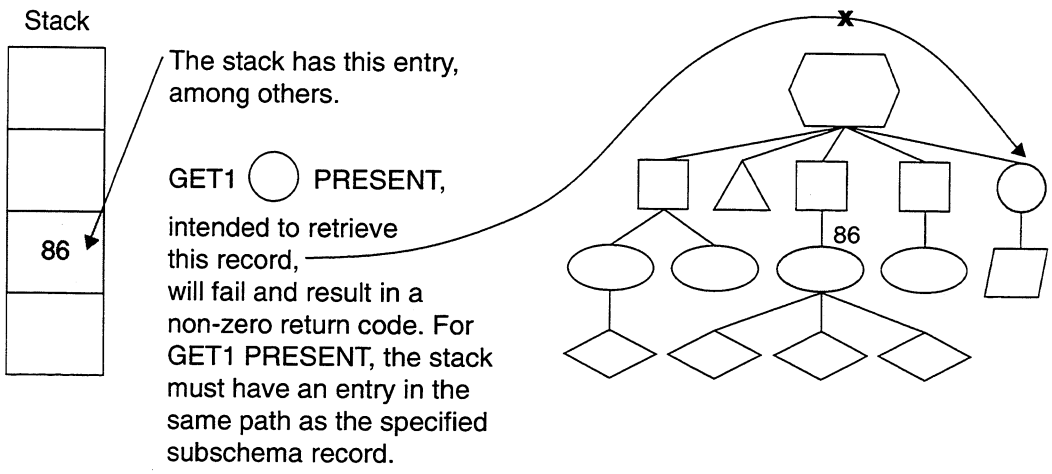

### Example 2

A GET1 PRESENT command must specify a subschema record at or above the level of a stack entry. If it does not, return code 03 results.



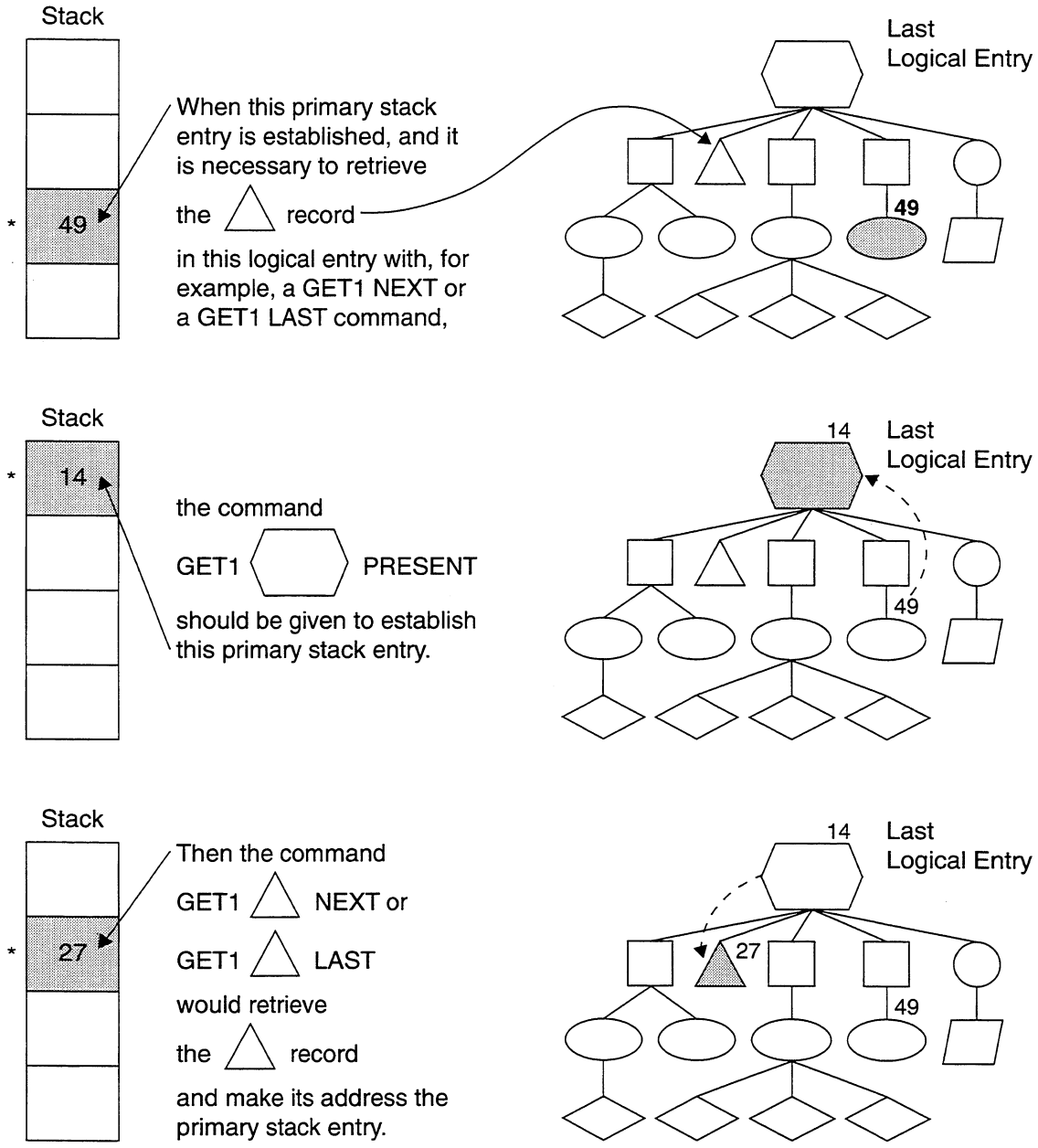
**Example 3**

A GET1 PRESENT command must specify a subschema record in the same path as the record for a stack entry. If it does not, return code 03 results.



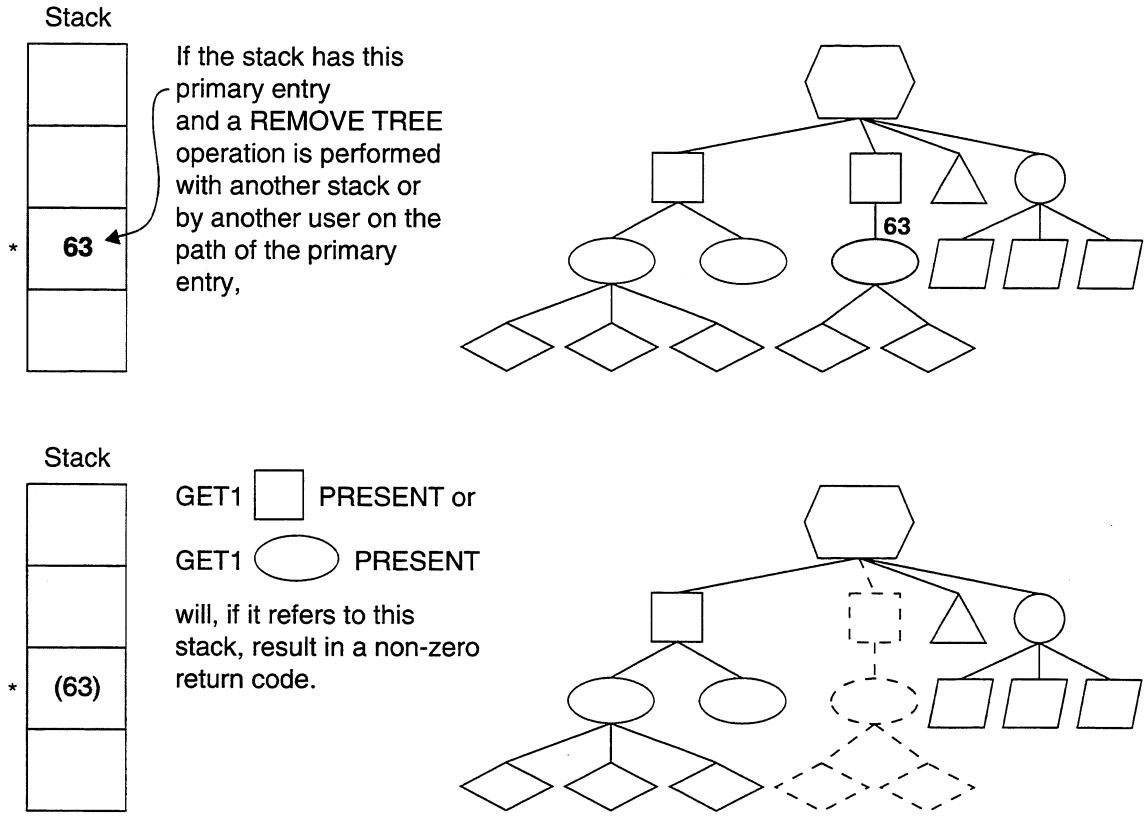
**Example 4**

A GET1 PRESENT command can be used in a sequence of GET1 commands to move backward in logical order.



**Example 5**

If a REMOVE TREE operation removed the record at the level of the specified subschema record, the GET1 PRESENT command results in return code 09.



## GET1 S2KCOUNT

```
GET1 [/HOLD] [(stack-number)] SSR [inclusion-list] [S2KCOUNT]
                                                   [USING 0]
```

You can use S2KCOUNT in the GET1 command to provide flexibility in retrieval during a program run and from one run to another. Depending on the value a program assigns to S2KCOUNT, a GET1 S2KCOUNT command can be the equivalent of GET1 NEXT or GET1 PRESENT. More importantly, it provides selectivity in retrieval that is not possible with NEXT or PRESENT; it allows you to navigate through the database retrieving every nth record, where n=S2KCOUNT.

S2KCOUNT has no default value. Before a program uses S2KCOUNT in a GET1 command, it must move a value to the variable S2KCOUNT, which resides in the S2KDUM control block. This value must be 0 or a positive integer for use with a GET1 command. It cannot be a negative value as it can with the GET command.

The value of S2KCOUNT tells SYSTEM 2000 software to skip to a data record that comes later in logical order than a similar data record already retrieved. If, for example, S2KCOUNT is set equal to 3, the command GET1 S2KCOUNT results in retrieval of the third similar data record beyond the current one in the logical data structure. If no similar data record was retrieved yet, retrieval with GET1 S2KCOUNT starts from the beginning of the logical data structure.

Following are equivalents of the GET1 S2KCOUNT command, which depend on the value assigned to S2KCOUNT:

| When | GET1 S2KCOUNT means |
|---|---|
| S2KCOUNT = 1 | GET1 NEXT |
| S2KCOUNT = 0 | GET1 PRESENT |

Be sure to review the items listed under **Considerations** on page 8-9 for any additional information. In particular, see item 11 concerning the effect of removing or moving nodes in a database for S2KCOUNT not equal to 0. See **GET1 PRESENT** on page 8-34 concerning the effect of removing or repositioning nodes and other general information that applies when S2KCOUNT equals 0. Review **GET1 NEXT** on page 8-12 if S2KCOUNT equals 1.

The examples show different retrieval situations using GET1 S2KCOUNT. They are summarized here.

**Example 1:** GET1 with S2KCOUNT equal to 1.

**Example 2:** GET1 with S2KCOUNT equal to 2, then 0.

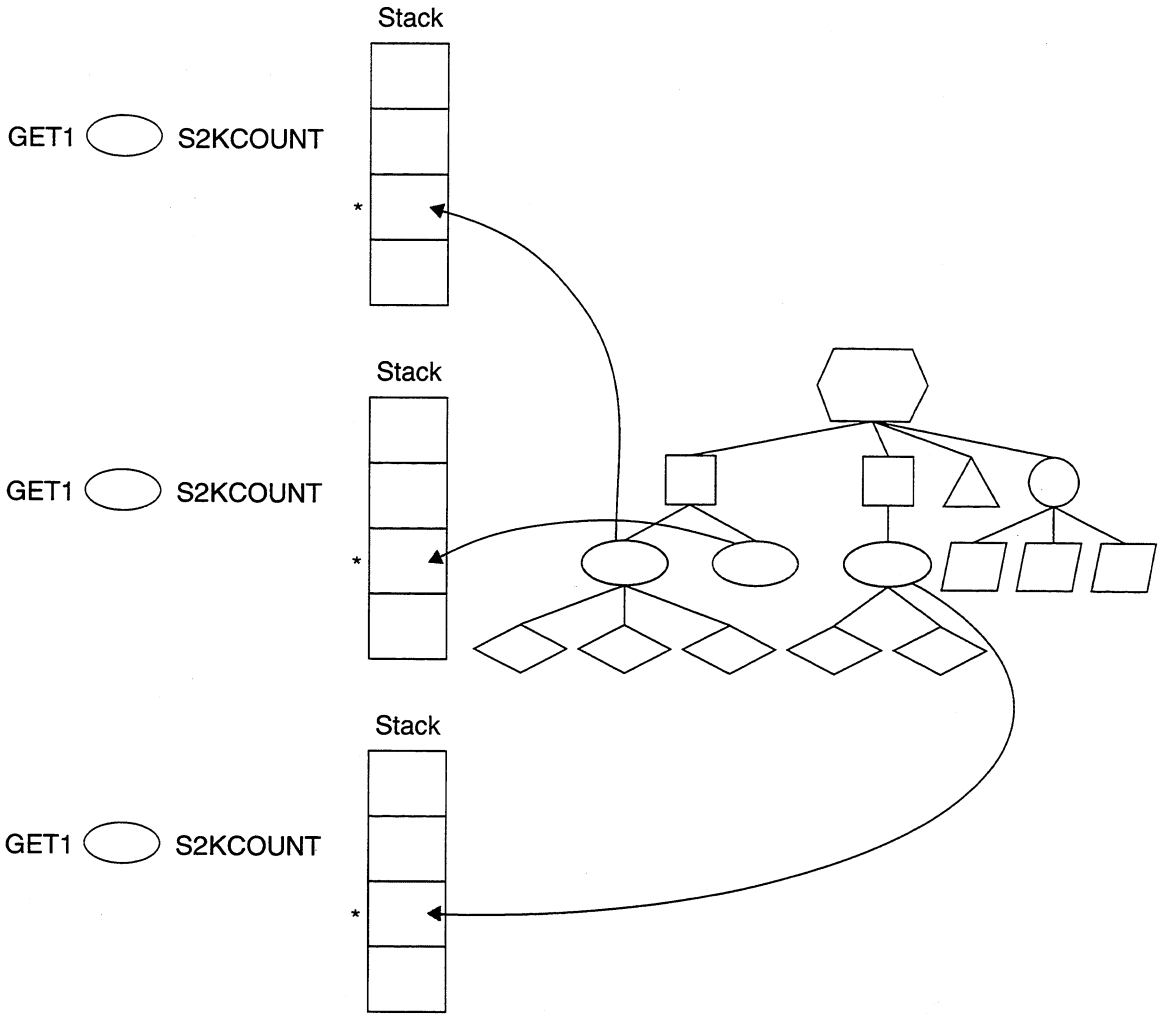
**Example 3:** GET1 S2KCOUNT after a REMOVE TREE command for the same stack.

**Example 4:** GET1 S2KCOUNT after a REMOVE TREE command for a different stack or from another user.

**Example 1**

With S2KCOUNT equal to 1, a GET1 S2KCOUNT sequence retrieves each record for the specified SSR. This is equivalent to GET1 NEXT.

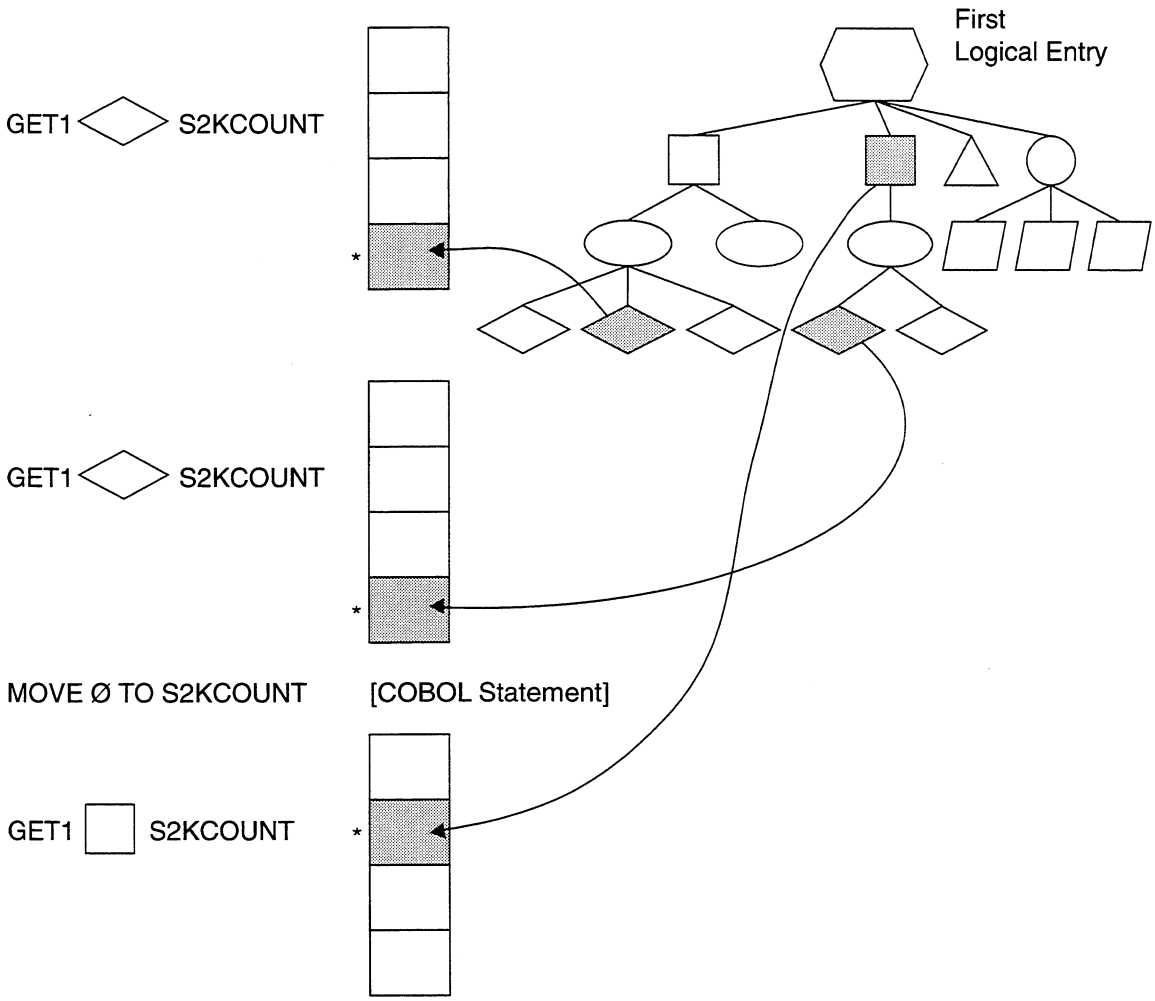
MOVE 1 TO S2KCOUNT [COBOL Statement]



**Example 2**

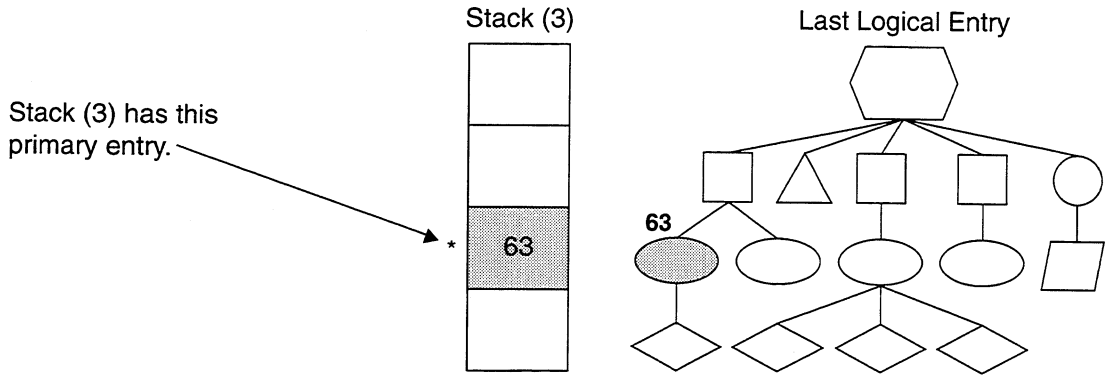
With S2KCOUNT equal to 2, a GET1 S2KCOUNT sequence retrieves every other record for the specified SSR. S2KCOUNT can be changed to zero to make it the equivalent of PRESENT. Assume the data tree represents the first logical entry in logical order.

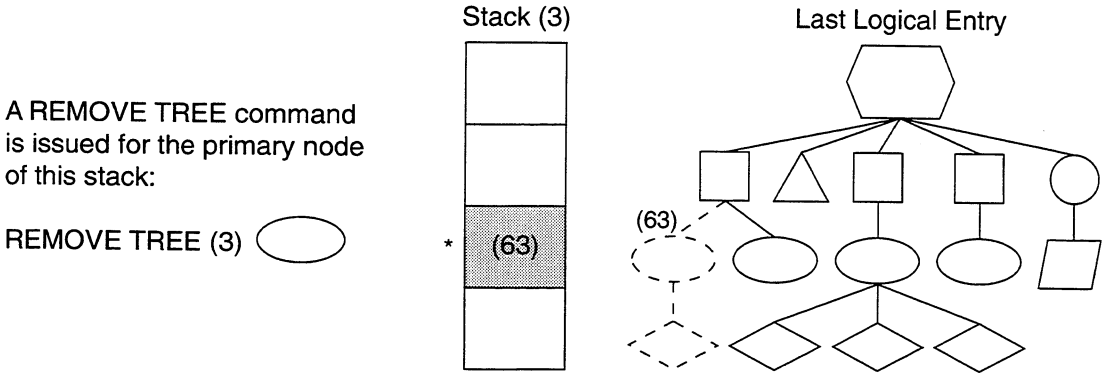
MOVE 2 TO S2KCOUNT [COBOL Statement]

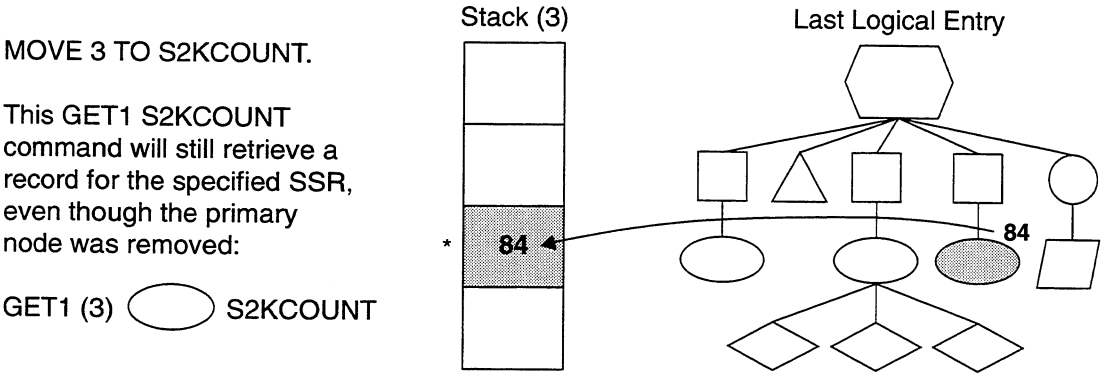


**Example 3**

GET1 S2KCOUNT results in data retrieval if a REMOVE TREE operation was performed using the same stack.

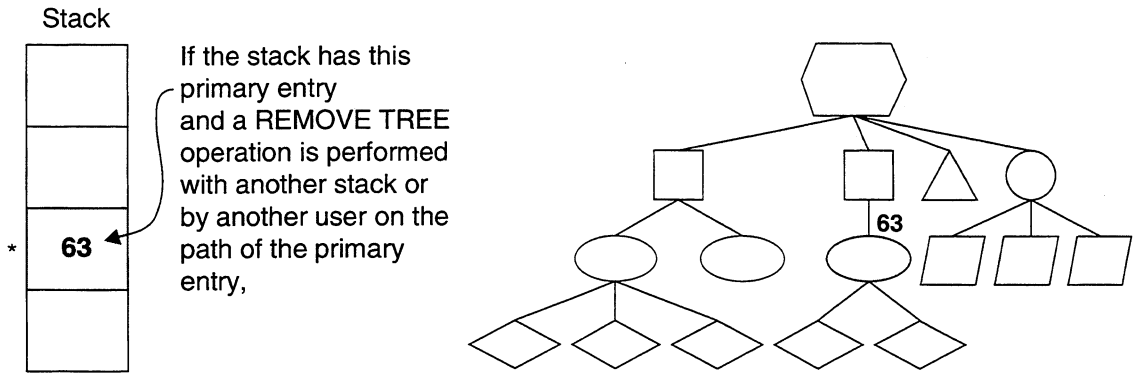


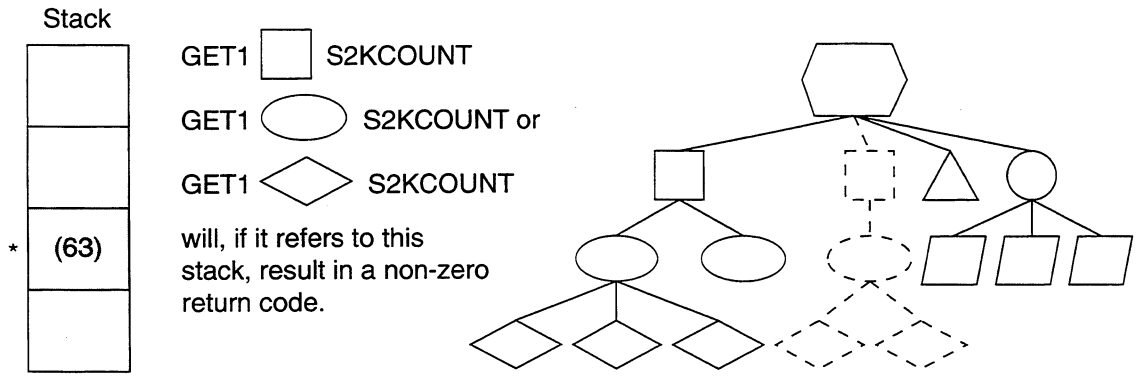



**Example 4**

If a REMOVE TREE operation from another stack or by another user removed the record at the level of the specified subschema record, the GET1 S2KCOUNT command results in return code 09, regardless of the value of S2KCOUNT.



## GET1 with a Where-Clause

---

**GET1 [/HOLD] [(***stack-number***)] *SSR* [*inclusion-list*] [*plex-where-clause*]**
**[USING 0]**

---

To retrieve a specific data record from the database, rather than navigating and testing each record in a logical path, you can code a GET1 command with a where-clause. A where-clause can cause retrieval of a single data record without any prior retrieval commands. (A GET1 command with a where-clause does not specify NEXT, FIRST, LAST, PRESENT, or S2KCOUNT.)

**Note**: When you need to retrieve only one record based on a single key condition with a unique value, use the GETP command, since it is more efficient than GET1 with a where-clause. When you need to retrieve more than one record based on the same where-clause, use the LOCATE command.

The where-clause states one or more conditions that a data record must meet in order to be retrieved. For GET1, the entire where-clause can include any of the operators and expressions that are discussed in Chapter 7, "Where-Clauses."

When using GET1 with a where-clause, take care in specifying the where-clause conditions. All records qualified by the where-clause are compiled by the software into a list of addresses for the qualified records. They are arranged in physical order, that is, the order in which data records are stored in the Data Table. If the first qualified record on the list is at the same level as the specified SSR, GET1 retrieves that first record. However, if the first qualified record is an ancestor of the specified SSR, GET1 retrieves the leftmost descendant record at the level of the specified SSR. (This leftmost descendant is the first in logical order.) Similarly, if the first qualified record on the list is a descendant of the specified SSR, GET1 retrieves the ancestor of the same level as the specified SSR. In any case, GET1 retrieves a single data record that meets the specified conditions, in essentially random order. Therefore, be sure that your where-clause conditions qualify the precise data record you want.

Although GET1 with a where-clause retrieves only one record, the number of records that were qualified through where-clause processing is indicated in the sixth field of the COMMBLOCK. (This field also shows the number of data record addresses in a Locate File.) Your program can check this field so that if there is more than one qualified record, you can take steps to ensure that the retrieved record (determined essentially through physical order) is the desired record.

If the sixth COMMBLOCK field contained the number of data record addresses in a Locate File, a GET1 command causes the sixth field to be reused if the command includes a where-clause and specifies a stack of the same number as the Locate File. But any other command given afterward specifying the same subscript (for the stack or Locate File) causes the Locate File information to be placed in the sixth field again.

The program's current password must have W-authority for each database item or schema record specified in a where-clause. If the password does not have W-authority, the GET1 command is ignored, and return code 46 results.

In a where-clause condition, the value of the variable to the right of the operator must be compatible with the database definition for the item in the condition. If it is not, return code 10 results.

If the where-clause consists of a single key condition, you can use the no-sort capability. See Appendix A for a discussion of this capability.

Be sure to review the items listed under **Considerations** on page 8-9 for any additional information.

The examples show different retrieval situations with a where-clause in a GET1 command. They are summarized here.

**Example 1:** GET1 with a single-condition where-clause at the level of the specified subschema record.

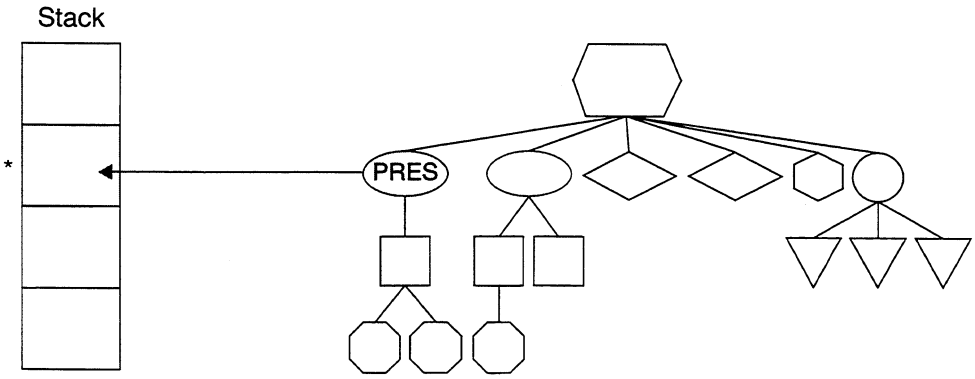
**Example 2:** GET1 that specifies a subschema record at a level below the level of the where-clause condition.

### Example 1

A GET1 command can specify a where-clause with a single condition at the same level as the specified SSR. If the database contains two ⬭ records with PRES for the position, the record with the lowest address is selected. The sixth field of the COMMBLOCK would indicate a count of 2.

```
WORKING-STORAGE SECTION.
      .
      .
01 JOB-TITLE        PICTURE X(4)
      .
      .
PROCEDURE DIVISION.
      .
      .
   MOVE 'PRES' TO JOB-TITLE

   GET1  ⬭   WHERE POSITION-TITLE EQ JOB TITLE.
```

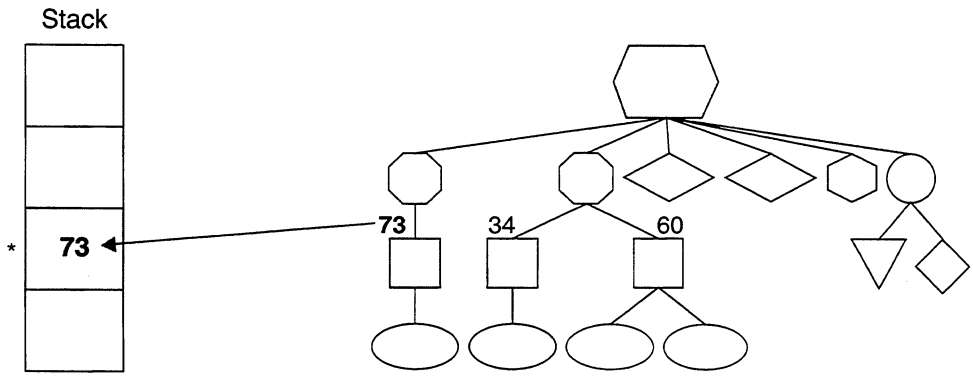


**Example 2**

A GET1 command can specify a subschema record at a level below the level of the where-clause condition. In this situation, the ☐ record with address 34 occurs earliest in physical order of the three ☐ descendants of the level 0 record. However, the record with address 73 is the first record in logical order when selection normalizes downward from the level of the where-clause condition. Even though there are three ☐ records in this logical entry, the count in the sixth COMMBLOCK field shows 1, for one qualified record at level 0.

```
WORKING-STORAGE SECTION.
      :
01 JOB-TITLE          PICTURE X(4)
      :
PROCEDURE DIVISION.
      :
   MOVE '4672' TO EMP-NO.

   GET1  ☐   WHERE EMPLOYEE-NUMBER EQ EMP-NO.
```



# GETA

The GETA command retrieves a data record that is an ancestor of a data record whose address is a stack entry. GETA establishes a secondary stack entry. The GETA command provides the following features:

- You can retrieve data records at any level in the path shown by the stack.
- You can specify any one of 16 stacks.
- With the USING 0 option, you can retrieve a data record address without retrieving data values.
- With an inclusion list, you can retrieve values for a specific subset of subschema items.
- The HOLD option in a Multi-User environment protects the integrity of retrieved data.

After a primary entry is established in the stack (and perhaps some secondary entries too), you can use the GETA command to create a secondary entry above any entry, up to level 0.

A GETA command specifies a stack to be used and names a subschema record associated with the data record to be retrieved. This data record can be the ancestor of any other data record whose address is a stack entry, from level 1 to the deepest level. GETA, then, can be used to retrieve a data record at a level above or below the primary.

When GETA is issued, data are retrieved into the subschema record, and the address of the retrieved data record becomes a secondary entry. No existing stack entries are affected.

## Format

```
GETA [/HOLD] [(stack-number)] SSR [inclusion-list]
                                  [USING 0]
```

*stack-number* is the number of the stack to be used in retrieval: a constant or a variable whose value is greater than or equal to zero and less than or equal to 15, that is, $0 \leq n \leq 15$. For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer. The default value of *stack-number* is 0.

*SSR* is the name of the subschema record for the data record to be retrieved.

*inclusion-list*

```
|item-list
|USING  |name-table
        |word-table
```

*item-list*

```
item-name  [, item-list]
```

*item-name* is the alias of an item in the specified subschema record.

*name-table* is the name of a table of component names or C-numbers for items to be included in the retrieval operation. For COBOL, the entry with the OCCURS clause must specify PICTURE X(32).

*word-table* is the name of a table of one-word indicators (1s and 0s), one indicator for each item in the specified subschema record. An item is included in the retrieval operation if its indicator in the table is 1. Conversely, an item is excluded from the operation if its indicator is 0. In each indicator, only the low-order bit is checked to see if it is 0 or 1. For COBOL, the entry with the OCCURS clause must specify COMP usage.

short form of the command and its equivalent

```
GETA SSR.     (short form, using the default)
GETA (0) SSR.
```

## Examples of Syntax

```
GETA SALARY.                        (short form, implies stack 0)

GETA (2) POSITIONS.                 (specifies stack (2))

GETA/HOLD (3) PAYROLL.              (specifies HOLD option for Multi-User)

GETA (1) POSITIONS TITLE, DIV.      (specifies item list)

GETA/HOLD (5) SKILLS USING 0.       (specifies address retrieval only)

GETA (10) POSITIONS USING
    POS-NAME-TABLE.                 (specifies name table)

GETA/HOLD POSITIONS USING
    POS-WORD-TABLE.                 (specifies word table)
```

## Examples of Usage

A sample sequence of PLEX commands, including GETA after GET1 and GET, is as follows:

```
WORKING-STORAGE SECTION.
      .
      .
      .
COMMBLOCK OF EMPLOYEE.
      .
      .
      .
   03  RETURN-KODE.
      .
      .
      .
SSR TRAINING OF EMPLOYEE.                  (SSR corresponds to TRAINING SR)
      .
      .
      .
SSR ENTRY-REC OF EMPLOYEE.                 (SSR corresponds to level 0 SR)
      .
      .
      .
SSR SALARY OF EMPLOYEE.                    (SSR corresponds to SALARY WITHIN
                                           POSITION SR)
      .
      .
      .
SSR PAYROLL OF EMPLOYEE.                   (SSR corresponds to MONTHLY
                                           PAYROLL ACCOUNTING SR)

PROCEDURE DIVISION

      .
      .
      .
   GET1 TRAINING NEXT.                     (retrieves next TRAINING data
                                           record; begins first loop)
      .
      .
      .
   GETA ENTRY-REC.                         (retrieves the ENTRY-REC
                                           ancestor of the TRAINING data
                                           record and ends first loop)
      .
      .
      .
   LOCATE PAYROLL WHERE . . .              (locates PAYROLL data records
                                           that meet the conditions of the
                                           where-clause)
      .
      .
      .
   GET PAYROLL NEXT.                       (retrieves the PAYROLL data record
                                           whose address comes next in the
                                           Locate File and begins second loop)
      .
      .
      .
   GETA SALARY.                            (retrieves the SALARY ancestor
                                           of the PAYROLL data record
                                           and ends second loop)
```

## Considerations

1. Check the return code after each GETA command is issued. The command's return code appears in the second field of the COMMBLOCK for the database to which the command refers.

2. The stack must have an entry below the level of the specified subschema record. This entry must be for a data record in the family of the specified subschema record. If the stack does not have such an entry, return code 07 results.

3. If the stack already has an entry for a data record associated with the subschema record specified in GETA, the data record is retrieved again, with return code 00. If the stack entry was the primary entry, it remains so.

4. A stack entry might be for the top of a tree removed by a REMOVE TREE command or moved by a MOVE TREE command and might be issued for another stack in the program or by another user in a Multi-User environment. If that stack entry is the highest one below the level of the subschema record specified in a GETA command, the command is ignored and return code 09 results. The eighth field in the COMMBLOCK indicates the level of the removed data record.

5. A GETA command can also be affected by a prior REMOVE TREE or MOVE TREE command if both commands are given for the same stack. In this situation, assume that a REMOVE TREE command removed a record whose address is in the specified stack, or assume that a MOVE TREE command moved a record whose address is in the specified stack and that no subsequent command changed the stack. If the stack entry for the removed or moved node is the highest entry below the level of the specified SSR, a check is made for the parent of the removed or moved node. If the parent is present, the GETA retrieval is performed; otherwise, return code 09 results, and the command is ignored.

6. For a discussion of inclusion lists, see **Inclusion List** on page 8-121. The USING 0 option is discussed in **USING 0 Option** on page 8-131.

7. If the MULTIPLE HOLDS option is enabled, your PLEX program can request any number of local holds on different records simultaneously. The MULTIPLE HOLDS option is disabled by default. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for details about setting the MULTIPLE HOLDS option. Local holds remain in effect until one of the following events occurs in the PLEX program:

   - a global hold is attempted for any database
   - a FRAME command is issued for any database
   - a synchpoint occurs
   - the program terminates
   - a DROP HOLD command occurs
   - the QUEUE command is issued.

   When any one of these occurs, all of the PLEX program's local holds on the database(s) are dropped. Also, an update command no longer causes a local hold to be dropped.

For complete discussion of local holds, see **HOLD Option** on page 80. See also **HOLD Option** on page 8-131 and **DROP HOLD** on page 8-144.

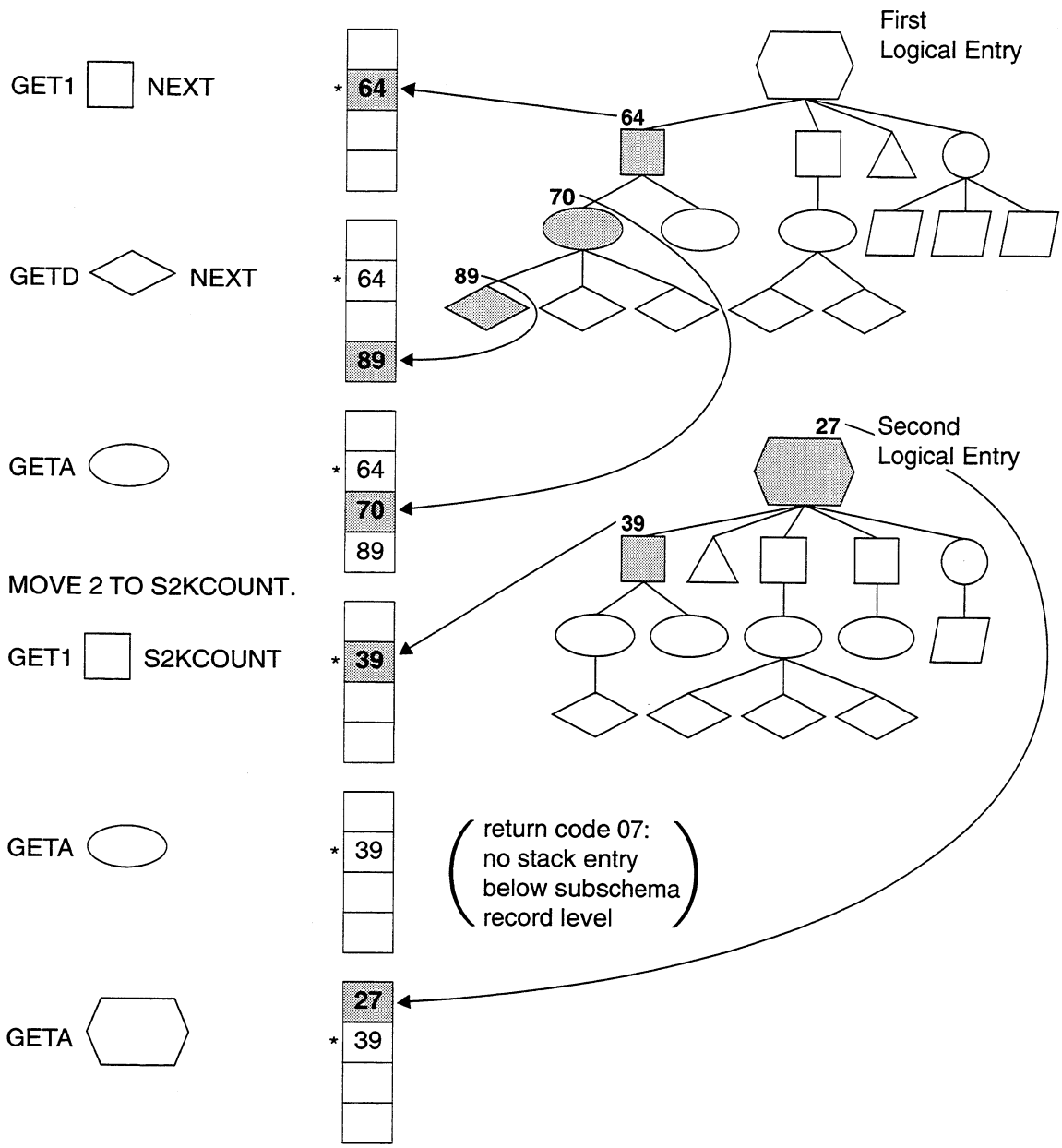
The examples show different situations of retrieving with GETA. They are summarized here.

**Example 1:** a command sequence including GETA.

**Example 2:** GETA after the primary node was removed.

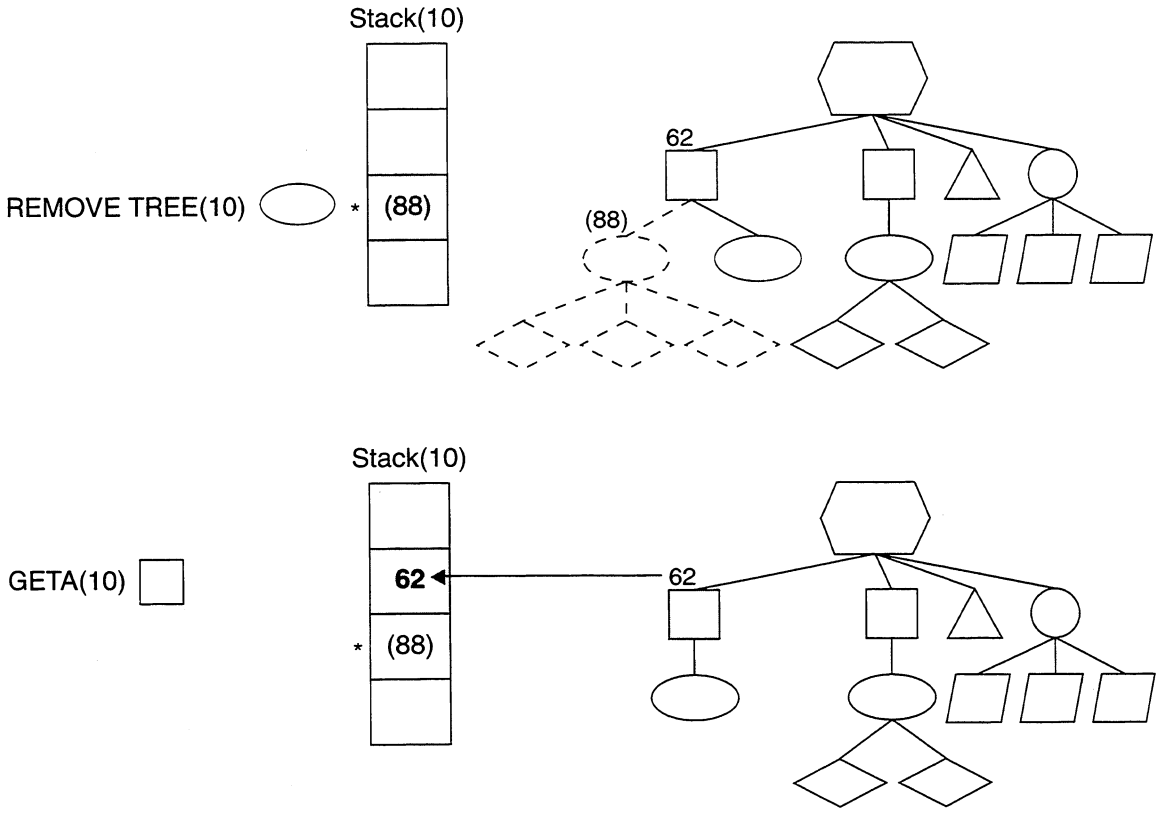
**Example 1**

GETA commands can be given to retrieve records above the level of a current stack entry.



### Example 2

Even if a REMOVE TREE command removes the primary node and its descendants, the program can still give a GETA command based on the removed primary. This is possible if the program did not give an intervening command that either established a new primary or referred to another stack.



# GETD

The GETD command retrieves a data record that is a descendant of the data record whose address is the primary stack entry. GETD establishes a secondary stack entry. The GETD command provides the following features:

- You can retrieve data records from more than one level below the primary stack entry.
- You can specify any one of 16 stacks.
- With the USING 0 option, you can retrieve a data record address without retrieving data values.
- With an inclusion list, you can retrieve values for specific (rather than all) items of a subschema record.
- You can retrieve a specific data descendant with NEXT, FIRST, LAST, PRESENT, or the specified S2KCOUNT variable.
- The HOLD option in a Multi-User environment protects the integrity of retrieved data.

A GETD command specifies the stack to be used and the subschema record associated with the data record to be retrieved. The entries in the chosen stack determine which of the data records associated with the subschema record can be retrieved. Retrievals with GETD apply to database levels below that of the primary stack entry. The domain of a GETD command is set by the deepest-level stack entry for a data record whose schema record is an ancestor of the schema record referred to by the command. See **GETD** on page 29 for an explanation of the concept of domain.

A successful GETD results in a secondary stack entry for the retrieved data record. All entries below this one are canceled. Entries between this one and the primary entry are also canceled if they are not for ancestors of the retrieved data record.

By giving a series of GETD commands, a program's retrieval operation can step through a chain of similar sibling data records in the domain. But the movement through the sibling chain need not be sequential. When GETD commands specify LAST and S2KCOUNT, certain data records in the chain are skipped.

## Format

```
GETD [/HOLD] [(stack-number)] SSR [inclusion-list] [NEXT]
                                  [USING 0]        [FIRST]
                                                   [LAST]
                                                   [PRESENT]
                                                   [S2KCOUNT]
```

*stack-number* is the number of the stack to be used in retrieval: a constant or a variable whose value is greater than or equal to zero and less than or equal to 15, that is, $0 \leq n \leq 15$. For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer. The default value of *stack-number* is 0.

*SSR* is the name of the subschema record for the data record to be retrieved.

*inclusion-list*

```
|USING  |item-list
       |name-table
       |word-table
```

*item-list*

```
|item-name  [, item-list]
```

*item-name* is the alias of an item in the specified subschema record.

*name-table* is the name of a table of component names, or C-numbers of items to be included in the retrieval operation. For COBOL, the entry with the OCCURS clause must specify PICTURE X(32).

*word-table* is the name of a table of one-word indicators (1s and 0s), one indicator for each item in the specified subschema record. An item is included in the retrieval operation if its indicator in the table is 1. Conversely, an item is excluded from the operation if its indicator is 0. In each indicator, only the low-order bit is checked to see whether it is 0 or 1. For COBOL, the entry with the OCCURS clause must have COMP usage.

short form of the command and its equivalents

```
GETD SSR.          (short form, using all defaults)
GETD (0) SSR.
GETD SSR NEXT.
GETD (0) SSR NEXT.
```

## Examples of Syntax

```
GETD (8) POSITIONS.
```
*(specifies stack and subschema record; NEXT is implied)*

```
GETD/HOLD (1) SKILLS.
```
*(specifies* HOLD *in Multi-User; NEXT is implied)*

```
GETD (9) PAYROLL NEXT.
```
*(specifies* NEXT *explicitly)*

```
GETD (3) EDUC S2KCOUNT.
```
*(specifies* S2KCOUNT *instead of* NEXT *with* S2KCOUNT *set prior to the command)*

```
GETD (0) POSITIONS TITLE,
     DIV LAST.
```
*(specifies an item list and* LAST*)*

```
GETD (2) SALARY USING 0.
```
*(specifies address retrieval only)*

```
GETD/HOLD (7) SALARY USING
     SAL-NAME-TABLE LAST.
```
*(specifies the name table called* SAL-NAME-TABLE *and the* HOLD *option)*

```
GETD (8) POSITIONS USING
     POS-WORD-TABLE LAST.
```
*(specifies the word table called* POS-WORD-TABLE*)*

## Example of Usage

A possible sequence of commands using GETD is as follows:

```
WORKING-STORAGE SECTION.
      .
      .
      .
COMMBLOCK OF EMPLOYEE.
      .
      .
      .
SSR ENTRY-REC OF EMPLOYEE.          (corresponds to the level 0
                                    schema record)
      .
      .
      .
SSR POSITION-REC OF EMPLOYEE.       (corresponds to the POSITION WITHIN
                                    COMPANY schema record)
      .
      .
      .
SSR EDUCATION OF EMPLOYEE.          (corresponds to the EDUCATION
                                    schema record)
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
   MAIN-LOOP.
   MOVE 3 TO S2KCOUNT.              (sets S2KCOUNT to 3 for the GET1 command;
                                    this variable has COMP usage and is part
                                    of the S2KDUM control block, which was
                                    added to the program by the PLEX processor)

   GET1 ENTRY-REC S2KCOUNT.         (retrieves the level 0 data record that
                                    is third in logical order after the one
                                    previously retrieved (S2KCOUNT=3))

   MOVE 2 TO S2KCOUNT.              (resets S2KCOUNT for the following GETD
                                    command)

   GETD POSITION-REC S2KCOUNT.      (retrieves the POSITION-REC data record
                                    that is (1) a descendant of the ENTRY
                                    data record, and (2) second in logical
                                    order after the POSITION-REC data record
(repeat GETD POSITION-REC loop      previously retrieved from the domain)
until end of domain reached)

      .
      .
      .
   GETD EDUCATION NEXT.             (retrieves the EDUCATION data record
                                    that is (1) a descendant of the
                                    ENTRY-REC data record, and (2) next
                                    in logical order after the EDUCATION
                                    data record previously retrieved
(repeat GETD EDUCATION loop         from the domain)
until end of domain reached)

(repeat MAIN LOOP)
```

## Considerations

1. A primary stack entry is required above the level of the data record to be retrieved.

2. Entries below the primary stack entry that are disjoint from the record to be retrieved do not influence the GETD processing.

3. After each GETD command, check the return code in the second field of the COMMBLOCK for the database to which the command refers. If the return code is 04, for example, due to the absence of a related stack entry or absence of a data record in the database, the stack is unchanged and available for use.

4. For a discussion of inclusion lists, see **Inclusion List** on page 8-121. The USING 0 option is discussed in **USING 0 Option** on page 8-131.

5. For all GETD commands, the retrieval of a record is based on two factors: the specified subschema record and the domain as defined by the stack entries. (See **GETD** on page 29 for a discussion of domain.) An attempt to retrieve beyond the domain established by the stack results in return code 04.

6. For a GETD command that specifies NEXT, S2KCOUNT, or LAST, retrieval movement within the domain starts at the record whose address is already the stack entry at the level of the specified subschema record, provided that record also corresponds to the subschema record. However, if a GETD FIRST command is issued or if the stack has no entry for a record associated with the subschema record, retrieval movement within the domain begins with the first record for the subschema record.

7. A GETD command can be affected by a prior REMOVE TREE or MOVE TREE command, with the following results:

   - If the REMOVE TREE or MOVE TREE command used another stack or was issued by another user, the following situations could occur:

     GETD NEXT or GETD S2KCOUNT results in return code 04 if the stack entry at the level of the specified subschema record was removed. If the stack has no entry at the level of the subschema record, but a higher-level stack entry was removed, return code 04 results.

     GETD FIRST or GETD LAST is executed as though the stack entry had never been removed if the removed stack entry is at the level of the specified subschema record. If a higher-level stack entry was removed, return code 03 results.

     GETD PRESENT results in return code 09 if the stack entry at or below the level of the specified subschema record was removed.

     In these situations, the eighth field of the COMMBLOCK indicates the level of the removed or moved node.

- If the REMOVE TREE or MOVE TREE command was given for the stack to be used for GETD and if no subsequent command changed the stack, the following situations could occur:

  - The removed stack entry is at the level of the specified subschema record. Retrieval takes place in the domain unless a REMOVE TREE or MOVE TREE command using another stack or issued by another user removed or moved the right sibling or the parent.

  - The removed or moved node is above the level of the subschema record and is in the path of a record for the subschema record. Retrieval takes place in the domain unless the primary and all of its descendants were also removed or moved. (See Example 3 in **GETD NEXT** on page 8-60.)

If retrieval is impossible because a stack entry was removed, return code 09 results, and the eighth field of the COMMBLOCK indicates the level of the removed or moved node.

8. With GETD, the value you assign to S2KCOUNT must be 0 or any integer greater than 0; otherwise, return code 31 results. If S2KCOUNT equals 0, the command is equivalent to GETD PRESENT. A value of 1 is equivalent to GETD NEXT.

9. If the MULTIPLE HOLDS option is enabled, your PLEX program can request any number of local holds on different records simultaneously. The MULTIPLE HOLDS option is disabled by default. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for details about setting the MULTIPLE HOLDS option. Local holds remain in effect until one of the following events occurs in the PLEX program:

   - a global hold is attempted for any database
   - a FRAME command is issued for any database
   - a synchpoint occurs
   - the program terminates
   - a DROP HOLD command occurs
   - the QUEUE command is issued.

   When any of these events occurs, all of the PLEX program's local holds on the database(s) are dropped. Also, an update command no longer causes a local hold to be dropped.

   For a complete discussion of local holds, see **HOLD Option** on page 80. See also **HOLD Option** on page 8-131 and **DROP HOLD** on page 8-144.

## GETD NEXT

```
GETD [/HOLD] [(stack-number)] SSR [inclusion-list] [NEXT]
                                  [USING 0]
```

A GETD command specifying NEXT moves through the domain's set of similar descendant data records, one by one, in logical order.

When FIRST, LAST, PRESENT, or S2KCOUNT are not specified, the GETD command is, by default, GETD NEXT. NEXT retrieves the next data record (in the domain) that is similar to a data record whose entry is already in the stack, in logical order.

If the stack does not yet contain an entry for a similar data record, GETD NEXT retrieves the first data record (in the domain) in the sibling chain of data records associated with the specified subschema record.

Be sure to review the items listed under **Considerations** on page 8-58 for any additional information. In particular, see item 7 regarding the effect of removing nodes from or moving nodes in a database.

The examples show different situations of retrieval with GETD NEXT. They are summarized here.

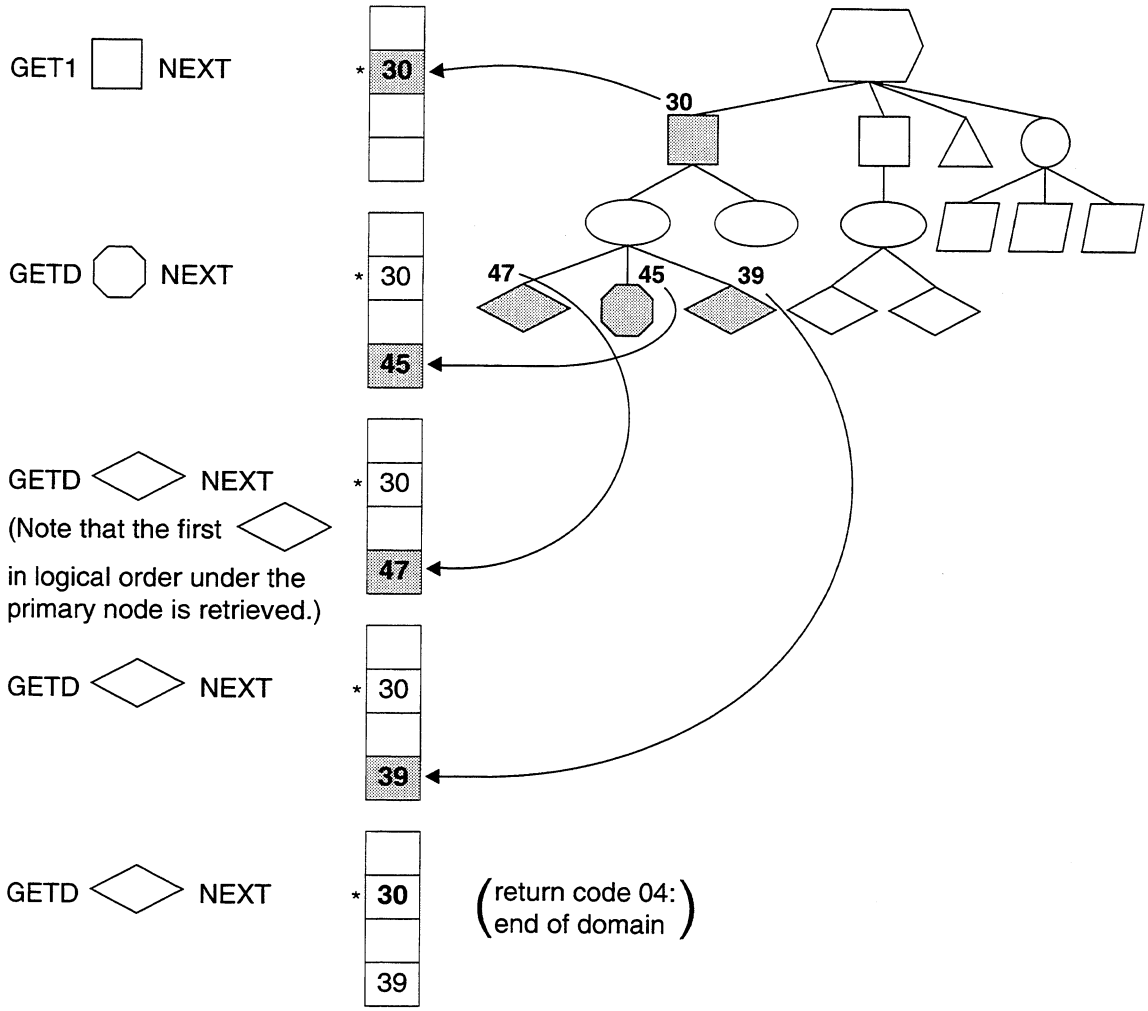
**Example 1:** GETD NEXT commands for different subschema records.

**Example 2:** GETD NEXT after a REMOVE TREE command for the same stack.

**Example 3:** GETD NEXT after REMOVE TREE at a higher level.

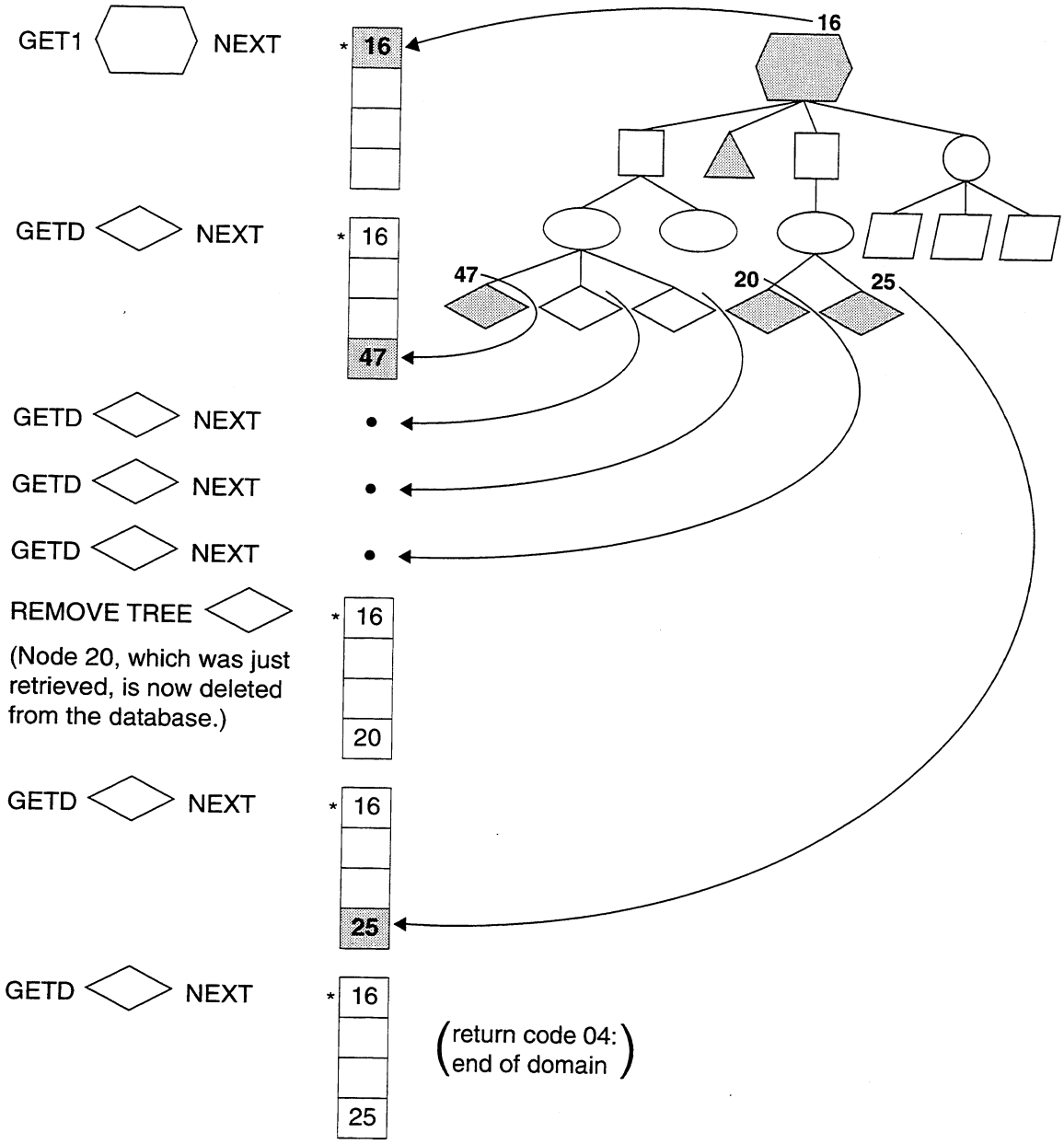
**Example 1**

This command sequence shows that the domain for GETD is determined at the level of retrieval only by a previously retrieved node for the same subschema record.



**Example 2**

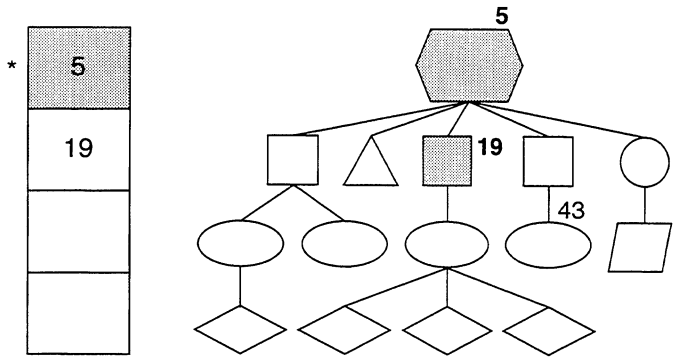
In this command sequence, a REMOVE TREE command deletes one of the retrieved nodes. But a GETD command can still be issued to retrieve the next similar sibling.



### Example 3

A REMOVE TREE command deletes a subtree, including a level 1 node whose address is a stack entry. A GETD NEXT command can still be given to retrieve a level 2 record in the domain.

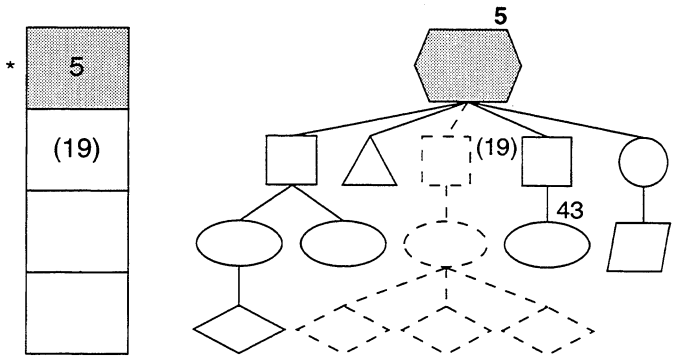
The stack already has these entries.




The command

REMOVE TREE ☐

is given for the same stack.

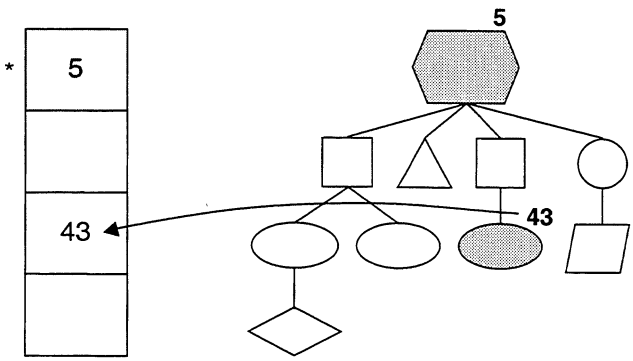



After the subtree is removed,

GETD ◯ NEXT

retrieves the ◯ record

that comes next in the domain.



## GETD FIRST

```
GETD [/HOLD] [(stack-number)] SSR [inclusion-list] [FIRST]
                                  [USING 0]
```

A GETD command specifying FIRST retrieves the first data record in a logical series of similar descendant data records in the domain. In other words, with FIRST, GETD retrieves the domain's first data record (in logical order) associated with the specified subschema record.

Be sure to review the considerations under **Considerations** on page 8-58 for any additional information. In particular, see item 7 for the effect of removing nodes from or moving nodes in a database.

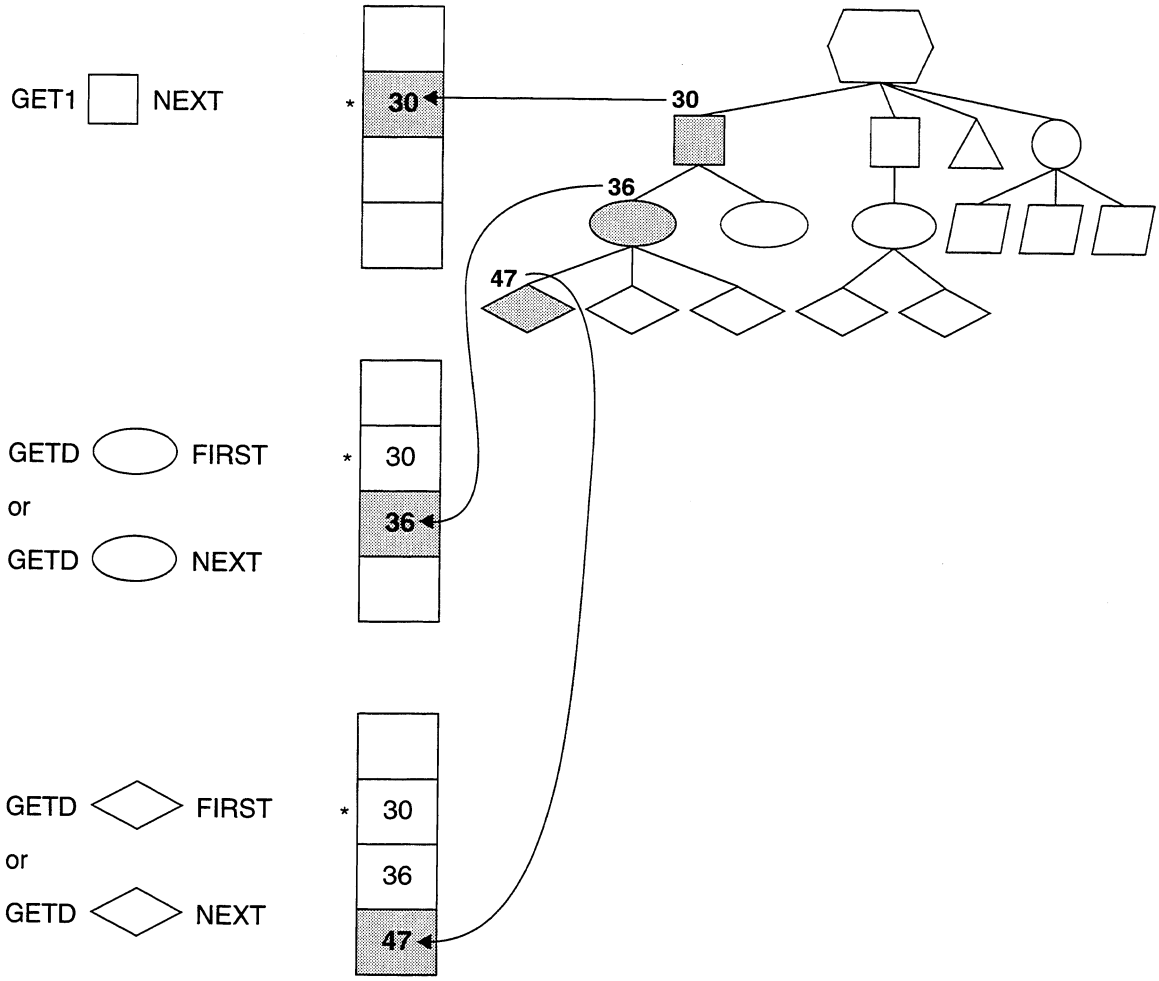
The examples show retrievals with GETD FIRST. They are summarized here.

**Example 1:** GETD FIRST commands for different subschema records.

**Example 2:** GETD FIRST after a REMOVE TREE for the same stack.

### Example 1

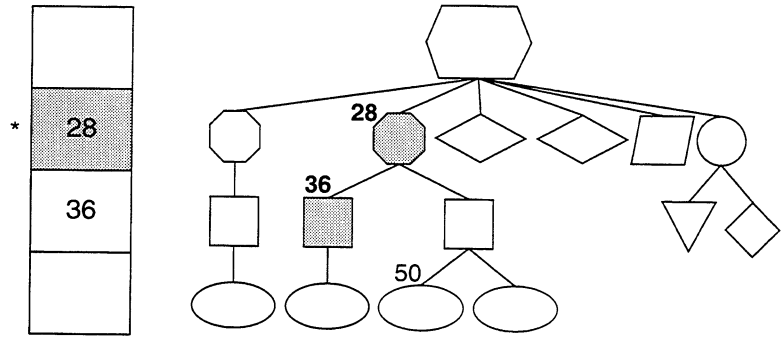
This command sequence shows that GETD FIRST retrieves the first record for the SSR in the domain. Also, the first GETD NEXT command for the domain is equivalent to GETD FIRST.



**Example 2**

In this command sequence, a REMOVE TREE command deletes a subtree of the logical entry. A GETD FIRST command can still be given to retrieve the record that is now the first data record for the SSR in the domain.

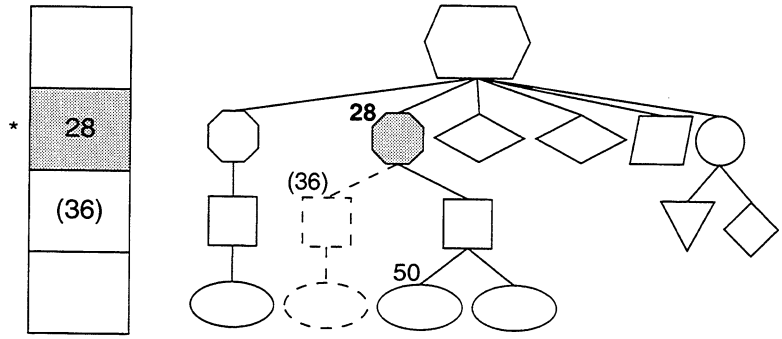
The stack already has these entries.




The command

REMOVE TREE 

is given for the same stack.



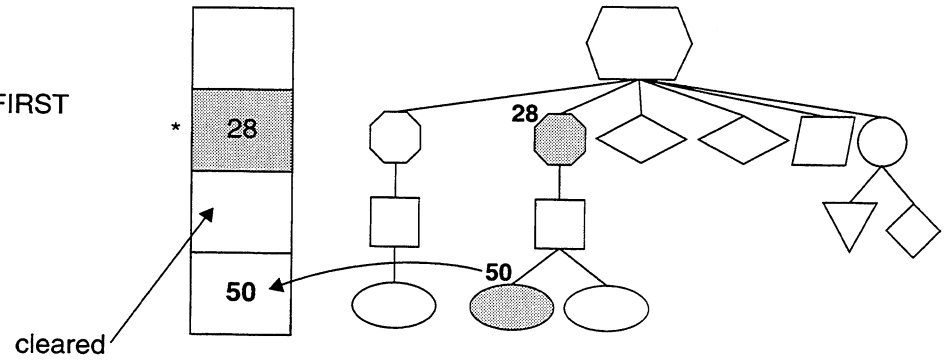

After the subtree is removed,

GETD  FIRST

retrieves the

 record

currently first in the domain.



## GETD LAST

```
GETD [/HOLD] [(stack-number)] SSR [inclusion-list] [LAST]
                                  [USING 0]
```

A GETD command specifying LAST retrieves the last data record in a logical series of similar descendant data records in the domain. In other words, with LAST, GETD retrieves the domain's last data record (in logical order) associated with the specified subschema record.

If the stack contains an entry at the SSR level and GETD LAST retrieves the record for that entry again as the last within its domain, return code 50 results.

Be sure to review the considerations listed under **Considerations** on page 8-58 for any additional information. In particular, see item 7 for the effect of removing nodes from or moving nodes in a database.

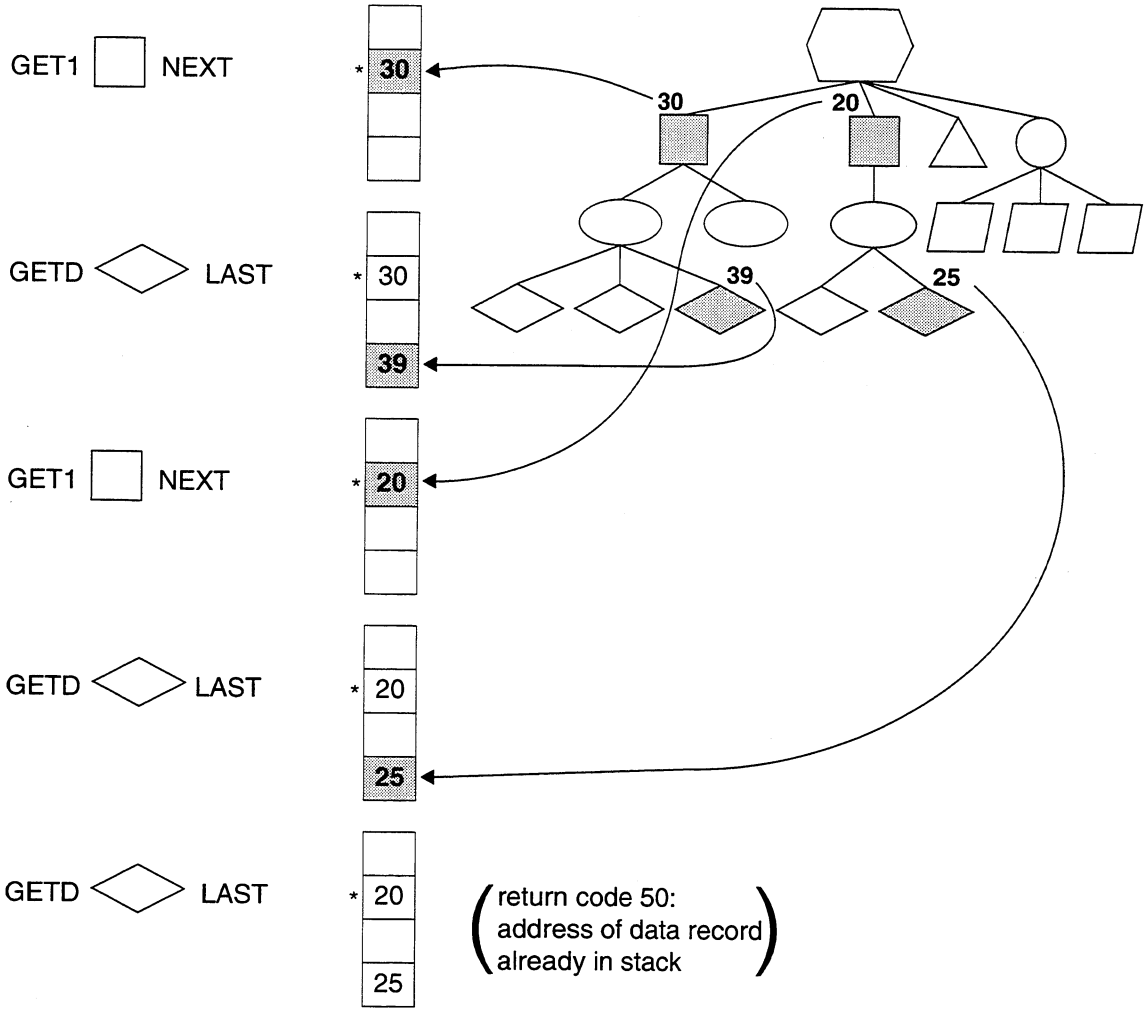
The examples show retrievals with GETD LAST. They are summarized here.

**Example 1:** GETD LAST commands for different domains; GETD LAST can retrieve the same record again.

**Example 2:** GETD LAST after a REMOVE TREE for the same stack.

**Example 1**

This command sequence shows that GETD LAST retrieves the last record for the SSR in the domain. In addition, if GETD LAST retrieves the same record again, return code 50 results.



### Example 2

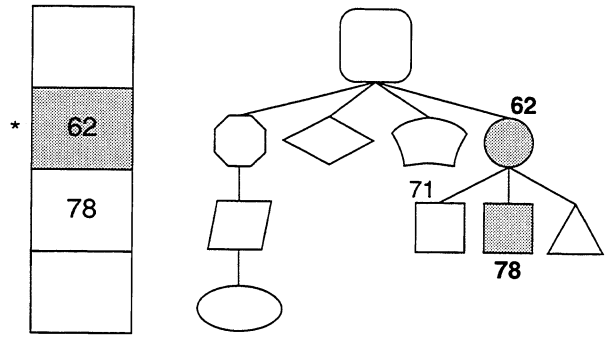
In this command sequence, a REMOVE TREE command deletes a record. Your program can issue a GETD LAST command to retrieve the record that is now the last data record for the SSR in the domain.

The commands

GET1 ◯ NEXT and

GETD □ LAST

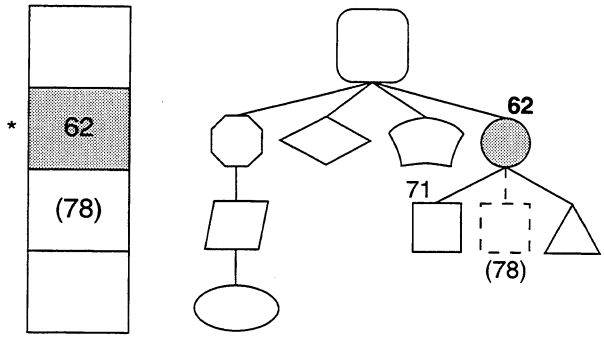
have established these stack entries.




The command

REMOVE TREE □

is given for the same stack.

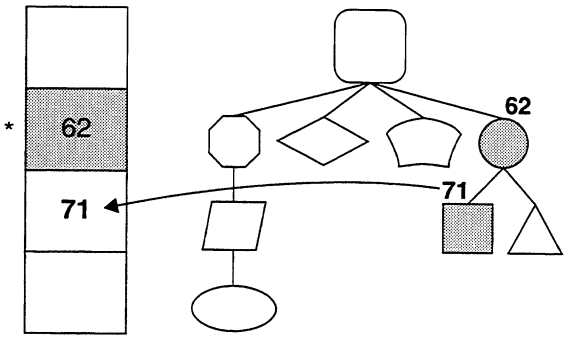



After that record is removed,

GETD □ LAST

retrieves the □ record

currently last in the domain.



## GETD PRESENT

```
GETD [/HOLD] [(stack-number)] SSR [inclusion-list] [PRESENT]
                                                   [USING 0]
```

A GETD command specifying PRESENT selects a record at the level of the specified SSR, establishes a secondary stack entry for the retrieved record, and clears the stack below that entry. The level of the specified SSR must be below the level of a primary stack entry and at or above the level of an already existing secondary stack entry. If there is no existing secondary stack entry or if the record to be retrieved is not in the path of a lower secondary stack entry, return code 03 is issued.

You can use the PRESENT option to specify an alternate SSR for the same schema record as the SSR in an original GETD command. For example, a program can first issue a GETD command specifying an SSR with some selection criteria. For each data record that meets the selection criteria, the program can then issue a GETD PRESENT command to retrieve the data record, this time retrieving a large number of items from the data record by specifying an SSR containing more items than the originally specified SSR.

Be sure to review the considerations listed under **Considerations** on page 8-58 for any additional information.

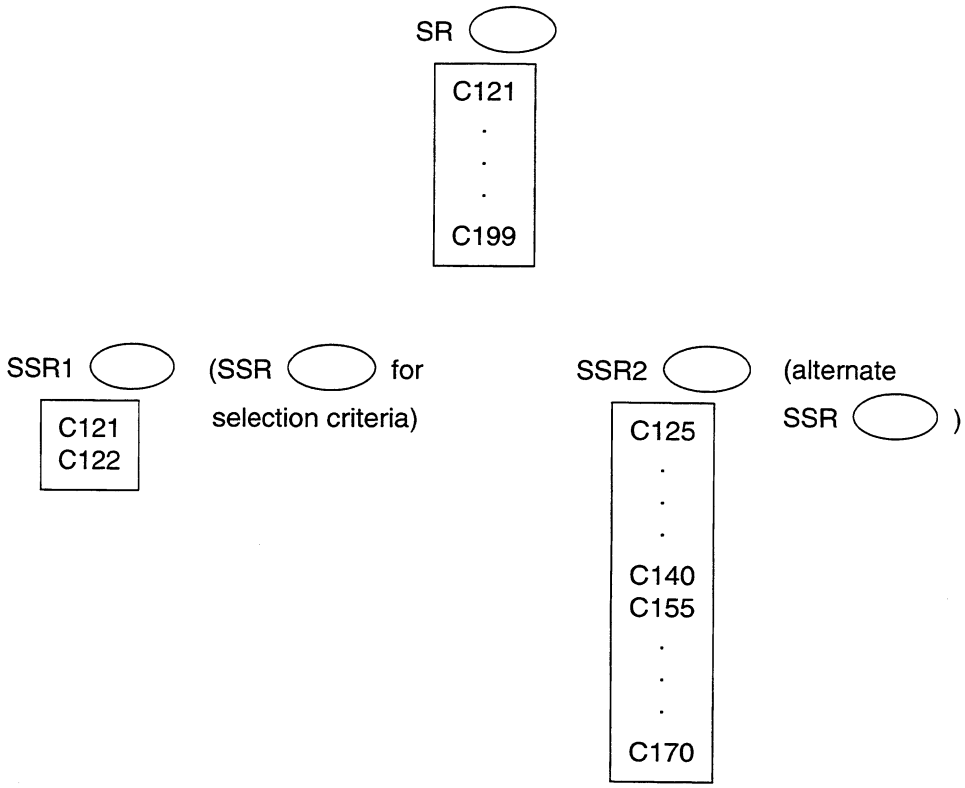
The following examples show retrievals with GETD PRESENT. They are summarized here.

**Example 1:** Relationship of alternate SSRs to GETD PRESENT.

**Example 2:** GETD PRESENT and referring to levels with regard to the stack entries.

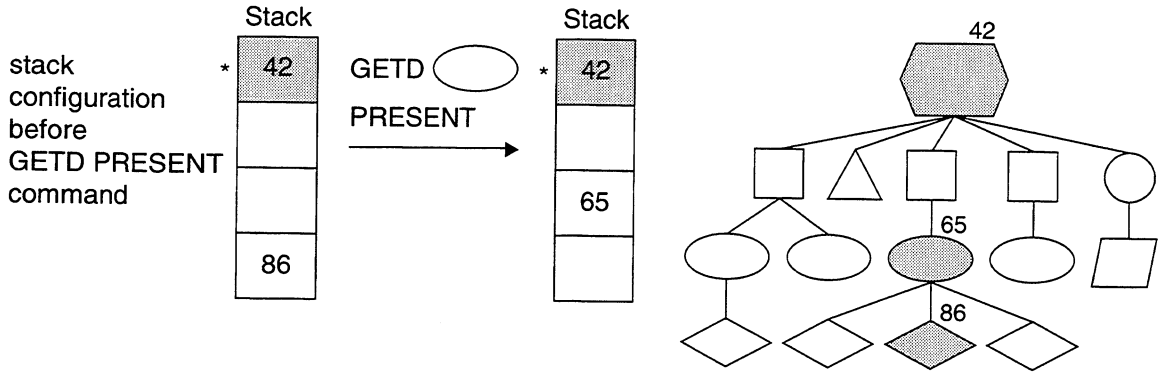
**Example 1**

This illustration shows the relationship of alternate SSRs to a GETD PRESENT command.



**Example 2**

GETD PRESENT must refer to a level below the primary stack entry and at or above an existing secondary stack entry.



## GETD S2KCOUNT

```
GETD [/HOLD] [(stack-number)] SSR [inclusion-list] [S2KCOUNT]
                                                   [USING 0]
```

S2KCOUNT can be used in a GETD command to provide a selectivity not possible with NEXT, FIRST, and LAST.

S2KCOUNT has no default value. Before a program can use S2KCOUNT in a GETD command, it must move a value to the variable S2KCOUNT, which resides in the S2KDUM control block. With GETD, S2KCOUNT can be assigned the value of zero or any integer greater than zero.

GETD S2KCOUNT retrieves a data record that occurs later (in logical order) in the domain than a similar data record whose entry is already in the stack. The value of S2KCOUNT tells SYSTEM 2000 software the number of similar sibling data records to move through, following the logical order, to retrieve a data record associated with the specified subschema record.

The following equivalents of the GETD S2KCOUNT command depend on the value assigned to S2KCOUNT:

| When | GETD S2KCOUNT means |
|---|---|
| S2KCOUNT = 1 | GETD NEXT |
| S2KCOUNT = 0 | GETD PRESENT |

Be sure to review the considerations listed under **Considerations** on page 8-58 for any additional information. In particular, see item 7 for the effect of removing nodes from or moving nodes in a database. Also, review **GETD PRESENT** on page 8-70 and **GETD NEXT** on page 8-60 if S2KCOUNT equals 0 or 1, respectively.

The examples show different situations of retrieval with GETD S2KCOUNT. They are summarized here.

**Example 1:** GETD S2KCOUNT with S2KCOUNT = 0.

**Example 2:** GETD S2KCOUNT with S2KCOUNT = 1.

**Example 3:** GETD S2KCOUNT with S2KCOUNT = 2.

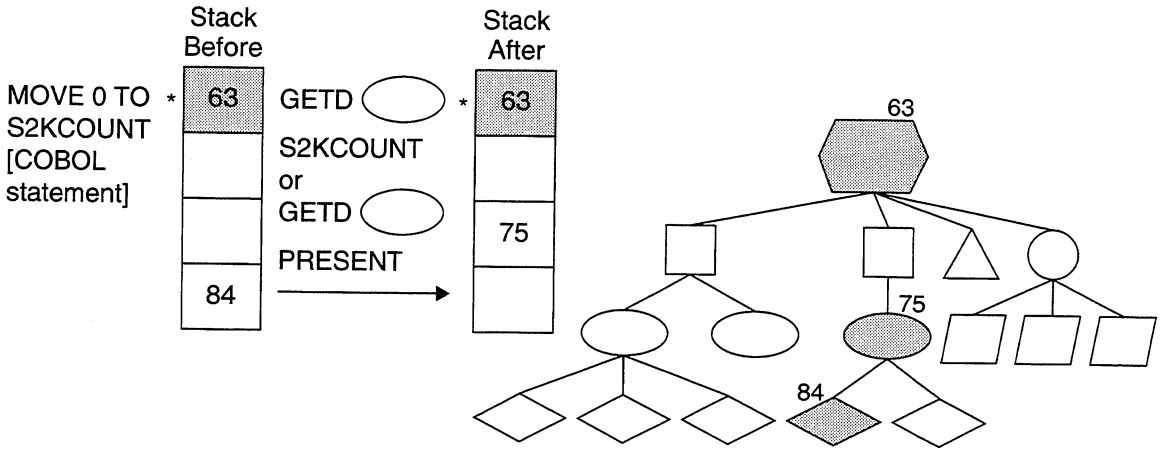
**Example 4:** GETD S2KCOUNT after REMOVE TREE for the same stack.

**Example 5:** GETD S2KCOUNT with S2KCOUNT = 3.

**Example 6:** GETD S2KCOUNT to retrieve pairs of records.

**Example 1**

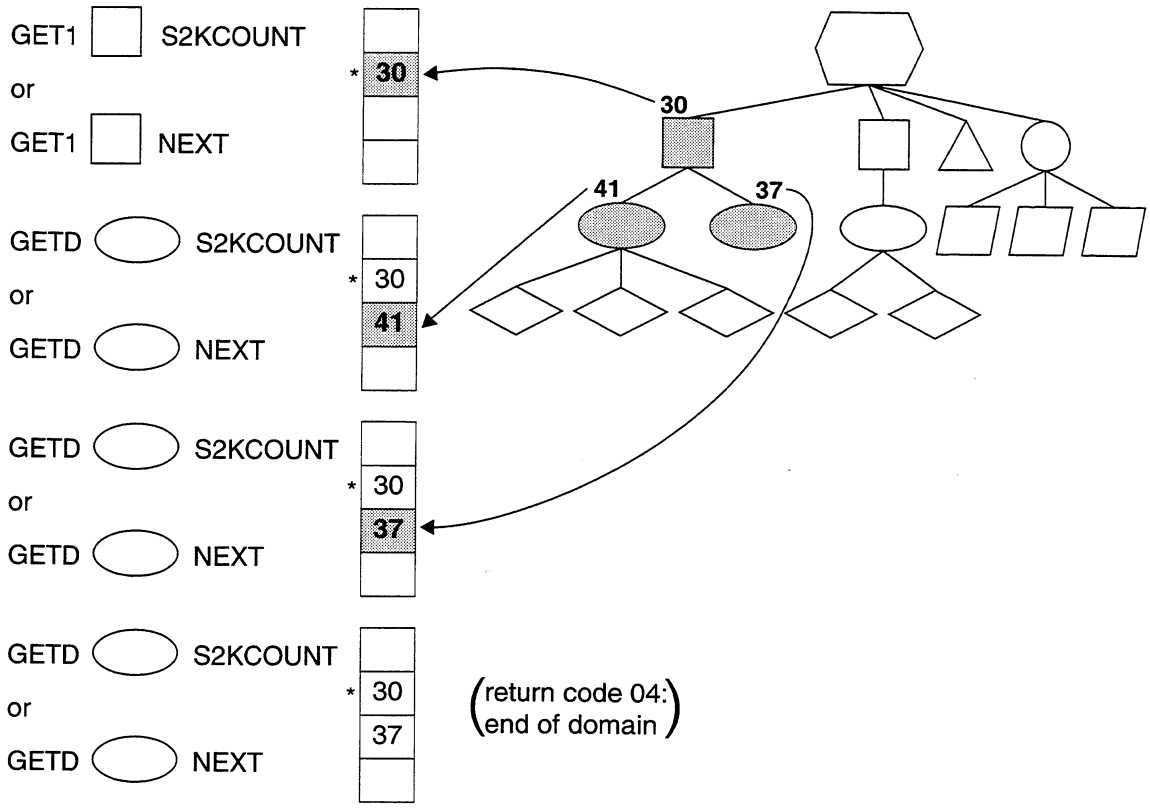
If S2KCOUNT equals 0, the GETD S2KCOUNT command gives the same results as a GETD PRESENT command.



### Example 2

If S2KCOUNT equals 1, the GETD S2KCOUNT command gives the same results as a GETD NEXT command.

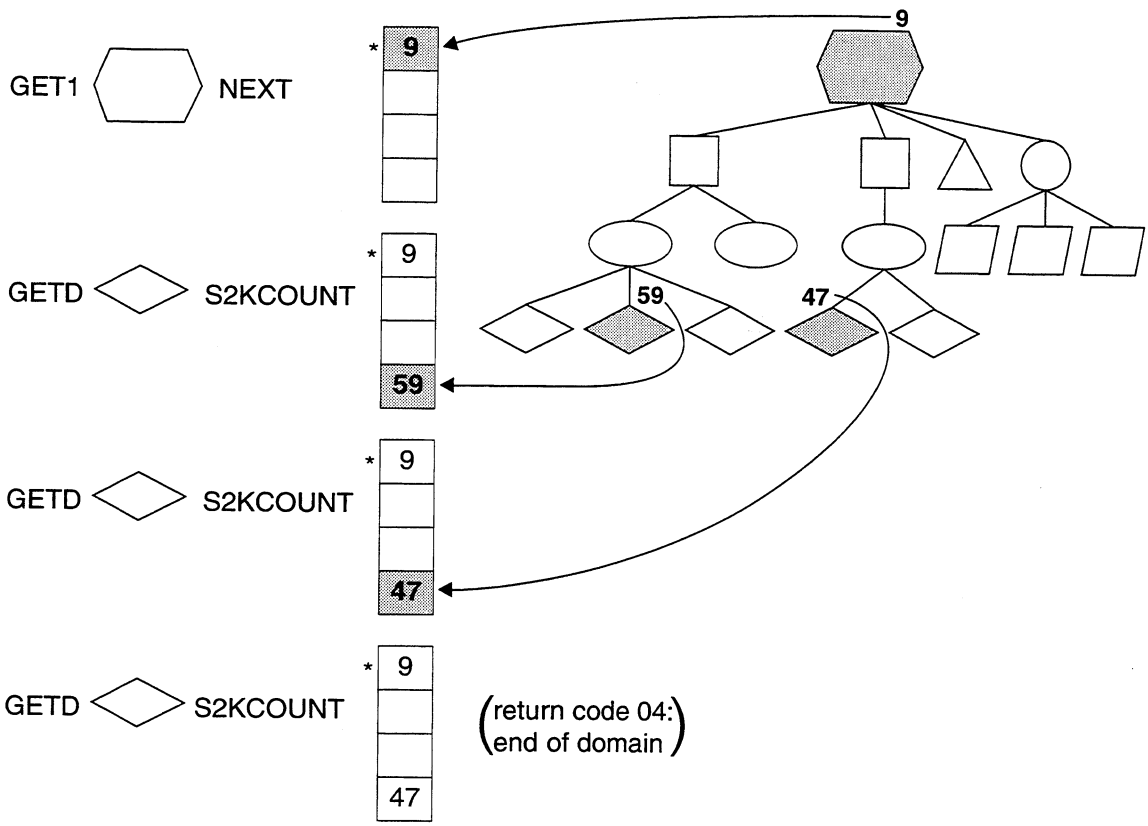
MOVE 1 TO S2KCOUNT. [COBOL Statement]



**Example 3**

If S2KCOUNT equals 2, GETD S2KCOUNT retrieves every other record for the SSR in the domain.

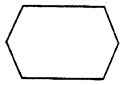
MOVE 2 TO S2KCOUNT. [COBOL Statement]

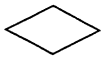


### Example 4

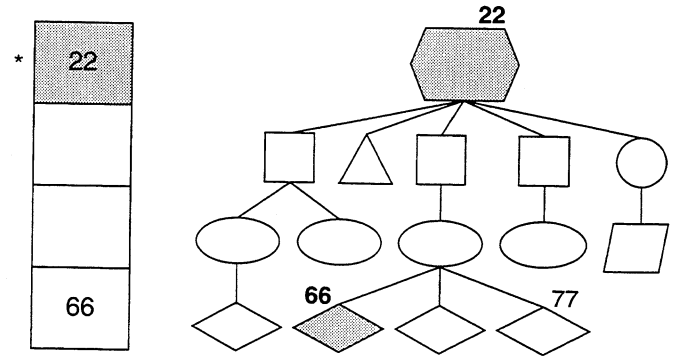
In this command sequence, a REMOVE TREE command deletes a record. Your program can still issue a GETD S2KCOUNT command to retrieve the second record after the one that was removed.

The commands

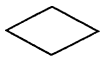
GET1  NEXT and

GETD  S2KCOUNT

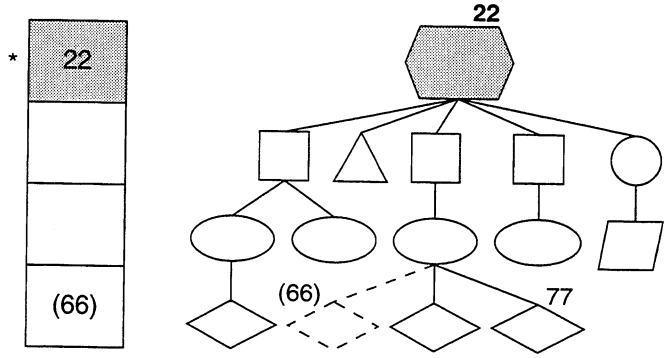
(with S2KCOUNT = 2) have established these stack entries.



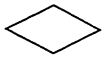

The command

REMOVE TREE 

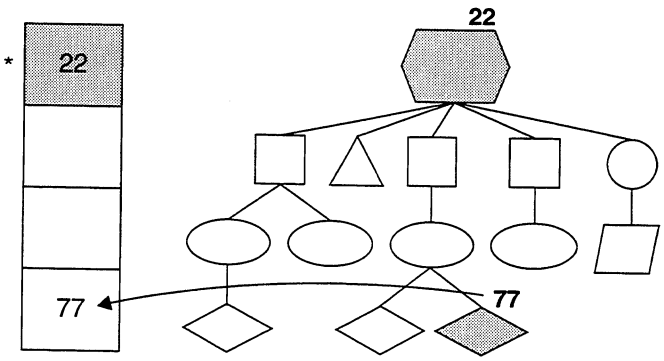
is given for the same stack.




The command

GETD  S2KCOUNT

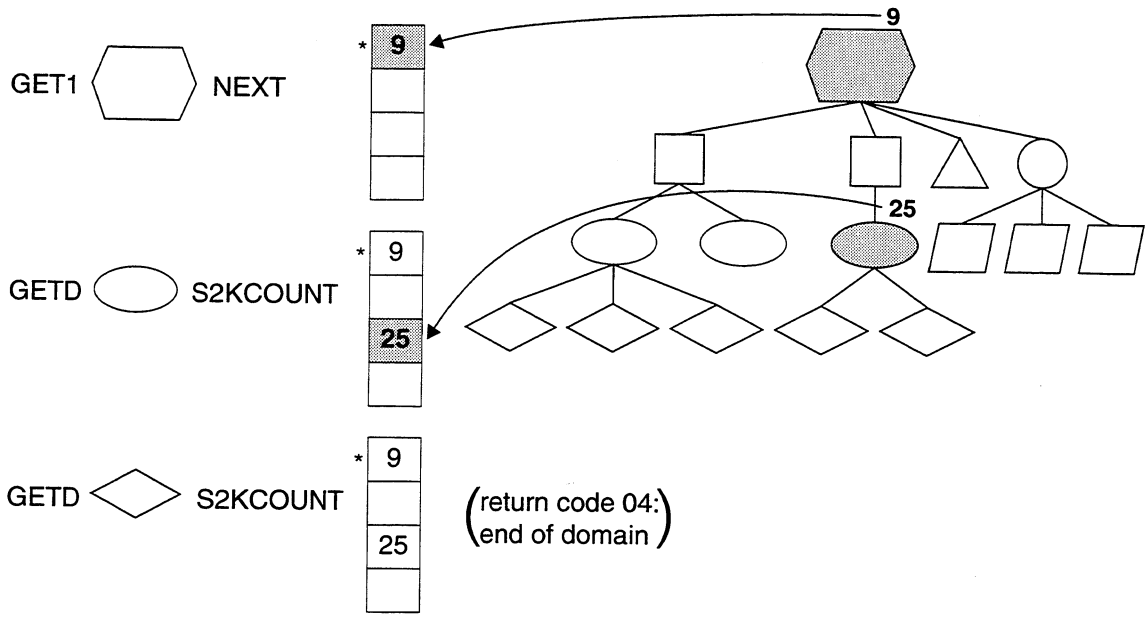
then retrieves the second record after the removed record.



**Example 5**

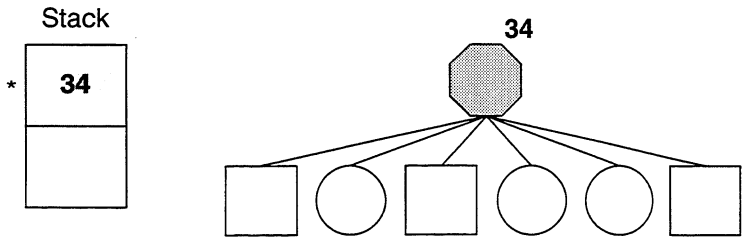
In this example, with S2KCOUNT equal to 3, the first GETD S2KCOUNT command succeeds because there are three ⬭ records in the domain. But the second GETD S2KCOUNT command fails because the domain includes only two ◇ records.

MOVE 3 TO S2KCOUNT. [COBOL Statement]



### Example 6

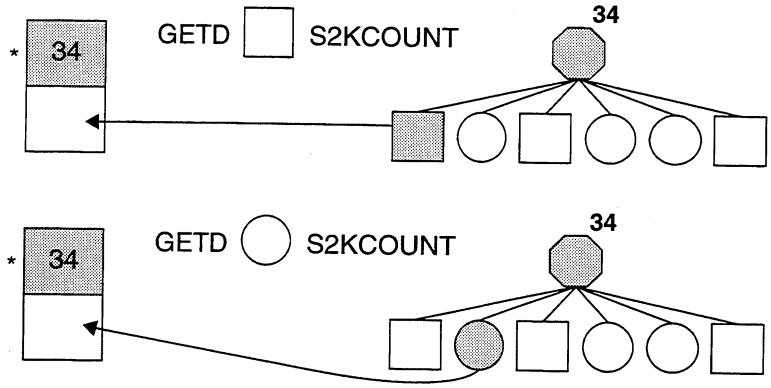
Records at level 1 are grouped under the parent in pairs ( □ and ○ in each pair). For this application, the records must be retrieved in pairs, and only paired records should be retrieved. A paragraph named in a PERFORM UNTIL statement retrieves a pair of records and increments S2KCOUNT. Using the same stack, dissimilar records are retrieved alternately, so that the retrieval of a □ record, for example, does not limit the domain of retrieval of ○ records. In this situation, therefore, the counting of positions for S2KCOUNT must start at the beginning of the sibling chain for each retrieval. In order to move through the sibling chain, S2KCOUNT must be incremented for each pair of retrievals. The results of executing the paragraph are shown on the next page.



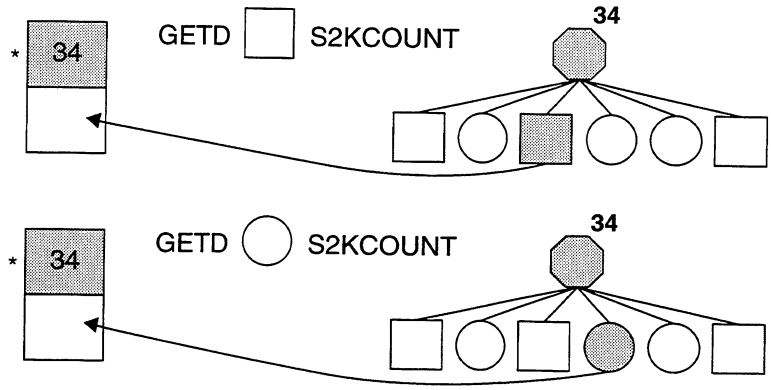

```
PERFORM GETD-S2KCOUNT UNTIL RETURN-KODE = 4.
:
GETD-S2KCOUNT.
    ADD 1 TO S2KCOUNT.
    GETD □ S2KCOUNT.
    IF RETURN-KODE NOT = 4
        GETD ○ S2KCOUNT;
        (Write retrieved records to Report File.)
```

**Example 6 (continued)**

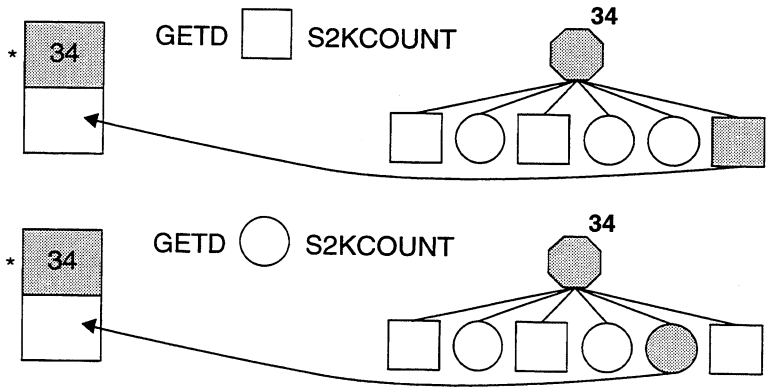
First time through loop
(S2KCOUNT = 1)




Second time through loop
(S2KCOUNT = 2)




Last time through loop
(S2KCOUNT = 3)



# LOCATE

The LOCATE command selects the addresses of data records qualified by where-clause conditions. The addresses are placed in a specified Locate File. The LOCATE command offers the following features:

- Retrieval with LOCATE isolates a subset of the data structure, using the powerful PLEX where-clause processor.
- You can retrieve data records at any level in the data structure.
- You can specify any one of up to 16 Locate Files.

A LOCATE command begins a process of subset retrieval, that is, retrieval based on where-clause conditions for qualifying data records. The command specifies the Locate File to be used, a subschema record associated with the data records whose addresses will be selected, and a where-clause stating the conditions for qualification. Those conditions can be as broad or as specific as necessary. See Chapter 7, "Where-Clauses,"for more information about where-clause conditions.

When the LOCATE command is executed, the where-clause processor finds all the data records associated with the specified subschema record that meet the conditions of the where-clause. The software places the addresses of all data records qualified by the where-clause in physical order on an internal list and compares them to the level of the specified subschema record. If the qualified records are at the same level as the specified subschema record, all the qualified addresses are copied into the Locate File (still in physical order) and become selected records. If the qualified records are at a different level than the specified SSR, they are normalized, and the normalized addresses are placed in the Locate File, in physical order, and become the selected records. In either case, the LOCATE command retrieves no actual data, only addresses.

The data record addresses held in the Locate File represent a selected subset of the data structure. A PLEX program can then retrieve the data records whose addresses are in the Locate File using a series of GET commands. This method is much more efficient when multiple data records will meet the conditions in the where-clause; the program need not give a long series of retrieval commands to qualify each one within the logical data structure.

Once you establish the contents of the Locate File, you can rearrange the addresses with an ORDER BY command; then, with a series of GET commands you can retrieve the records one-by-one into a stack, in the desired order. For more introductory information, see **Introduction to Retrieval** on page 11.

## Format

```
LOCATE [(Locate-File-number)] SSR plex-where-clause.
```

*Locate-File-number* is the number of the Locate File that will hold the addresses of located data records: a constant or a variable whose value is equal to or greater than zero and equal to or less than the number of extra Locate Files requested in the START S2K

command. The default value is zero. If no number is specified on the START S2K command, Locate File 0 is the only one available. For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*SSR* is the name of the subschema record for the data records to be located.

*plex-where-clause* is the statement of the conditions of retrieval. See Chapter 7, "Where-Clauses" for a detailed discussion of the where-clause.

short form of the command and its equivalent:

**LOCATE** *SSR plex-where-clause.* *(short form, using the default)*

**LOCATE (0)** *SSR plex-where-clause*

## Examples of Syntax

```
LOCATE PAYROLL WHERE MONTH EQ                    (implies Locate File 0)
  THIS-MONTH.

LOCATE (6) POSITION-REC WHERE CO HAS             (specifies Locate File 6)
  SKILL EQ SKL-TYPE.
```

## Example of Usage

Possible sequence of commands, including LOCATE.

```
   WORKING-STORAGE SECTION.
         :
         :
   01  PAY MONTH. . .
         :
         :
   COMMBLOCK OF EMPLOYEE.
   01  EMPLOYEE.
         :
         :
       03  RETURN-KODE.                  (field 2)
         :
         :
       03  NUMBER-OF-ADDRESSES.          (field 6)
         :
         :
   SSR PAYROLL OF EMPLOYEE.
   01  PAYROLL.
       03  MONTH . . .
           EQ PAYROLL MONTH.
         :
         :
   PROCEDURE DIVISION.
         :
         :
      START S2K (6).                      (first PLEX command; requests six
                                          Locate Files besides Locate File 0;
                                          check return code in S2KRTC)

      OPEN EMPLOYEE.                      (opens database; check RETURN-KODE)
         :
         :
      MOVE 'FEBRUARY' TO PAYMONTH.

      LOCATE (6) PAYROLL WHERE MONTH EQ PAYMONTH.
                                          (Check RETURN-KODE and
                                          NUMBER-OF-ADDRESSES before
                                          proceeding with an ORDER BY or
                                          GET command; NUMBER-OF-ADDRESSES
                                          might show that no data records
                                          were located.)
```

## Considerations

1. Check the return code after the LOCATE command is given. The command's return code is placed in the second field of the COMMBLOCK for the database to which the command refers. If the return code is zero, the LOCATE command was processed without error. However, this does not ensure that any data record addresses were selected into the Locate File (see next consideration).

2. After the LOCATE command is given, the sixth field of the COMMBLOCK should be checked. This field tells the number of data record addresses selected by the command. If it is zero, either no data records met the conditions of the where-clause, or no data records were selected into the Locate File because of normalization. If the sixth field is zero, subsequent GET commands produce return code 06.

3. When your program needs to retrieve one data record by stating a unique condition in a where-clause, use the GET1 or GETP command, rather than the LOCATE command, since GET1 and GETP always cause the retrieval of a single data record. LOCATE never retrieves any data.

4. The subschema record specified in a GET command can be different from the one specified in the associated LOCATE command, as long as both subschema records correspond to the same schema record. However, if they do not, the GET command is ignored and return code 18 results.

5. In a LOCATE command, the specified SSR must correspond to a schema record in the same path indicated by the schema records and items specified in the where-clause conditions. If the records and items specified in the conditions do not fall in one path, or if the specified SSR is not in the indicated path, the LOCATE command is ignored, and return code 58 results.

6. After a LOCATE command is executed, a data record whose address is in the Locate File might be deleted from the database by a REMOVE TREE operation or moved by a MOVE TREE operation. Even after the deletion or repositioning of that data record, its address remains in the Locate File until an ORDER BY command or another LOCATE command is given referring to that Locate File. Check for return code 08 after each GET command to see whether the address of a deleted or moved data record was retrieved. An ORDER BY command not only sorts the Locate File addresses but also removes the addresses of any data records that were deleted or moved since the Locate File was established. Thus, the Locate File could contain fewer addresses after an ORDER BY command than it did before ORDER BY.

   If more than one user has the database open, the space in the Hierarchical and Data Tables that is freed by a REMOVE TREE command is not immediately reusable. Another PLEX program might have a stack or Locate File containing the address of the removed record. It would not be appropriate to insert new, unrelated data into the record and give it a different structural position.

If no other user has a certain database open, space from removed records will be reused. The presence of this reused address in a Locate File or stack could produce the following situation. The program could give a REMOVE TREE command that causes the deletion of a data record whose address is in the Locate File. While the address for the deleted data record is still in the Locate File, an INSERT command given by the program could cause that address to be reused for a newly inserted data record. The presence of this reused address in the Locate File could cause problems for GET and ORDER BY commands using that Locate File. To avoid these problems, the program should coordinate its retrievals and deletions of data records whose addresses are in Locate Files.

If it is not possible for some program to predict and control record removal and reassignment for a particular schema record, the installation can request a special zap to prevent reuse of space until all users have closed a database. (Contact your SYSTEM 2000 Software Representative.) The space is reused when the database is opened again.

7. If the Locate File number in a LOCATE command is negative or greater than the number of extra Locate Files specified in the most recent START S2K command, the LOCATE command is ignored and return code 40 results. If no extra Locate Files are specified, only Locate File 0 is available.

8. In a where-clause condition, both the type and value of the variable must be compatible with the database definition of the item in the condition; otherwise, return code 10 results.

9. The program's current password must have W-authority (where-clause) for each item or record specified in a where-clause. If it does not, the LOCATE command will be ignored, and return code 46 or 124 (and SCF message -808-) will result.

10. The fourth field (last record indicator), sixth field (number of selected records), and seventh field (Locate File pointer) of the COMMBLOCK are set for the Locate File currently in use by the program. When a program first uses one Locate File, then switches to a second one, these COMMBLOCK fields are set with the appropriate information for the second Locate File. If the program resumes retrievals with the first Locate File, the information for that Locate File is brought back into the COMMBLOCK. Thus, for example, a program can give GET commands to retrieve records for the first five addresses in the first Locate File, then work with another Locate File for a while, and then resume retrievals at the point where it left off in the first Locate File.

11. The sixth field (number of records) of the COMMBLOCK is set by a GET1 command with a where-clause, as well as a LOCATE command. If the field is already set for a Locate File, a GET1 command with a where-clause changes it to the number of records qualified by the where-clause. The next command for the same database that is not a GET1 command with a where-clause restores the previous value for the Locate file to the sixth field.

12. Under certain conditions, the physical-order sorting of data record addresses in the Locate File can be avoided for a LOCATE command. If the where-clause consists of a single condition for a key item in the schema record associated with the SSR specified in the LOCATE command, the program can use the no-sort capability. For information on the no-sort capability, see Appendix A. This capability is useful for applications in which the desired order of data record addresses corresponds to the sorted order of values for the key item. By using no-sort, the program prevents the physical-order sort and thus eliminates the need for an ORDER BY command to arrange the addresses in the desired order.

13. If a LOCATE command is processed for a database different from the database associated with the stack of the same number, the stack is cleared.

The examples show different situations of retrieval with LOCATE. They are summarized here.

**Example 1:** The specified SSR is on the same level as the qualified records.

**Example 2:** The specified SSR is above the level of the qualified records.

**Example 3:** The specified SSR is below the level of the qualified records.

**Example 4:** The LE operator is used for a date item.

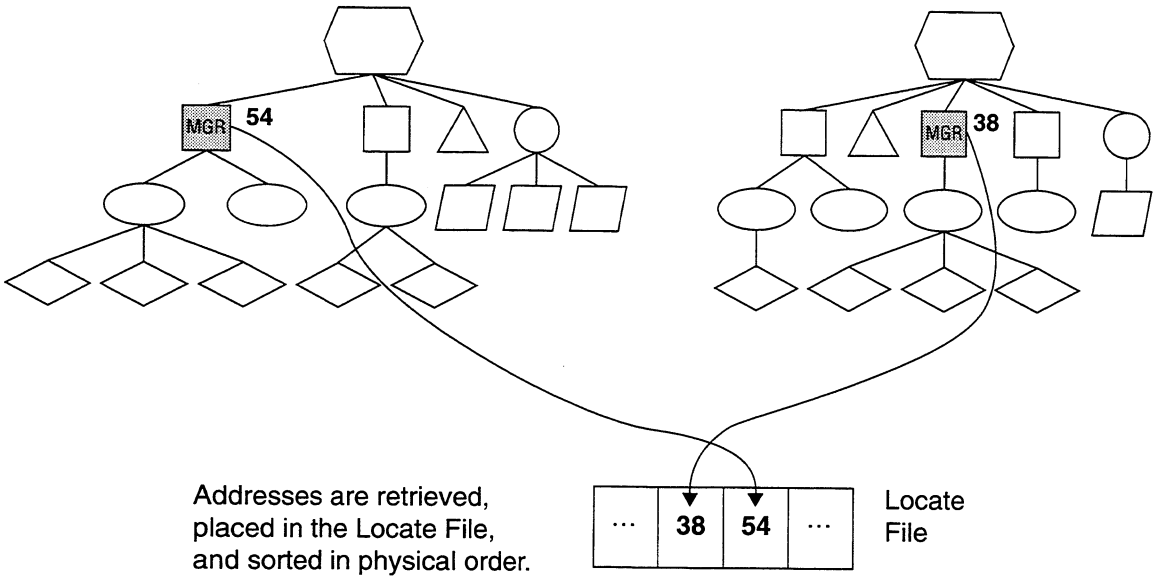
**Example 5:** The HAS operator is used to cause normalization.

**Example 1**

The object here is to retrieve ☐ records for those employees who hold or have held positions of the type MGR. The LOCATE command in this example includes a where-clause referring to the same records specified by the SSR to be located. Subschema item POSITION-TYPE is in SSR ☐ . The level of where-clause qualification is the same as the level of record selection.

MOVE 'MGR' TO POS-TYPE.

LOCATE ☐ WHERE POSITION-TYPE EQ POS-TYPE.

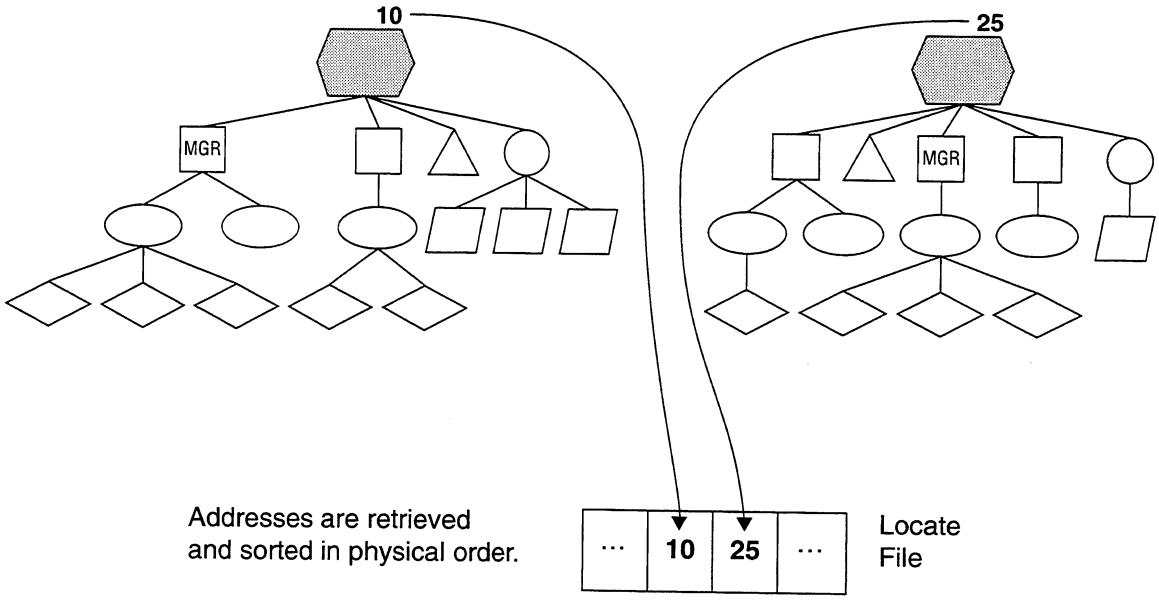


**Example 2**

The program must retrieve ⬡ records for all employees who have ever held MGR positions. The selected ⬡ records are parents of the qualified □ records.

MOVE 'MGR' TO POS-TYPE.

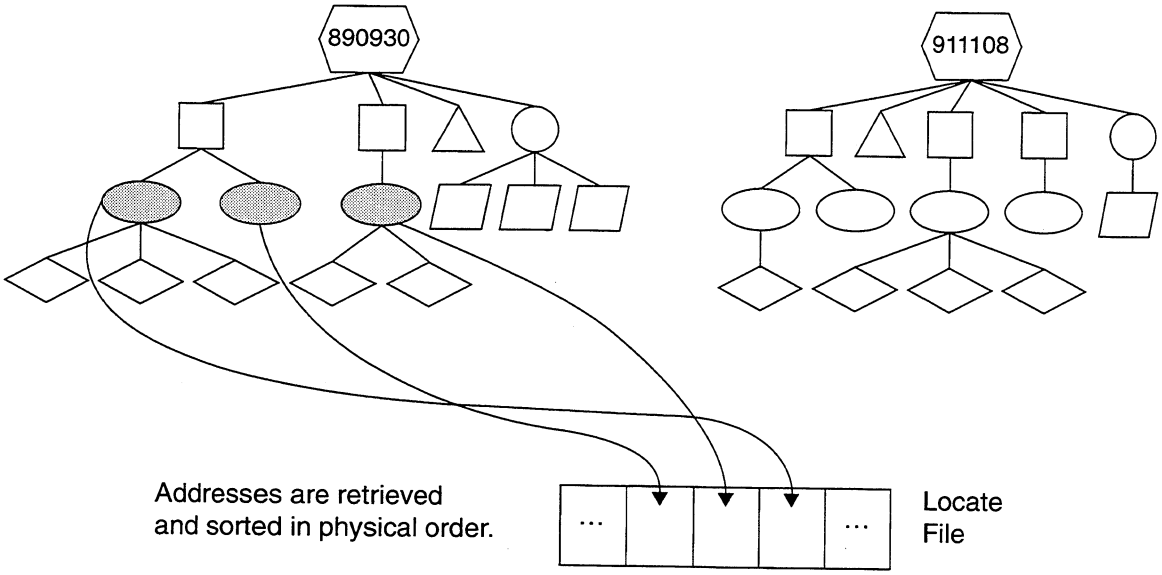
LOCATE ⬡ WHERE POSITION-TYPE EQ POS-TYPE.



**Example 4**

The program must retrieve ⬭ records for employees hired on or before January 15, 1990. Subschema item HIRE-DATE is in SSR ⬡ and has a date format of YYMMDD, so 900115 denotes January 15, 1990. The LE (less-than-or-equal-to) operator is used in this example, in which the program retrieves ⬭ descendants of qualified ⬡ records.

MOVE '900115' TO HDATE.
LOCATE ⬭ WHERE HIRE-DATE LE HDATE.

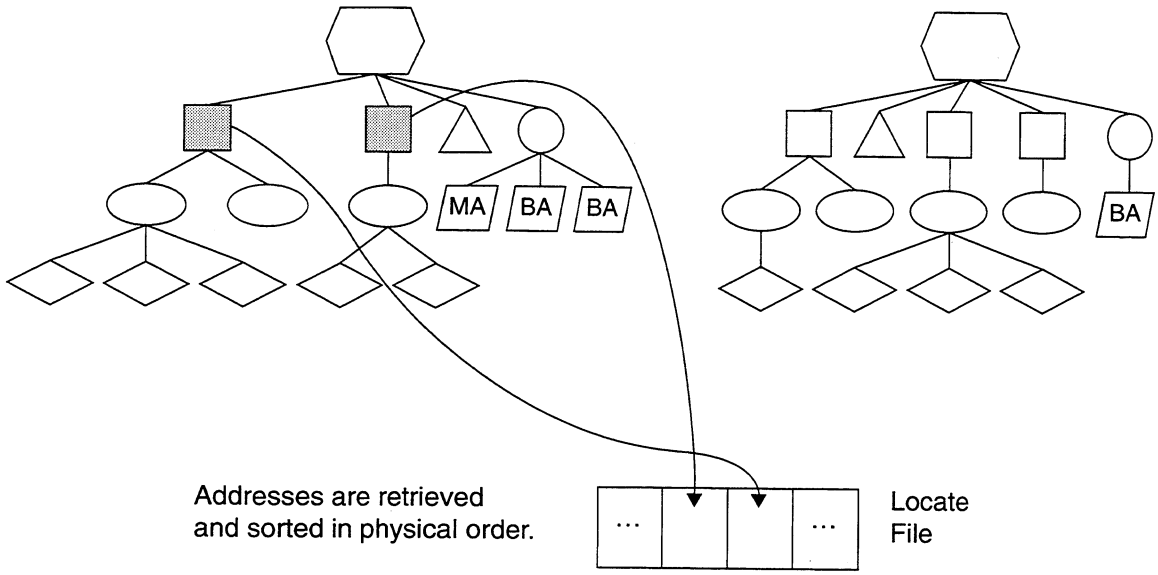


**Example 5**

The object here is to retrieve □ records for employees who have held MA degrees. Subschema item DEGREE-CERTIFICATE is in SSR ▱ . Since □ , the specified SSR, is not in the same schema path as SSR ▱ , the HAS operator is used in the where-clause to cause the necessary normalization.

MOVE 'MA' TO DEGREE.

LOCATE □ WHERE ⬡ HAS DEGREE-CERTIFICATE EQ DEGREE.



# ORDER BY

The ORDER BY command resequences the data record addresses in a Locate File according to specified sort keys.

- You can specify up to 40 sort keys in one command.
- You can specify ascending or descending sort order for each sort key.
- You can resequence any one of up to 16 Locate Files with one command.
- The sort keys in one command can be data values or data records from one or several levels of the database, all ancestors of the Locate File records.

After a LOCATE command, the selected data record addresses are in physical order in the Locate File, that is, the order in which the records are stored in the Data Table. The addresses can be resequenced with an ORDER BY command specifying that Locate File. The command can include up to 40 sort keys to determine the new sequence of addresses.

Each sort key is the name of a subschema item or of a record for data records that are ancestors of the located data records. For each sort key, either ascending or descending sort order can be specified. Ascending order (LOW) means low-to-high. Descending order means (HIGH) high-to-low. The first sort key given in an ORDER BY command is the primary sort key, the second is secondary, and so on.

You can issue the ORDER BY command at any time after a LOCATE command has established the addresses in a Locate File. After ORDER BY is executed, the Locate File pointer, which is used for GET commands, is set to the beginning of the Locate File, before the first address.

## Format

```
ORDER BY [(Locate-File-number)] ordering-list.
```

*Locate-File-number* is the number of the Locate File that holds the addresses to be resequenced: a constant or a variable whose value is equal to or greater than zero and is equal to or less than the number of extra Locate Files requested in the START S2K command. The default for *Locate-File-number* is zero. If the START S2K command does not specify a number of extra Locate Files, only Locate File 0 is available for use. For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*ordering-list*

```
[LOW ] sort-key [, ordering-list]
[HIGH]
```

The default value is LOW.

*sort-key* is the name of a subschema item or record. The value of a specified item serves as the sort key.

the short form of the command and its equivalents

**ORDER BY** *sort-key, sort-key, . . .* *(short form, using all defaults)*
**ORDER BY (0)** *sort-key, sort-key, . . .*
**ORDER BY LOW** *sort-key,* **LOW** *sort-key, . . .*
**ORDER BY (0) LOW** *sort-key,* **LOW** *sort-key, . . .*

## Examples of Syntax

**ORDER BY PAYRATE, PAYSCHED.**
*(short form implies Locate File 0; specifies* **PAYRATE** *as the primary sort key,* **PAYSCHED** *as the secondary key; sort in ascending order (default))*

**ORDER BY (3) PAYMONTH.**
*(specifies Locate File 3 and a sort key of* **PAYMONTH;** *ascending order)*

**ORDER BY HIGH PAYRATE, DEPT.**
*(specifies* **HIGH** *(for descending sort order) with a primary sort key of* **PAYRATE,** *and implies* **LOW** *(default, ascending order) with a secondary sort key)*

**ORDER BY (7) LOW LASTNAME,**
**HIGH STARTDATE, LOW PAYRATE.**
*(specifies Locate File 7, an ascending order sort on primary key, a descending order sort on the second key, and an ascending order sort on the third key)*

## Example of Usage

Here is a possible sequence of commands, including ORDER BY.

```
COMMBLOCK OF EMPLOYEE.
01  EMPLOYEE.
      .
      .
    03  RETURN-KODE.                 (field 2)
      .
      .
    03  NUMBER-OF-ADDRESSES.         (field 6)
      .
      .
SSR ENTRY-REC OF EMPLOYEE.
01  ENTRY-REC.                       (corresponds to level 0 schema record)
    03  LASTNAME . . .
        EQ LAST NAME.
    03  FORENAME . . .
      .
      .
SSR POSITION-REC  OF EMPLOYEE.        (corresponds to the level 1 POSITION
                                      WITHIN COMPANY schema record)
      .
      .

SSR SALARY OF EMPLOYEE.               (corresponds to the level² SALARY WITHIN
                                      POSITION schema record)
01  SALARY.
    03  RATE . . .
        EQ PAY RATE.
      .
      .
LOCATE SALARY WHERE . . .             (selects all records qualified;
                                      check RETURN-KODE and
                                      NUMBER-OF-ADDRESSES)

ORDER BY LASTNAME, FORENAME,          (resequence the data record addresses)
POSITION-REC, HIGH RATE.
```

**Note**: The sort order is

1. in ascending order by LASTNAME and FORENAME values in ancestor (level 0) data records

2. within the order resulting from step 1, in ascending order by ancestor POSITION-REC (level 1) data records,

3. within the order resulting from step 2, in descending order (with HIGH) by RATE values in the located (level 2) data records.

## Considerations

1. Check the return code after the ORDER BY command is given. The command's return code is placed in the second field of the COMMBLOCK for the database to which the command refers.

2. If a sort key is a subschema record for data records that are ancestors of the located data records, the Locate File addresses will be arranged according to the physical order of the ancestor data records. Within this arrangement, the addresses for descendants of any one ancestor data record will be placed in random order, unless ordered by some other key.

3. Locate File addresses that were resequenced according to specified sort keys will be in random order with respect to other subschema records and subschema items not specified as sort keys.

4. Null items for a sort key are placed before all other values by a sort in ascending order, and after all other values by a sort in descending order.

5. Before an ORDER BY command is given, check the sixth field of the COMMBLOCK. The value in this field tells the number of data record addresses currently in the Locate File. If an ORDER BY command refers to a Locate File that contains no addresses, the command is ignored and return code 23 results.

6. The program's current password must have R-authority (retrieval) for each item or schema record to which the ORDER BY command refers. If it does not, the command is ignored, return code 45 results, and the Locate File is unaffected.

7. A subschema item specified as a sort key can be part of a subschema record for the schema record of the located data records. Similarly, a sort key can be a subschema item that is part of a subschema record for an ancestor schema record. But if a specified sort key is an item in a subschema record for a descendant schema record, the command is ignored, and return code 24 results.

8. A subschema record named as a sort key must correspond to an ancestor schema record of the data records that are ancestors of the located data records. If it does not, return code 24 results.

9. If the name of a subschema record or item is repeated in the list of sort keys, the PLEX processor issues an error message for the ORDER BY command.

10. If the specified Locate File number is negative or is greater than the number of Locate Files specified in the START S2K command, the command is ignored, and return code 40 results. If no extra Locate Files are specified, only Locate File 0 is available.

11. After a LOCATE command is given, a data record whose address is in the Locate File might be deleted from the database by a REMOVE TREE operation or might be moved by a MOVE TREE operation. Even after the data record is deleted or moved, its address remains in the Locate File until an ORDER BY command or another LOCATE command is given referring to that Locate File. An ORDER BY command causes a specified resequencing of the Locate File addresses and the removal of addresses of any data records that were deleted or moved since the most recent LOCATE command was given for the Locate File.

12. If an ORDER BY command causes the removal of all the addresses in a Locate File, return code 23 results. In addition, these three COMMBLOCK fields are cleared: the flag for the last Locate File address (fourth field), the number of Locate File addresses (sixth field), and the position of the Locate File pointer (seventh field).

13. The space in the Hierarchical and Data Tables that is freed by a REMOVE TREE command is not immediately reusable if more than one user has the database open. Another PLEX program can have a stack or Locate File containing the address of the removed record. It would not be appropriate to insert new, unrelated data into the record and give it a different structural position.

    If no other user has a certain database open, space from removed records will be reused. The presence of this reused address in a Locate File or stack could produce the following situation: A program could give a REMOVE TREE command that causes the deletion of a data record whose address is in the Locate File. While the address of the deleted data record is still in the Locate File, an INSERT command given by the program could cause that address to be reused for a newly inserted data record. The presence of this reused address in the Locate File could cause problems for later GET or ORDER BY commands referring to that Locate File. To avoid these problems, the program should coordinate its retrievals and deletions of data records whose addresses are in Locate Files.

    If it is impossible for a program to predict and control record removal and reassignment for a particular schema record, your site can request a special zap to prevent reuse of space until all users have closed a database. (Contact your SYSTEM 2000 Software Representative.) The space is reused when the database is opened again.

14. When an ORDER BY command is processed, a combined sort key is assembled for each located data record. Each combined sort key consists of one sort field per value or ancestor record address corresponding to ORDER BY specifications. The HIGH or LOW option is represented as part of each field. ORDER BY processing then sorts these combined sort keys in relation to one another according to the specification of HIGH or LOW for each sort field. The resulting order of these combined sort keys determines the new order of data record addresses in the Locate File.

    The length of the combined sort key to be used for a located record depends on the combined number of full and partial words occupied by the values for the sort keys specified in the ORDER BY command, according to the following table:

| Data Type | Number of Words |
|---|---|
| CHARACTER, TEXT,or UNDEFINED value | (number of full words and partial word) + 1 |
| INTEGER, DECIMAL, MONEY, REAL, DOUBLE, or DATE value | 2 |
| Null data item | 1 |
| Ancestor record address | 1 |

If the length of the combined sort key is greater than 62 words, the combined sort-key is truncated to 62 words during ORDER BY processing. In addition, return code 25 results, indicating that the combined sort-key was truncated.

15. LOCATE command processing includes a physical-order sort of the located record addresses (unless the no-sort capability is used). When an ORDER BY command is executed after a LOCATE command, each Data Table page is read once, and combined sort keys are created. A program can give a subsequent ORDER BY command for the same Locate File, as shown here.

```
LOCATE . . .
       :
       :
ORDER BY . . .
       :
       :
 (commands using the ordered Locate File)
       :
       :
ORDER BY . . .
       :
       :
 (commands using the reordered Locate File)
```

When the second ORDER BY command is processed, Data Table pages are read according to the order of addresses established by the first ORDER BY. In this case, Data Table pages might have to be accessed more than once, causing either excessive disk-head movement or I/O or both. If the program has another Locate File available to it and if the volume of data warrants the effort, your program can issue a COPYL command to make a copy of the Locate File before you give the first ORDER BY command. It can then use the copy for the second ORDER BY.

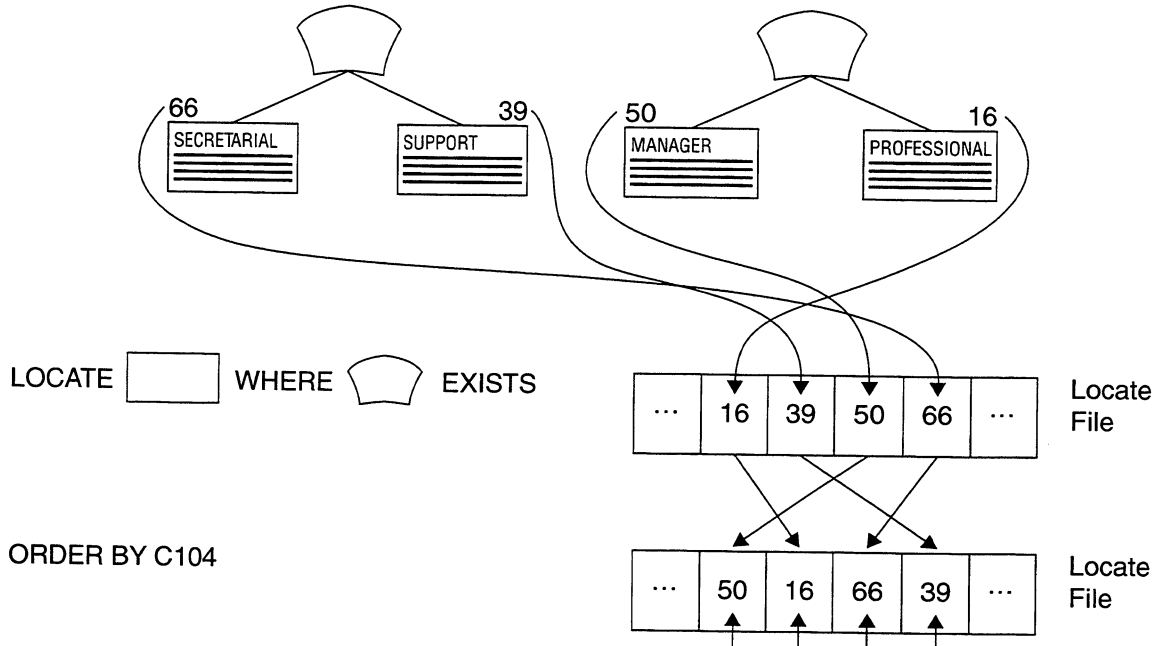
The illustrations show different situations of using ORDER BY. They are summarized here.

**Example 1:** sorts by the values in the located records.

**Example 2:** sorts by the parents of the located records.

**Example 1**

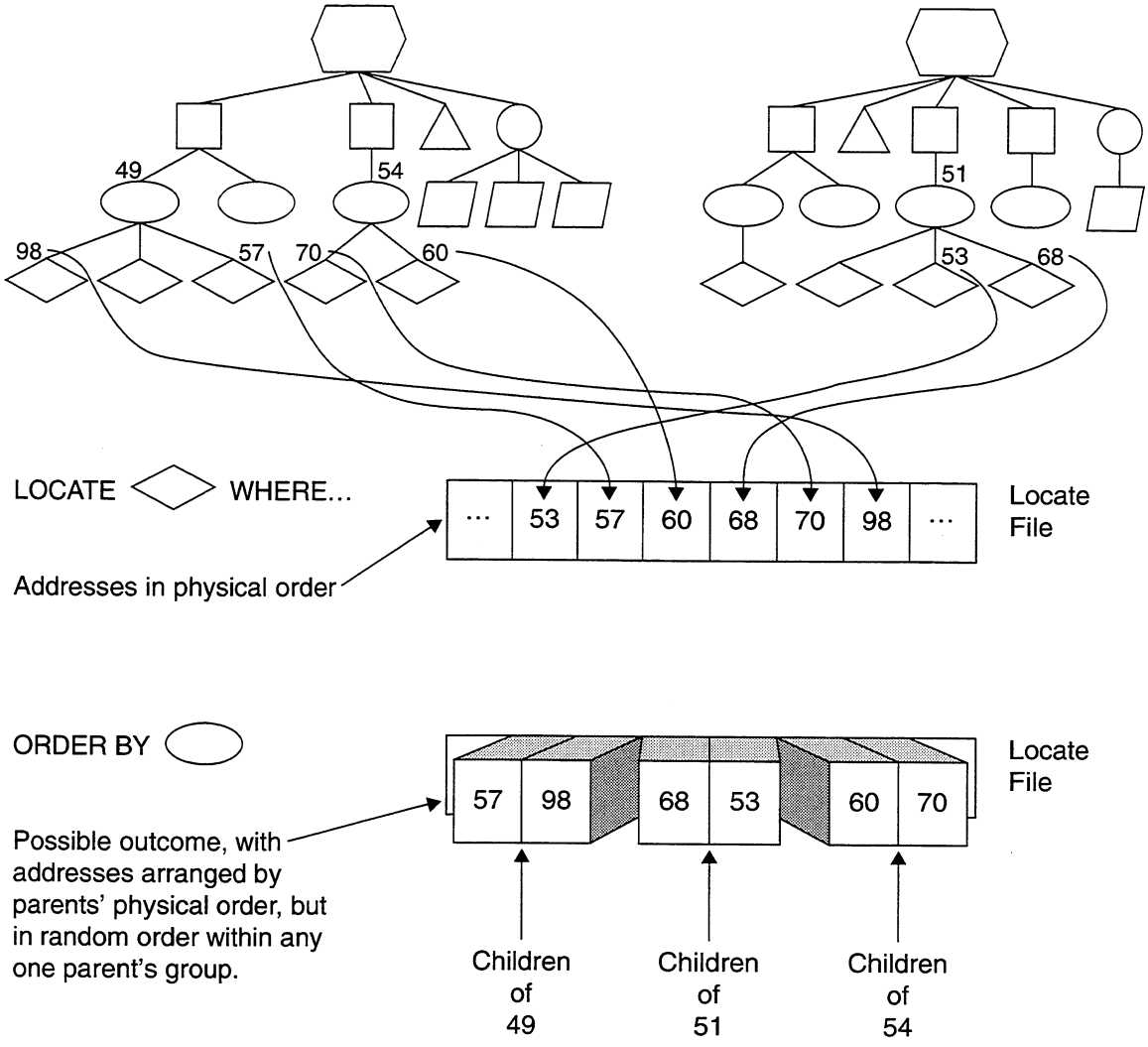
After a LOCATE command selects the addresses of ▭ records, an ORDER BY command sorts those addresses according to the alphabetical order of values for an item in each of those records. Subschema item POSITION-TYPE (C104) is in SSR ▭ .




| data record address | value | alphabetical order of values |
|---|---|---|
| 16 | PROFESSIONAL | 2nd |
| 39 | SUPPORT | 4th |
| 50 | MANAGER | 1st |
| 66 | SECRETARIAL | 3rd |

**Example 2**

In this example, an ORDER BY command arranges the selected record addresses according to the physical order of parents of the selected data records.



# GET

The GET command retrieves a data record whose address is in a specified Locate File. GET establishes a primary stack entry. The GET command offers the following features:

- Like LOCATE, the GET command enables retrieval from any level of the database.
- You can specify any Locate File/stack pair.
- Movement through the Locate File of data record addresses can be left-to-right or right-to-left, depending on your choice of NEXT, FIRST, LAST, PREVIOUS, PRESENT, or your specified S2KCOUNT.
- With the USING 0 option, your program can retrieve a data record address without retrieving data values.
- You can retrieve values for specific (rather than all) items of a subschema record by using an inclusion list.
- The HOLD option in a Multi-User environment protects the integrity of retrieved data. See **HOLD Option** on page 8-131 for details.

A GET command specifies a stack, a Locate File of data record addresses, and a subschema record associated with those data records. The number specified for the stack also indicates the Locate File to be used.

A GET command retrieves values from one of the located data records, placing the values into the subschema record area. The address of that data record becomes the primary stack entry; any other entries in the stack are canceled. A series of GET NEXT commands, each using the same Locate File, causes the retrieval operation to progress sequentially through the Locate File. When GET commands specify FIRST, LAST, PREVIOUS, PRESENT, or S2KCOUNT, the program can access the first, last, or previous address, or a specific address (skipping over others).

For a discussion of inclusion lists, see **Inclusion List** on page 8-121. The USING 0 option is discussed in **USING 0 Option** on page 8-131.

## Format

```
GET [/HOLD] [(m)] SSR |[inclusion-list] |[NEXT]
                                        |[FIRST]
                                        |[LAST]
                      |[USING 0]        |[PREVIOUS]
                                        |[PRESENT]
                                        |[S2KCOUNT]
```

*m* is the number of the Locate File and stack to be used in retrieval: a constant or a variable whose value is equal to or greater than zero and is equal to or less than the number of extra Locate Files requested in the START S2K command. The default is 0 (both Locate File and stack). For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*SSR* is the name of the subschema record for the data record to be retrieved.

*inclusion-list*

```
|item-list

|USING |name-table
       |word-table
```

*item-list*

```
item-name  [, item-list]
```

*item-name* is the alias of an item in the specified subschema record.

*name-table* is the name of a table of component names or C-numbers of items to be included in the retrieval operation. For COBOL, the entry with the OCCURS clause must specify PICTURE X(32).

*word-table* is the name of a table of one-word indicators (1s and 0s), one indicator for each item in the specified subschema record. An item is included in the retrieval operation if its indicator in the table is 1. Conversely, an item is excluded from the operation if its indicator is zero. In each indicator, only the low-order bit is checked to see if it is zero or 1. For COBOL, the entry with the OCCURS clause must specify COMP usage.

short form of the command and its equivalents

```
GET SSR.          (short form implies GET SSR NEXT)
GET SSR NEXT.
GET (0) SSR.
GET (0) SSR NEXT.
```

## Examples of Syntax

```
GET (3) POSITIONS.                          (specifies stack and subschema record;
                                            implies NEXT)

GET/HOLD (3) SKILLS.                        (specifies HOLD in Multi-User;
                                            implies NEXT)

GET (2) PAYROLL NEXT.                       (gives NEXT explicitly)

GET EDUC LAST.                              (specifies LAST instead of NEXT)

GET (7) POSITIONS TITLE, DIV PRESENT.       (specifies item list and PRESENT)

GET (4) TRAINING USING 0 FIRST.             (specifies address retrieval only
                                            and FIRST)

GET/HOLD (1) SALARY USING                   (specifies name table and HOLD option
     SAL-NAME-TABLE S2KCOUNT.               and S2KCOUNT)

GET (0) POSITIONS USING                     (specifies word table and NEXT)
   POS-WORD-TABLE NEXT.
```

## Example of Usage

Possible sequence of commands, including GET

```
WORKING-STORAGE SECTION.
      :
      :
COMMBLOCK OF EMPLOYEE.
      :
      :
SSR SKILLS OF EMPLOYEE.          (subschema record corresponds
                                 to JOB SKILLS schema record)

      :
      :
PROCEDURE DIVISION.
      :
      :
   LOCATE SKILLS WHERE . . .     (retrieves the address of all SKILLS
                                 data records meeting the where-clause
                                 conditions)

   MOVE -4 TO S2KCOUNT.          (sets S2KCOUNT to -4 for the GET command;
                                 S2KCOUNT has COMP usage and is declared
                                 in the S2KDUM control block.)

   GET SKILLS S2KCOUNT.          (retrieves the SKILLS data record
                                 whose address is 4 ordinal places
                                 to the left (S2KCOUNT = - 4) of
                                 the address of the data record
                                 previously retrieved. This ordinal
                                 place is 4 less than the current
                                 value of the seventh COMMBLOCK field.)
```

**Note**: If the Locate File is positioned before the first address and the count is negative, movement starts from the end. In this example, if the Locate File contained 20 addresses, the 17th address would be used by the GET command.

## Considerations

1. Check the return code after each GET command is given. The command's return code is in the second field of the COMMBLOCK for the database to which the command refers.

2. The sixth COMMBLOCK field contains the number of data record addresses in the Locate File established by the most recent LOCATE command. It should be checked before the first GET command is given referring to that Locate File. The purpose of this check is to find out if the database actually has one or more data records that met the where-clause conditions in the LOCATE command. A return code of zero for a LOCATE command means only that the syntax and processing of the command were without error; it does not indicate whether any data records were located. If a GET command is given that refers to an empty Locate File, return code 06 results.

3. The seventh COMMBLOCK field represents the Locate File pointer; it indicates the ordinal place in the Locate File of the address of the data record most recently retrieved by a GET command. A LOCATE command causes this field to be set to 0. A subsequent GET command that retrieves a data record whose address is in this Locate File sets the field to the number for the place of that data record in the Locate File. For example, if a GET command just retrieved the data record whose address is second in the Locate File, the seventh COMMBLOCK field is set to 2.

4. The fourth field of the COMMBLOCK is a flag indicating whether the data record whose address is last in the Locate File was retrieved with a GET command. If the last located data record was retrieved by the most recent GET command, the value of this field is 1. If not, the value of this field is 0.

5. Although a series of GET1 NEXT or GETD NEXT commands will cause retrieval in logical order, a series of GET NEXT commands will cause the retrieval of data records in the order shown by the addresses in the Locate File. The addresses in the Locate File can be either in physical order or in an order determined by the sort keys of a previous ORDER BY command.

6. For moving through the addresses of the Locate File from left to right, a GET NEXT command can be given repeatedly, without an initial GET FIRST. When a LOCATE or ORDER BY command was given, and the value of the seventh COMMBLOCK field is 0 (indicating no data records were retrieved from Locate File addresses), GET NEXT retrieves the data record whose address is leftmost in the Locate File, just as GET FIRST will. The second GET NEXT retrieves the data record whose address is second, and so on, through the end of the Locate File.

7. For moving through the addresses of the Locate File from right to left, the GET PREVIOUS command can be given repeatedly, without an initial GET LAST. When a LOCATE or ORDER BY command was just given, and the value of the seventh COMMBLOCK field is 0, GET PREVIOUS retrieves the data record whose address is rightmost in the Locate File, just as GET LAST will. The second GET PREVIOUS retrieves the data record whose address is next to last, and so on, through to the beginning of the Locate File.

8. If a GET command specifies an ordinal place for a data record address that is beyond the end or the beginning of the Locate File, the command is ignored and return code 04 results. This could happen in any of the following four situations:

   - if the data record whose address is last in the Locate File was retrieved (the fourth COMMBLOCK field equals 1) and another GET command specifies NEXT.
   - if the value of S2KCOUNT is a positive integer greater than the difference between the values of the sixth COMMBLOCK field (number of located data records) and the seventh COMMBLOCK field (Locate File pointer).
   - if the data record whose address is first in the Locate File was just retrieved and another GET command specifies PREVIOUS.
   - if the value of S2KCOUNT is a negative integer whose absolute value is greater than the value of the seventh COMMBLOCK field (Locate File pointer).

9. If the absolute value of S2KCOUNT for a GET command is greater than the value of the sixth COMMBLOCK field (the number of located data records), the command is ignored and return code 04 results.

10. You can use a GET PRESENT (or S2KCOUNT = 0) to restore the primary stack entry when it was changed by a GET1 command referring to the same stack. It also restores the corresponding subschema record values. In response to GET PRESENT, SYSTEM 2000 software retrieves from the specified Locate File the address last referred to by a GET command for that Locate File. The address once again becomes the primary stack entry.

11. If no data record whose address is in the Locate File was retrieved yet with a GET command (the seventh COMMBLOCK field equals 0), a GET command specifying PRESENT (or S2KCOUNT = 0) is ignored and return code 06 results.

12. If the value moved to S2KCOUNT by the program is not an integer (positive, negative, or zero), a GET S2KCOUNT command results in return code 31.

13. The subschema record specified in a GET command can be different from the one specified in the associated LOCATE command, as long as both subschema records correspond to the same schema record. If they do not, the GET command is ignored and return code 18 results.

14. If the number given for the stack and Locate File to be used in a GET command is different from the number specified in the previous command, the information about the Locate File with that number is brought into the COMMBLOCK. These fields of the COMMBLOCK are set as follows:

    Field 4 is a flag indicating retrieval of the last located data record.
    Field 6 is the number of located data records.
    Field 7 is the Locate File pointer, that is, the ordinal position of the address of the most recently retrieved data record.

15. The number for the stack and Locate File referred to in a GET command can be a variable. If the value for that variable is negative or greater than the number of extra Locate Files being used, return code 40 results.

16. After a LOCATE command is given, a data record whose address is in the Locate File might be deleted from the database by a REMOVE TREE operation or moved by a MOVE TREE operation. Even after the data record is deleted or moved, its address remains in the Locate File until an ORDER BY command or another LOCATE command is given referring to that Locate File. While this address remains in the Locate File, it can be retrieved into a stack by a GET command, with appropriate settings in COMMBLOCK fields 4, 6, and 7. If this happens, return code 08 results.

17. The space in the Hierarchical and Data Tables that is freed by a REMOVE TREE command is not immediately reusable if more than one user has the database open. Another PLEX program can have a stack or Locate File containing the address of the removed record. It would not be appropriate to insert new, unrelated data into the record and give it a different structural position.

    If no other user has a certain database open, space from removed records is reused. The presence of this reused address in a Locate File or stack could produce the following situation. The program could give a REMOVE TREE command that causes the deletion of a data record whose address is in the Locate File. While the address for the deleted data record is still in the Locate File, an INSERT command given by the program could cause the address to be reused for a newly inserted data record. The presence of this reused address in the Locate File could cause problems for GET and ORDER BY commands using that Locate File. To avoid these problems, the program should coordinate its retrievals and deletions of data records whose addresses are in Locate Files.

    If it is impossible for some program to predict and control record removal and reassignment for a particular schema record, the installation can request a special zap to prevent reuse of space until all users have closed a database. (Contact your SYSTEM 2000 Software Representative.) The space is reused when the database is opened again.

18. If the MULTIPLE HOLDS option is enabled, the PLEX program can request any number of local holds on different records simultaneously. The MULTIPLE HOLDS option is disabled by default. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for details about setting the MULTIPLE HOLDS option. Local holds remain in effect until one of the following events occurs in the PLEX program:

    - a global hold is attempted for any database
    - a FRAME command is issued for any database
    - a synchpoint occurs
    - the program terminates
    - a DROP HOLD command occurs
    - the QUEUE command is issued.

When any of these situations occurs, all of the PLEX program's local holds on the database(s) are dropped. Also, an update command no longer causes a local hold to be dropped.

See **HOLD Option** on page 80 for a complete discussion of local holds. See also **HOLD Option** on page 8-131 and **DROP HOLD** on page 8-144.

## GET NEXT

```
GET [/HOLD] [(m)] SSR |[inclusion-list] [NEXT]
                      |[USING 0]
```

A GET command using NEXT (by specifying NEXT or leaving blank for the default) moves from left to right through a Locate File's addresses, retrieving corresponding data records from the database.

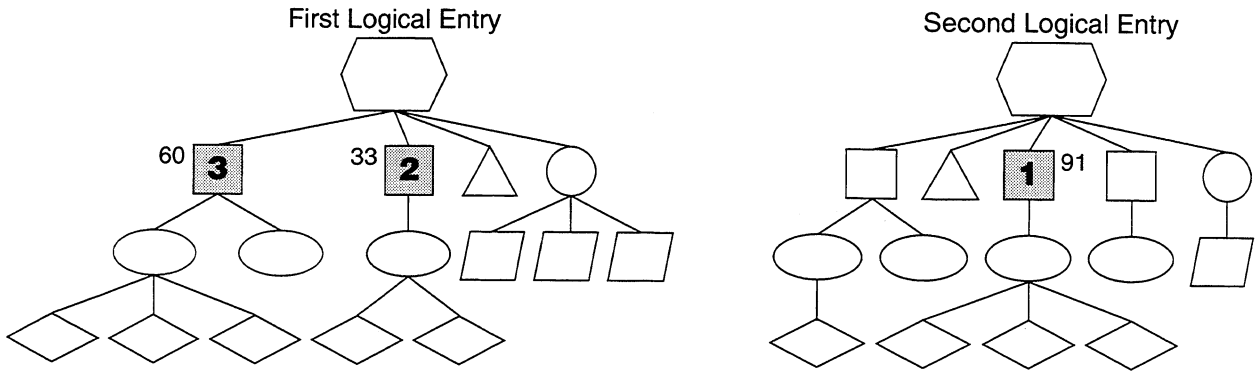
With NEXT, the GET command retrieves the data record whose address in the Locate File comes next (to the immediate right of the address last referred to by a GET command) even if some command besides GET was issued in the meantime. If no data record was retrieved yet based on the Locate File's addresses, following either a LOCATE command or an ORDER BY command, GET NEXT causes the retrieval of the data record whose address is first (leftmost) in the Locate File.

Be sure to review the items listed in **Considerations** on page 8-104 for any additional information.

The following example shows a sample retrieval with GET NEXT.

**Example 1**

Suppose a Locate File is established containing three record addresses. Following the LOCATE and ORDER BY commands, a sequence of GET NEXT commands is issued. The first GET NEXT gives the same result as GET FIRST. For the third GET NEXT, the end of the Locate File is reached; the next GET NEXT results in return code 04.




LOCATE [ ] WHERE ...

| 33 | 60 | 91 | Locate File (0) |
|---|---|---|---|

ORDER BY ...

| 91 | 33 | 60 | Locate File (0) |
|---|---|---|---|

**1** GET [ ] NEXT

Stack (0)

| **91** | 33 | 60 | Locate File (0) |
|---|---|---|---|

**2** GET [ ] NEXT

Stack (0)

| 91 | **33** | 60 | Locate File (0) |
|---|---|---|---|

**3** GET [ ] NEXT

Stack (0)

| 91 | 33 | **60** | Locate File (0) |
|---|---|---|---|

(Fourth field of COMMBLOCK (last record flag) set to show that last address was accessed.)

GET [ ] NEXT

Stack (0)

| 91 | 33 | 60 | Locate File (0) |
|---|---|---|---|

(return code 04: tried to retrieve past end of Locate File)

## GET FIRST

```
GET [/HOLD] [(m)] SSR |[inclusion-list] FIRST
                      |[USING 0]
```

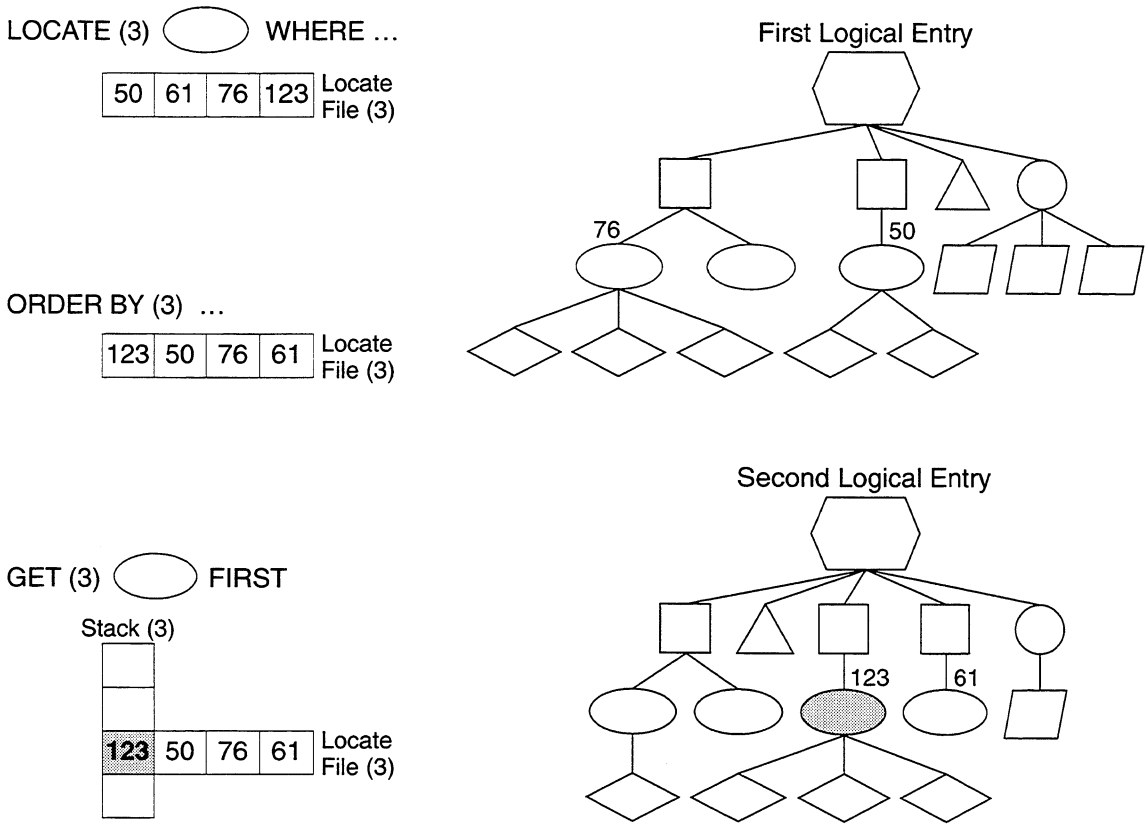
A GET command specifying FIRST retrieves the data record whose address is first (leftmost) in the specified Locate File. If a GET NEXT command is the first GET command following the LOCATE or ORDER BY command (the seventh COMMBLOCK field is zero), it gives the same result as GET FIRST.

Be sure to review the items under **Considerations** on page 8-104 for any additional information.

The following example shows a sample retrieval with GET FIRST.

**Example 1**

Suppose a Locate File is established containing four record addresses. Following the LOCATE and ORDER BY commands, GET FIRST retrieves the record whose address comes first in the Locate File.



## GET LAST

```
GET [/HOLD] [(m)] SSR |[inclusion-list] LAST
                      |[USING 0]
```

A GET command specifying LAST retrieves the data record whose address is last (rightmost) in the specified Locate File.

If a GET LAST command is successful, the fourth field (last record flag) of the COMMBLOCK is set to show that the last record address in the Locate File was accessed. This field is set also if a GET LAST command results in return code 08 signifying that the last record was removed or return code 45 if the program's password lacks R-authority for an item in the specified subschema record.

Be sure to review the items listed under **Considerations** on page 8-104 for any additional information.

The illustrations show sample retrievals with GET LAST. They are summarized here.

**Example 1:** GET LAST retrieves the record whose address is last.

**Example 2:** The last record was removed, so GET LAST results in return code 08.

## Example 1

A LOCATE and an ORDER BY command are given for Locate File 1. Then a GET LAST command is issued to retrieve the record whose address comes last in the Locate File. The fourth COMMBLOCK field (last record flag) is set to show that the last record was retrieved.

LOCATE (1) ☐ WHERE ...

| 33 | 60 | 91 | Locate File (1) |
|---|---|---|---|

ORDER BY (1) ...

| 60 | 91 | 33 | Locate File (1) |
|---|---|---|---|

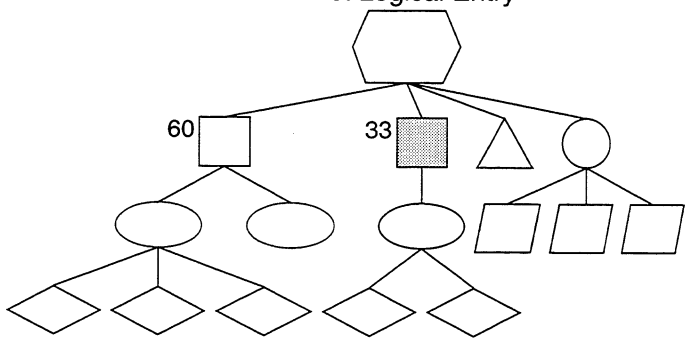
GET (1) ☐ LAST

Stack (1)

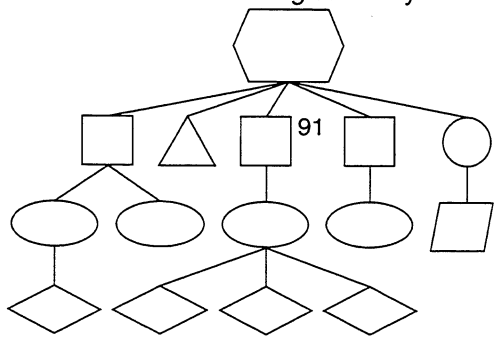
| 60 | 91 | **33** | Locate File (1) |
|---|---|---|---|

(The fourth COMMBLOCK field (last record flag) is set to show that the last address was accessed.)

First Logical Entry




Second Logical Entry



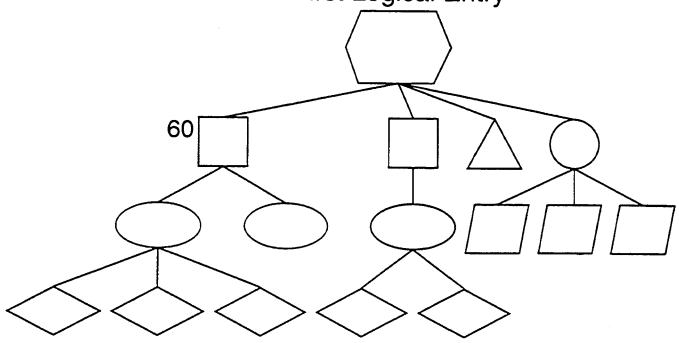
**Example 2**

After a Locate File is established, a REMOVE TREE operation removes one of the located records. The GET LAST command results in return code 08, signifying that the record was removed. The fourth COMMBLOCK field is still set to show that the last address was accessed.

LOCATE (1) [ ] WHERE ...

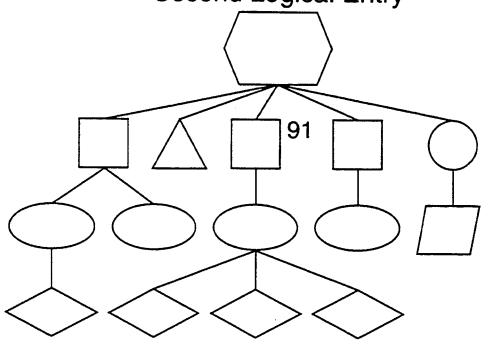
| 33 | 60 | 91 | Locate File (1) |
|---|---|---|---|

First Logical Entry




( A REMOVE TREE issued either by this program or by another user removes the subtree with [ ] 91 as its top node. )

Second Logical Entry

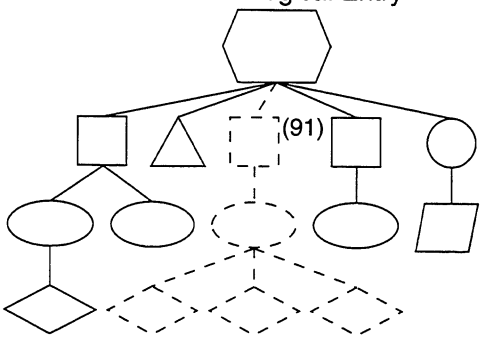



GET (1) [ ] LAST

Stack (1)

| | | (91) | Locate File (1) |
|---|---|---|---|

Second Logical Entry




( A non-zero return code results, signifying that the record is no longer in the data base. The fourth COMMBLOCK field (last record flag) is set to show that the last address was accessed. )

## GET PREVIOUS

```
GET [/HOLD] [(m)] SSR |[inclusion-list] PREVIOUS
                      |[USING 0]
```

A GET command specifying PREVIOUS is the opposite of GET NEXT. PREVIOUS moves right-to-left through a Locate File's addresses, retrieving data records at those addresses in the database.

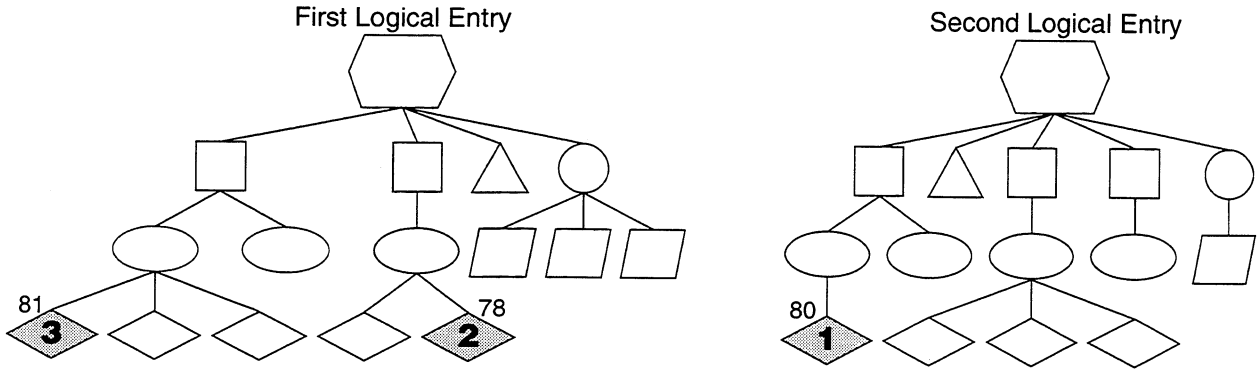
With PREVIOUS, the GET command results in the retrieval of the data record whose address in the Locate File is to the immediate left of the address last referred to by a GET command, even if some command besides GET was issued in the meantime. If no located data record was retrieved yet (following either a LOCATE or an ORDER BY command), GET PREVIOUS retrieves the data record whose address is last (rightmost) in the Locate File.

Be sure to review the items in **Considerations** on page 8-104 for any additional information.

The following example illustrates retrieval with GET PREVIOUS.

**Example 1**

Suppose a Locate File is established containing three record addresses. Following the LOCATE and ORDER BY commands, a sequence of GET PREVIOUS commands is issued. The first GET PREVIOUS gives the same result as a GET LAST. For the third GET PREVIOUS, the beginning of the Locate file is reached; the next GET PREVIOUS results in return code 04.




LOCATE ◇ WHERE ...

| 78 | 80 | 81 | Locate File (0) |
|---|---|---|---|

ORDER BY ...

| 81 | 78 | 80 | Locate File (0) |
|---|---|---|---|

**1** GET ◇ PREVIOUS
or
GET ◇ LAST

Stack (0)

| 81 | 78 | **80** | Locate File (0) |
|---|---|---|---|

**2** GET ◇ PREVIOUS

Stack (0)

| 81 | **78** | 80 | Locate File (0) |
|---|---|---|---|

**3** GET ◇ PREVIOUS

Stack (0)

| **81** | 78 | 80 | Locate File (0) |
|---|---|---|---|

**4** GET ◇ PREVIOUS

Stack (0)

| 81 | 78 | 80 | Locate File (0) |
|---|---|---|---|

(return code 04: tried to retrieve past beginning of Locate File)

## GET PRESENT

```
GET [/HOLD] [(m)] SSR |[inclusion-list] PRESENT
                      |[USING 0]
```

A GET command specifying PRESENT retrieves the data record most recently retrieved using the specified Locate File. GET PRESENT is useful when the program must issue a GETD or GETA command based on a primary stack entry established by the latest GET command but canceled by an intervening GET1 command for the same stack. GET PRESENT can also be used to specify an alternate SSR for the same schema record as the SSR in an original GET NEXT command, for example. A program might first issue a GET NEXT command specifying either USING 0 or an SSR containing only a small fraction of the number of items in each corresponding data record. Then, the program might check for some condition in related records by GETA or GETD commands. If the program finds the condition, it can issue a GET PRESENT command to retrieve the located data record again, this time retrieving a large number of items from the record by specifying an SSR containing more items than the originally specified SSR.

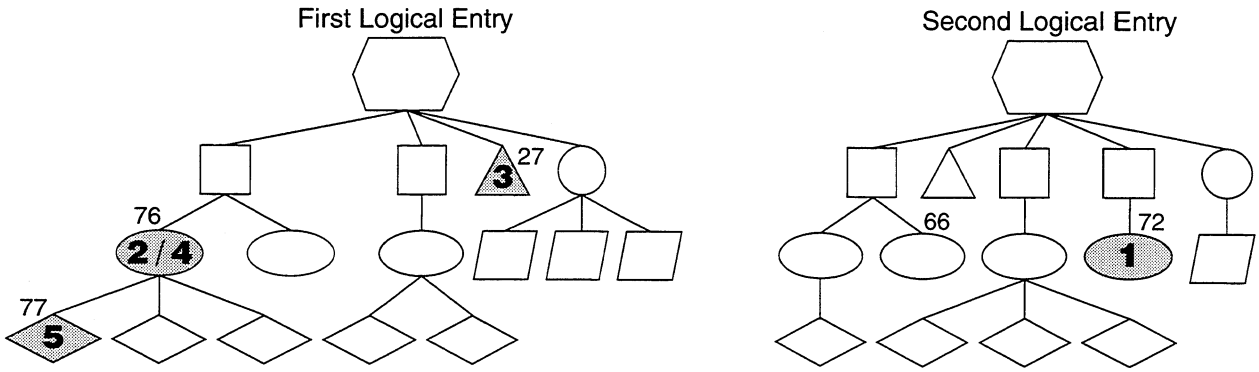
Be sure to review the items listed under **Considerations** on page 8-104 for any additional information.

If a sequence of GET commands is interrupted by a GET1 or GETP command for the same stack, the sequence can be resumed where it left off in the Locate File. If your application needs to return to the record it retrieved with the most recent GET command, issue a GET PRESENT command. However, if you do not need to retrieve that record again, give only the next GET command in the sequence.

The following example illustrates retrieval with GET PRESENT.

**Example 1**

After a Locate File is established containing three addresses, suppose a sequence of GET NEXT commands is issued. The sequence is interrupted by a GET1 command that changes the primary entry in the stack. Then the program gives a GET PRESENT command to retrieve the same record retrieved by the most recent GET NEXT command. The program gives the GET PRESENT command to reestablish the primary stack entry so that a GETD command can retrieve a descendant record.




LOCATE (10) ⬭ WHERE ...

| 66 | 72 | 76 | Locate File (10) |
|---|---|---|---|

ORDER BY (10) ...

| 72 | 76 | 66 | Locate File (10) |
|---|---|---|---|

**1** GET (10) ⬭ NEXT

Stack (10)

| | | | |
|---|---|---|---|
| **72** | 76 | 66 | Locate File (10) |

**2** GET (10) ⬭ NEXT

Stack (10)

| 72 | **76** | 66 | Locate File (10) |
|---|---|---|---|

**3** GET1 (10) △ NEXT

Stack (10)

| * | **27** |
|---|---|

**4** GET (10) ⬭ PRESENT

Stack (10)

| 72 | **76** | 66 | Locate File (10) |
|---|---|---|---|

**5** GET (10) ◇ NEXT

Stack (10)

| 72 | 76 | 66 | Locate File (10) |
|---|---|---|---|
| | **77** | | |

## GET S2KCOUNT

```
GET [/HOLD] [(m)] SSR |[inclusion-list] S2KCOUNT
                      |[USING 0]
```

S2KCOUNT provides flexibility in retrieval during a program run and from one run to another. Depending on the value a program assigns to S2KCOUNT, a GET S2KCOUNT command can be the equivalent of GET NEXT, GET PREVIOUS, or GET PRESENT. More importantly, S2KCOUNT provides the ability to select every *nth* record when S2KCOUNT=*n*.

S2KCOUNT has no default value. Before using S2KCOUNT in a GET command, your program must move a value to the variable S2KCOUNT, which resides in the S2KDUM control block. This value can be 0 or a positive or negative integer. Any value that is not an integer causes return code 31 when executed.

GET S2KCOUNT indirectly specifies the ordinal position in the Locate File of the address of the data record to be retrieved. The ordinal position of the address for the data record to be retrieved is determined by adding S2KCOUNT to the current value of the seventh COMMBLOCK field. Thus, if S2KCOUNT is set to a positive value, an address is used whose ordinal position is greater than (to the right of) that of the most recently retrieved data record. Conversely, if S2KCOUNT is a negative value, an address is used whose ordinal position is less than (to the left of) that of the most recently retrieved data record. The following are equivalents of the GET S2KCOUNT command when the value is 0, 1 , or -1.

| When | GET S2KCOUNT means |
|---|---|
| **S2KCOUNT = 1** | **GET NEXT** |
| **S2KCOUNT = 0** | **GET PRESENT** |
| **S2KCOUNT = -1** | **GET PREVIOUS** |

If a GET S2KCOUNT is issued immediately after a LOCATE or an ORDER BY command, the value in S2KCOUNT determines the outcome.

| S2KCOUNT | Result |
|---|---|
| **+n** | Starts at beginning and retrieves the nth address. |
| **0** | Return code is 04. |
| **-n** | Starts at the end; -1 retrieves the last address. |

Be sure to review the items listed under **Considerations** on page 8-104 for any additional information. Note especially item 10, regarding restoring the primary stack entry.

The illustrations show different sample retrievals with GET S2KCOUNT. They are summarized here.

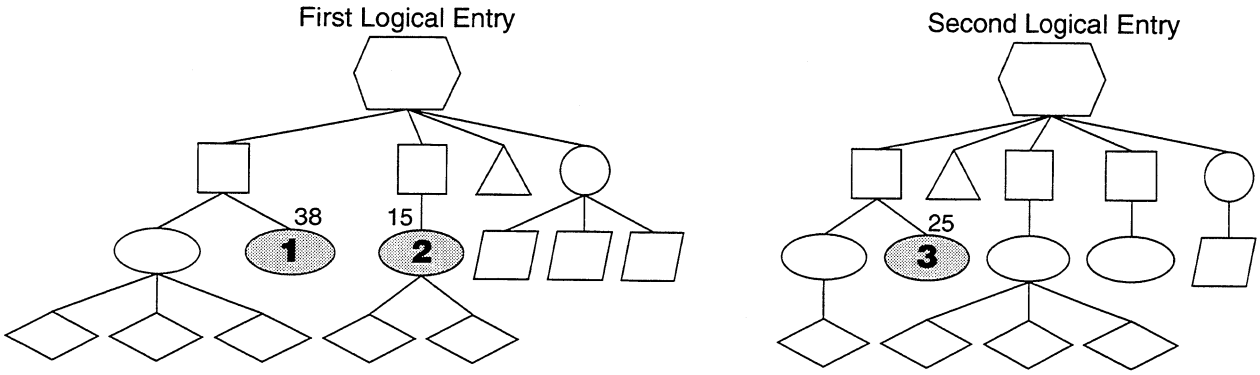
**Example 1:** S2KCOUNT equals 1.

**Example 2:** S2KCOUNT equals 2.

**Example 3:** S2KCOUNT equals -1.

**Example 1**

After a Locate File is established, the program issues a sequence of GET S2KCOUNT commands. It has already assigned S2KCOUNT the value 1. When S2KCOUNT equals 1, GET S2KCOUNT results in the same retrieval as GET NEXT.




MOVE 1 TO S2KCOUNT.
[COBOL Statement]

LOCATE (5) ⬭ WHERE ...

| 15 | 25 | 38 | Locate File (5) |
|---|---|---|---|

ORDER BY (5) ...

| 38 | 15 | 25 | Locate File (5) |
|---|---|---|---|

**1** GET (5) ⬭ S2KCOUNT

Stack (5)

| **38** | 15 | 25 | Locate File (5) |
|---|---|---|---|

**2** GET (5) ⬭ S2KCOUNT

Stack (5)

| 38 | **15** | 25 | Locate File (5) |
|---|---|---|---|

**3** GET (5) ⬭ S2KCOUNT

Stack (5)

| 38 | 15 | **25** | Locate File (5) |
|---|---|---|---|

**4** GET (5) ⬭ S2KCOUNT

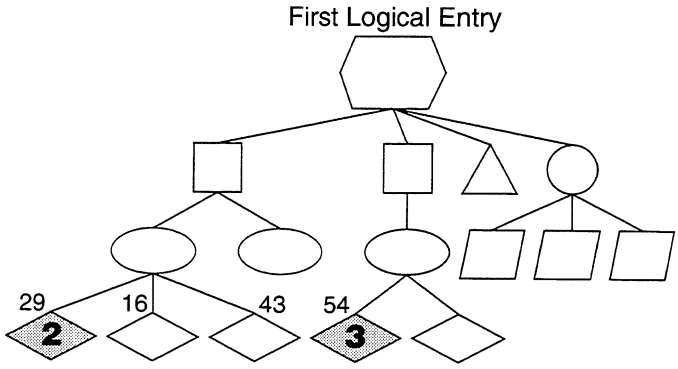
Stack (5)

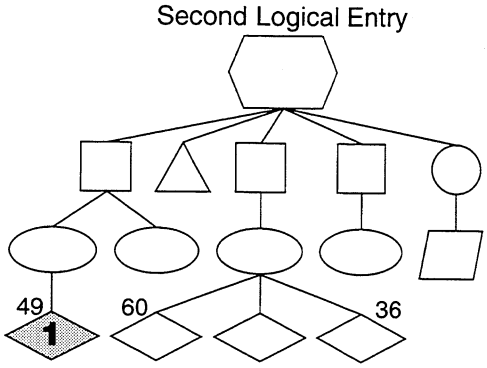
| 38 | 15 | **25** | Locate File (5) |
|---|---|---|---|

(return code 04: tried to retrieve past end of Locate File)

**Example 2**

When S2KCOUNT equals 2, a sequence of GET S2KCOUNT commands accesses every other address in the Locate File, starting with the second address.




MOVE 2 TO S2KCOUNT.
[COBOL Statement]

LOCATE (12) ◇ WHERE ...

| 16 | 29 | 36 | 43 | 49 | 54 | 60 | Locate File (12) |
|---|---|---|---|---|---|---|---|

ORDER BY (12) ...

| 43 | 49 | 36 | 29 | 60 | 54 | 16 | Locate File (12) |
|---|---|---|---|---|---|---|---|

**1** GET (12) ◇ S2KCOUNT

Stack (12)

| 43 | **49** | 36 | 29 | 60 | 54 | 16 | Locate File (12) |
|---|---|---|---|---|---|---|---|

**2** GET (12) ◇ S2KCOUNT

Stack (12)

| 43 | 49 | 36 | **29** | 60 | 54 | 16 | Locate File (12) |
|---|---|---|---|---|---|---|---|

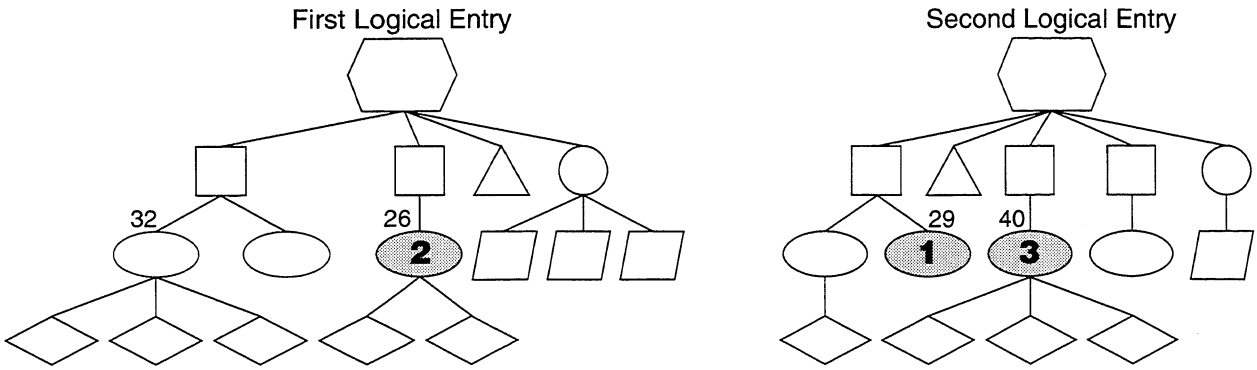
**3** GET (12) ◇ S2KCOUNT

Stack (12)

| 43 | 49 | 36 | 29 | 60 | **54** | 16 | Locate File (12) |
|---|---|---|---|---|---|---|---|

**Example 3**

In this situation, the program moved the value -1 to S2KCOUNT. The sequence of GET S2KCOUNT commands achieves the same result as a sequence of GET PREVIOUS commands.




MOVE -1 TO S2KCOUNT.
[COBOL Statement]

LOCATE ( ) WHERE ...

| 26 | 29 | 32 | 40 | Locate File (0) |
|---|---|---|---|---|

ORDER BY ...

| 32 | 40 | 26 | 29 | Locate File (0) |
|---|---|---|---|---|

**1** GET ( ) S2KCOUNT

Stack (0)

| 32 | 40 | 26 | **29** | Locate File (0) |
|---|---|---|---|---|

**2** GET ( ) S2KCOUNT

Stack (0)

| 32 | 40 | **26** | 29 | Locate File (0) |
|---|---|---|---|---|

**3** GET ( ) S2KCOUNT

Stack (0)

| 32 | **40** | 26 | 29 | Locate File (0) |
|---|---|---|---|---|

# INCLUSION LIST

This section is specific for COBOL and FORTRAN PLEX. For PL/I or Assembler PLEX information, see the appropriate appendix in this manual.

An inclusion list is optional syntax for retrieval or update commands. The list indicates the subschema items to be used for the operation.

## Format

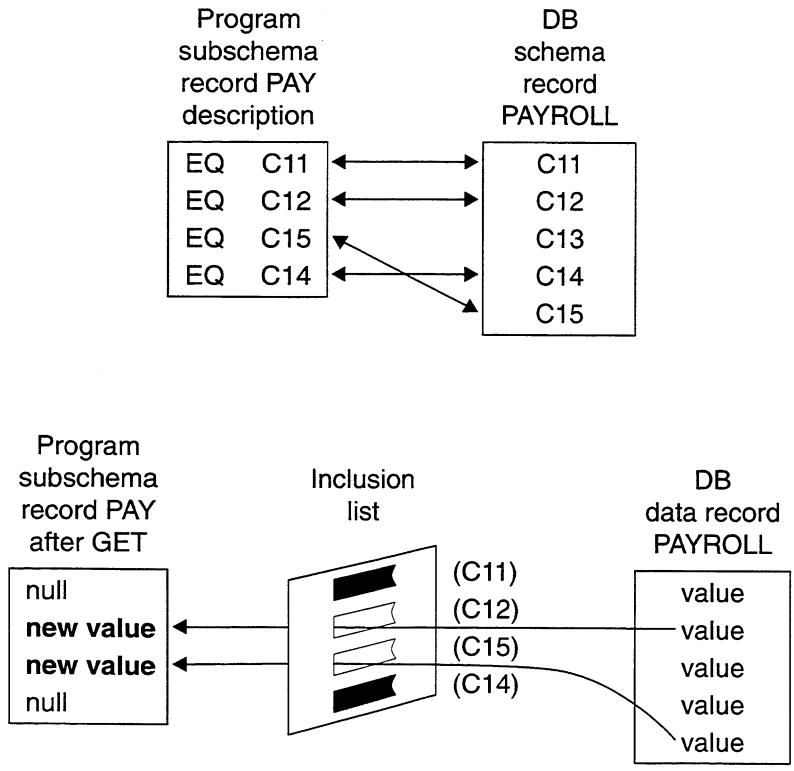
```
|item-list

|USING |name-table
       |word-table
```

A program can use an inclusion list with the retrieval commands GET1, GET, GETA, GETD, and GETP, and also with the update commands MODIFY, REMOVE, and INSERT. If such a command does not specify an inclusion list, all items in the subschema record are affected by the command.

But, when an inclusion list is specified, it indicates which items of the subschema record will be used in the operation. The process of retrieving values with an inclusion list is shown in the following illustration.

**Illus. 8.1** Retrieval with an Inclusion List



As Illus. 8.1 on page 8-121 shows, the subschema record description consists of four of the five items in the corresponding data record. Thus, four values could be retrieved by the program. But the inclusion list screens out two of those four values, allowing the other two to be passed to the subschema record.

You can specify any of three kinds of inclusion lists:

- *item-list*, which is a series of aliases that determine the subschema items that will be affected. You specify the items in the command syntax.
- *name-table*, which is a table, built by the program, that specifies the component names or C-numbers of subschema items that will be affected. A name table is usually an appropriate choice if the program needs to create an inclusion list based on component names. However, name tables are not as efficient as word tables.
- *word-table*, which is a table, built by the program, that specifies one-word indicators (containing 1 or 0) corresponding to all items in the subschema record. A 1 indicates that the item will be affected, and a 0 indicates that the item will not be affected. In each indicator, only the low-order bit is checked to see whether it is zero or 1. Word tables are the most efficient to process because SYSTEM 2000 software processes the tables positionally by indexing into the SSR.

The next topics discuss each kind of inclusion list.

## Item List

```
GET [/HOLD] [(m)] SSR |[item-list] |[NEXT]
                                   |[FIRST]
                                   |[LAST]
                                   |[PREVIOUS]
                                   |[PRESENT]
                                   |[S2KCOUNT]
```

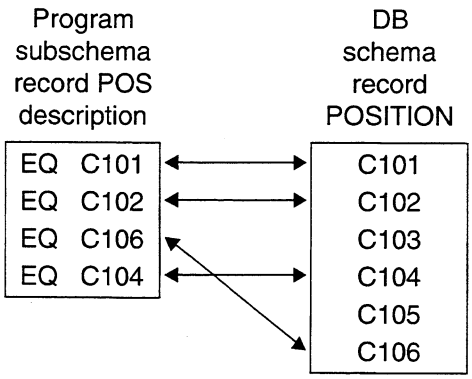
An item list is part of the command. It is a series of aliases, separated by commas, of subschema items to be included in the command's operation. If an item list includes an item that does not belong to the specified subschema record, the PLEX processor issues an error message. Since an item list is part of the command, it is static and cannot be changed. If you need to generate an inclusion list dynamically, use a name table or a word table.

The next examples show the items of the schema record called POSITION and the corresponding items of the subschema record called POS.

Four items are included in the subschema record, and, therefore, four values would ordinarily be retrieved from or modified in the corresponding data record. As the examples show, an item list included in the command qualifies the process.

### Example 1

This example shows a GET1 command with an item list. The correspondence between the subschema record POS and the schema record POSITION is shown below.




The command

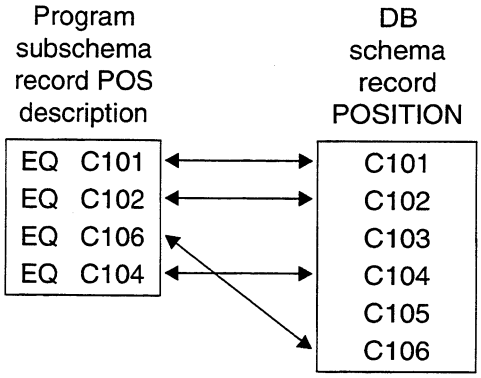
GET1 POS **C102, C106** NEXT.

results in the following subschema contents. Note that the subschema items not encoded in the command are null. SYSTEM 2000 software clears the subschema record to null before beginning the retrieval processing.

| | Program subschema record POS after GET1 |
|---|---|
| (C101) | null |
| (C102) | **new value** |
| (C106) | **new value** |
| (C104) | null |

**Example 2**

This example shows a REMOVE command with an item list. Assume the correspondence between the subschema record POS and the schema record POSITION is, again, as shown.




The command

```
REMOVE POS C102, C106.
```

results in the following data record contents. Note that only those items specified in the item list become null.

DB data record after REMOVE

| | |
|---|---|
| value | (C101) |
| **null** | (C102) |
| value | (C103) |
| value | (C104) |
| value | (C105) |
| **null** | (C106) |

## Name Table

```
GET [/HOLD] [(m)] SSR |[USING name-table] |[NEXT]
                                          |[FIRST]
                                          |[LAST]
                                          |[PREVIOUS]
                                          |[PRESENT]
                                          |[S2KCOUNT]
```

A name table is built by the program and subsequently named in the command, following the keyword USING. The name table contains the component name or C-number for each subschema item to be included in the command's operation. Aliases of subschema items can be used only if they are identical to the component names or component numbers. If a name table includes an alias that is not a component name or C-number, the command is ignored and return code 59 results.

One name table can be referenced by any number of commands and must be large enough for its maximum use to include an eight-word entry for each subschema item to be affected. If the name table does not specify each item in the SSR, it must include an element indicating the end of the list. This end-of-list element can fall anywhere in the table and must have a blank in the first byte of its first word. If the name table does not include such an element and does not specify all items in the SSR, the command is not executed and return code 59 results. When a COBOL program builds a name table, the OCCURS clause must specify PICTURE X(32).

Since a command refers to only one subschema record, the subschema items referred to in the table must come from only one subschema record.

Each component name or C-number in the table must be left-justified and blank-filled, without non-significant embedded blanks. The component names or C-numbers can be in any order in the table, but it is more efficient if they are in the order of variables given in the subschema record description.

If a C-number to be used in a name table was declared with the EQ option in the specified subschema record, that number must be stated in the table exactly as it was declared. For example, suppose the subschema record description included the following line:

```
EQ  C0001,
```

The C-number as stated in the name table should also be C0001, not C1.

The next examples show using a name table with retrieval and update commands.

**Example 1**

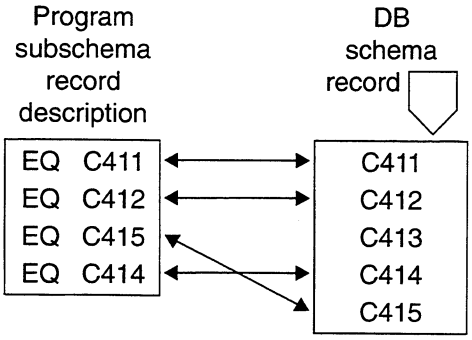
The following name table specifies three of the four subschema items:

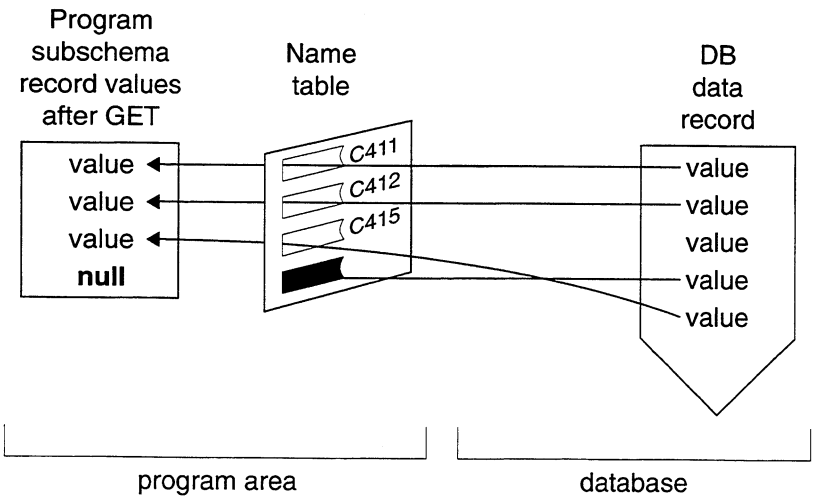
```
WORKING-STORAGE SECTION.
01  EDUC-NAME-TABLE.
    05  EDUC-NAME-ITEM OCCURS 4 TIMES PICTURE X(32).
      :
      :
PROCEDURE DIVISION.
      :
      :
   MOVE 'C411' TO EDUC-NAME-ITEM (1).
   MOVE 'C412' TO EDUC-NAME-ITEM (2).
   MOVE 'C415' TO EDUC-NAME-ITEM (3).
   MOVE SPACES TO EDUC-NAME-ITEM (4).
```

A subsequent retrieval command can refer to the name table.

```
GETD EDUC USING EDUC-NAME-TABLE NEXT.
```

The table EDUC-NAME-TABLE allows the values for subschema items C411, C412, and C415 to be brought into the subschema record area, without bringing the value for C414.



**Example 2**

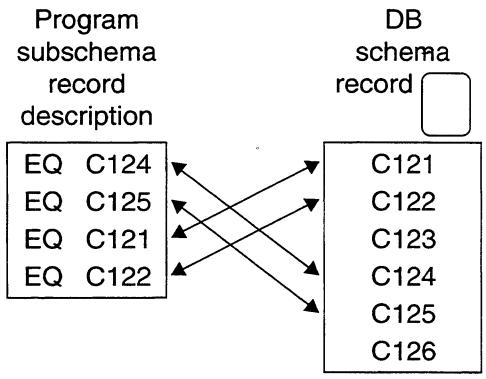
Similarly, for a MODIFY command, you can declare a name table and use it to change only two values, for example, in a data record.

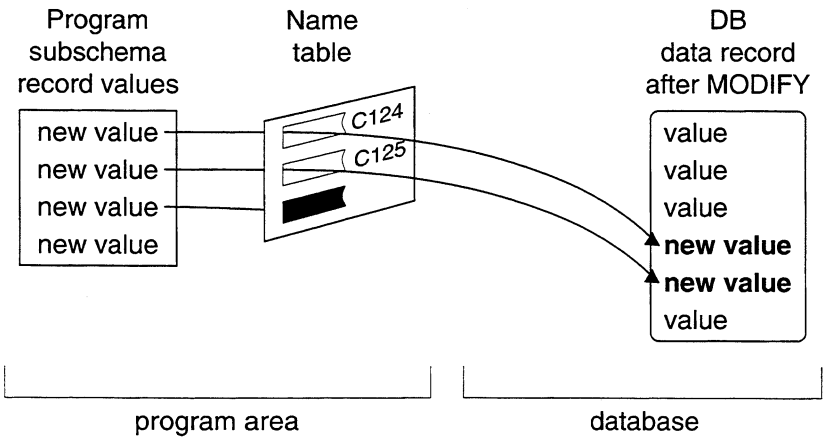
```
WORKING-STORAGE SECTION.
01  PAYROLL-NAME-TABLE.
    05  PAYROLL-NAME-ITEM OCCURS 4 TIMES PICTURE X(32).
      :
      :
PROCEDURE DIVISION.
      :
      :
   MOVE 'C124' TO PAYROLL-NAME-ITEM (1).
   MOVE 'C125' TO PAYROLL-NAME-ITEM (2).
   MOVE SPACES TO PAYROLL-NAME-ITEM (3).
```

Suppose the following MODIFY command refers to that name table.

```
MODIFY PAYROLL USING PAYROLL-NAME-TABLE NEXT.
```

The data record is modified according to the values in the SSR for items C124 and C125, which were specified in the name-table.



## Word Table

```
GET [/HOLD] [(m)] SSR |[USING word-table] |[NEXT]
                                          |[FIRST]
                                          |[LAST]
                                          |[PREVIOUS]
                                          |[PRESENT]
                                          |[S2KCOUNT]
```

Like a name table, a word table is built by the program and named in a command, following the keyword USING. It is a table of one-word indicators, one indicator for each item in the specified subschema record. Each word contains zero (0) or one (1). In each indicator, only the low-order bit is checked to see whether it is zero or one. An item is included in the operation if the value of its corresponding indicator in the table is one. Conversely, an item is excluded from the operation if its corresponding indicator is zero. For COBOL, the PLEX program must specify COMP usage for a word table.

The next examples show how word tables work for retrieval and update commands.

### Example 1

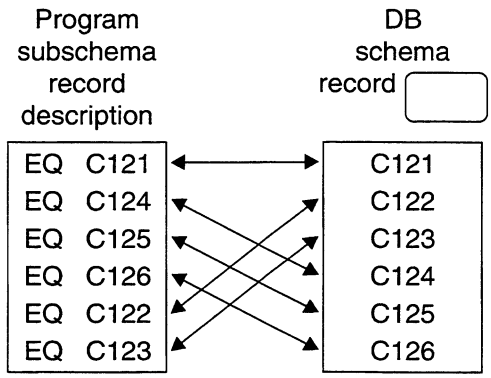
All six items in the SSR are reflected in the word table. The program places 1's in the elements corresponding to the data values to be retrieved, and 0's in the remaining elements. (Correspondence is relative to the order declared in the SSR.)

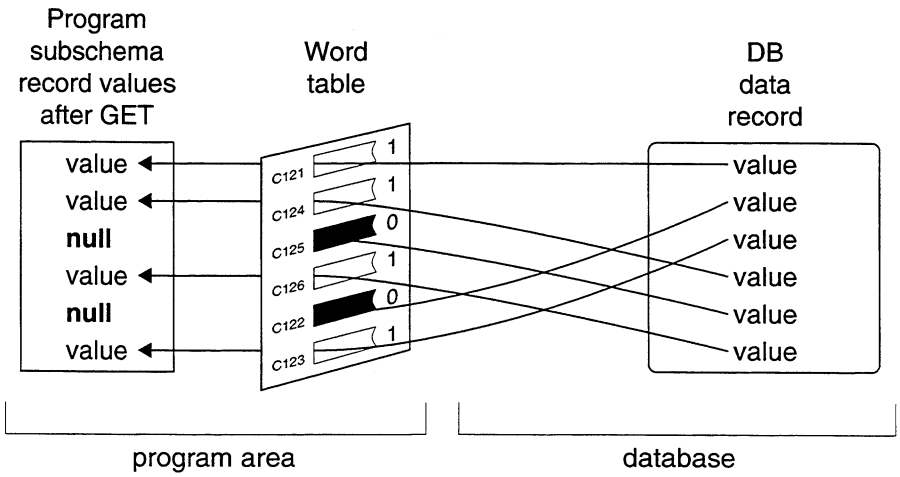
```
WORKING-STORAGE SECTION.
01  PAYROLL-WORD-TABLE.
    05  PAYROLL-WORD-ITEM OCCURS 6 TIMES PICTURE S9(9) COMP.
      :
      :
PROCEDURE DIVISION.
      :
      :
  MOVE 1 TO PAYROLL-WORD-ITEM (1).
  MOVE 1 TO PAYROLL-WORD-ITEM (2).
  MOVE 0 TO PAYROLL-WORD-ITEM (3).
  MOVE 1 TO PAYROLL-WORD-ITEM (4).
  MOVE 0 TO PAYROLL-WORD-ITEM (5).
  MOVE 1 TO PAYROLL-WORD-ITEM (6).
```

Suppose the following GETA command refers to that word table.

```
GETA PAYROLL USING PAYROLL-WORD-TABLE NEXT.
```

According to the specified word table, the values for subschema items C121, C124, C126, and C123 will be brought into the subschema record area, but the values for C125 and C122 will not, as shown in the following illustration.



**Example 2**

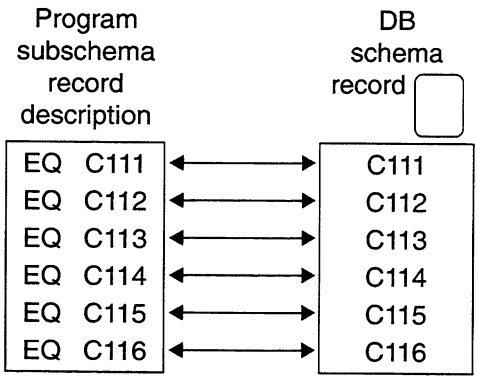
Similarly, for an INSERT command, you can declare a word table and use it to insert a new data record.

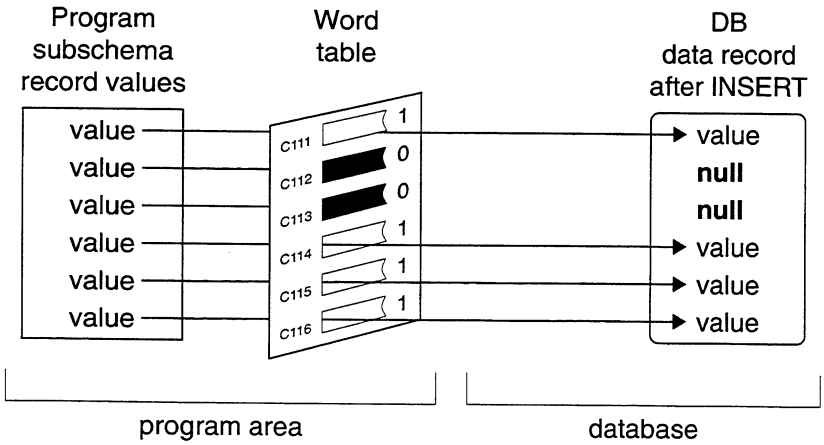
```
WORKING-STORAGE SECTION.
01  WORD-TABLE.
    05  WORD-ITEM OCCURS 6 TIMES PIC S9(9) COMP.
      :
      :
PROCEDURE DIVISION.
      :
      :
  MOVE 1 TO PAYROLL-WORD-ITEM (1).
  MOVE 0 TO WORD-ITEM (2).
  MOVE 0 TO WORD-ITEM (3).
  MOVE 1 TO WORD-ITEM (4).
  MOVE 1 TO WORD-ITEM (5).
  MOVE 1 TO WORD-ITEM (6).
```

A subsequent command such as

```
INSERT EMPL USING WORD-TABLE.
```

inserts only those values corresponding to the table elements that contain 1s.



# USING 0 OPTION

The USING 0 option is available in retrieval commands to indicate that only addresses are to be retrieved, no data. (Use in an update command results in an error message.)

A program can specify the USING 0 option in a GET1, GET, GETA, GETD, or GETP command. If a command does not specify the USING 0 option, two kinds of retrieval take place: retrieval of the address of the data record and retrieval of data from a data record. But if the USING 0 option is specified, only address retrieval takes place.

You can specify USING 0 to prevent data retrieval. Preceding an update operation, the program might need only to establish a stack entry to indicate where an update is to be performed. It might not be necessary in this situation to retrieve any data from the record for that stack entry.

Another way in which a program can cause address retrieval without data retrieval is to specify a subschema record that has no items. However, more flexibility is allowed with the USING 0 option, since you can use a subschema record that has items in some commands for data retrieval and updating and in other commands (with the USING 0 option) for address retrieval only.

### Format

```
GET [/HOLD] [(m)] SSR [USING 0]  |[NEXT]
                                 |[FIRST]
                                 |[LAST]
                                 |[PREVIOUS]
                                 |[PRESENT]
                                 |[S2KCOUNT]
```

### Examples

```
GET1 (13) SALARY USING 0 FIRST.
GET  (10) POSITION-REC USING 0.
GETD (5) SKILLS USING 0 S2KCOUNT.
GETA (8) PAYROLL USING 0.
```

# HOLD OPTION

A PLEX program that runs in a Multi-User environment can specify the HOLD option in a GET1, GET, GETA, GETD, GETP, or LINK command. HOLD signifies a local hold (a hold on a data record) or a global hold (a hold on the entire database), depending on the execution mode.

In immediate mode, the HOLD option signifies a *local hold*, which prevents other users from obtaining a global hold or obtaining local holds on the specific data record that is retrieved. A local hold applies only to a single data record, not to any of its descendants or ancestors. While your program has a local hold, other users are permitted to retrieve from and update

the database and even to obtain their own local holds on other data records in the same database. But those other users are not permitted to obtain local holds on the data record for which your program has the local hold.

In queue mode, by contrast, the HOLD option signifies a global hold, which prevents other users from updating or holding any record in the database. Other users are allowed to retrieve, but not to update, any data record in the database. However, those other users are not permitted to acquire local holds or global holds on the database.

## Format

```
GET [/HOLD] [(m)] SSR |[inclusion-list] |[NEXT]
                      |[USING 0]        |[FIRST]
                                        |[LAST]
                                        |[PREVIOUS]
                                        |[PRESENT]
                                        |[S2KCOUNT]
```

## Local Holds (/HOLD in Immediate Mode)

When a program requests a local hold (by using the HOLD option in immediate mode), it obtains the local hold when no other user has a local hold for the same data record or a global hold on the database, and when no other user is updating the database. Several users, then, can have local holds on different data records in the same database.

A local hold is obtained as a preliminary step to a MODIFY or REMOVE operation. A PLEX program can acquire local holds for records in more than one database at the same time. Moreover, a program can have multiple local holds on different records in the same database. The CONTROL language allows the DBA to enable multiple holds for a specific database through the ENABLE/DISABLE MULTIPLE HOLDS command. By default, the multiple hold capability for one database is disabled (see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*). Local holds remain in effect until one of the following events occurs in the PLEX program:

- a global hold is attempted for any database
- a FRAME command is issued for any database
- a synchpoint occurs
- the program terminates
- a DROP HOLD command occurs
- the QUEUE command is issued.

Once any of the above occurs, all of the PLEX program's local holds are dropped for all databases opened in the program. The only exception is where the DROP HOLD command specifies a particular database; in this specific situation, only the holds for that one database are dropped.

An update command does not cause a local hold to be dropped. That is, when a record is held for update processing and that processing completes, the record still has a local hold on it. Thus, for batches of PLEX update commands, many local holds on a database could accumulate. In a Multi-User environment with many users, the list of local holds can become very long and can involve considerable search time. Therefore, if many records are

being updated (and held) in immediate mode, consider using a FRAME command, a synchpoint, or queue mode, which causes a global hold. Or issue a command such as DROP HOLD to drop the local holds. Otherwise, all local holds are retained on all databases. The effect of a DROP HOLD command is the same as that of the COMMIT command.

SYSTEM 2000 software checks the lists of local holds whenever a user requests a local hold. If a possible deadlock is detected, return code 110 is issued to the user who is requesting a hold that could cause the deadlock. This means, for example, that if user A requests a record held by user B and user B then requests a record held by user A, user B receives return code 110, the retrieval request by user B is ignored, and all holds for user B are dropped. In other words, the records held for user B are no longer protected. For databases that have Rollback enabled, return code 110 causes an automatic rollback, as though the ROLLBACK command had been issued.

If a user requests a hold on a record being held by another user and no deadlock condition exists, the requesting hold is suspended out of thread until the hold on the record is dropped. No local hold lists are maintained when a user has exclusive use of the database or has a frame or global hold on the database.

Multiple local holds and the COMMIT command are less restrictive than global holds (with the FRAME/END FRAME commands). Therefore, especially in a Multi-User environment with several users needing to update a database at the same time, the MULTIPLE HOLDS option along with COMMIT should be considered rather than using global holds implied by FRAME/END FRAME.

In a Multi-User environment, users working with the same database in immediate mode often request retrieval or update operations whose execution would conflict with other operations already in progress for the database. SYSTEM 2000 software handles the contentions among users of the same database in a prescribed manner to protect the integrity of data in the database.

When a local hold is in effect, not only are retrievals allowed, but also updates and other local holds in those situations where they do not interfere with the local hold. Another program cannot request a global hold on a database for which one or more local holds are in effect.

If a non-key update operation (as described in item 3 of **Considerations** on page 8-137) is in progress for a certain database when a local hold is requested for the same database, the request is honored. But if a key update is being performed, the local hold is postponed until the update is completed. Similarly, if a global hold is in effect, the local hold is postponed until the global hold is ended. A requested global hold is postponed until after any current update operation completes or any current local hold or global hold ends.

Note that if two different users place local holds on, for example, a level 1 record and its parent, and if the parent is removed, the user who put the hold on the level 1 record receives a return code 09 upon attempting to reference the record. This situation can be avoided only by holding the record and its ancestors.

Illus. 8.2 on page 8-135 shows the disposition of requests for retrievals, holds, and updates in immediate mode. Depending upon the operation currently underway for a database, a request is either processed immediately or suspended until the current operation is completed.

## Global Holds (/HOLD Option in Queue Mode)

A program requests a global hold by using the HOLD option in queue mode. It obtains the global hold when no other user has a local hold or a global hold for the database, or is updating the database. A program can have only one global hold at a time; it cannot have global holds on two databases simultaneously.

A global hold can be obtained as a preliminary step to an update operation. The update operation is executed by the software after any current retrieval operation by another user (for the same database) has been completed.

In queue mode, a program can retain its global hold for an entire sequence of update commands. The global hold on a database is ended only when one of the following occurs in the program:

- a retrieval command without the HOLD option
- a retrieval command with the HOLD option that refers to another database
- the TERMINATE command, provided that the hold was retained until then.

For more details about how requests for retrievals, updates, and holds are handled in a Multi-User environment, see Illus. 8.3 on page 8-136. In this illustration, the current Multi-User activity is shown as a combination of operations, including retrievals and local and global holds. Three main types of requests that a PLEX program makes in a Multi-User environment are shown: a retrieval without a local hold, a local hold, and a global hold.

**Illus. 8.2** Disposition of Request for Retrievals, Holds, and Updates in Immediate Mode

| | **OPERATION UNDERWAY** | | | | |
|---|---|---|---|---|---|
| **NEW REQUEST** | Retrieval(s) | Local hold(s) | Global hold | Key update | Non-key update |
| Retrieval(s) | A | A | A | S | A |
| Local hold(s) | A | A* | S | S | A |
| Global hold | A | S | S | S | S |
| Key update** | S | A | S | S | S |
| Non-key update** | A | A | S | S | S |

A = allow request to be processed.
S = suspend request until active operation is completed.

---

* More than one local hold is allowed for a single database.

** If the Multi-User execution parameter LHOLD was set to YES (the default), each update command must be preceded by a command requesting a local hold. If not, the update command receives a return code 111.

**Illus. 8.3** Detailed View of Multi-User

| | Requested Operation | | |
|---|---|---|---|
| Current Activity | Retrieval without HOLD | Local hold in immed. mode | Global hold in queue mode |
| **Retrievals** | | | |
| No key update, No non-key<br>No local, No global | A | A | A |
| No key update, Non-key update<br>No local, No global | A | A | BC |
| No key update, Non-key update<br>Local holds, No global | A | BD | BCF |
| No key update, No non-key<br>Local holds, No global | A | BD | BF |
| No key update, No non-key<br>No local, Global hold | A | BE | BE |
| **No Retrievals** | | | |
| Key updates, No non-key<br>No local, No global | BC | BC | BC |
| No key update, Non-key update<br>No local, No global | A | A | BC |
| No key update, Non-key update<br>Local holds, No global | A | BD | BCF |
| No key update, No non-key<br>Local holds, No global | A | BD | BF |
| No key update, No non-key<br>No local, Global hold | A | BE | BE |

- **A** Allow request immediately
- **B** Postpone action
- **C** Until current update operation is completed
- **D** If requested local hold is for a record already under a local hold, until that current local hold is ended
- **E** Until global hold is ended
- **F** Until all current local holds are ended

**Note**: If the Multi-User execution parameter LHOLD is set to YES (the default), and if the request was not preceded by a local hold, the software issues return code 111 and ignores the request.

Once the appropriate combination of current Multi-User activity and requested operation is selected, the disposition of the request can be determined from the illustration. The disposition is expressed as one or more clauses, each assigned a letter from A to F.

After the software processes the current command, the program's action is no longer part of current activity, except for a hold established by the command.

## Considerations

1. If the HOLD option is specified in a LINK command, SYSTEM 2000 software executes the equivalent of a LOCATE command followed by a GET/HOLD FIRST command for the specified subschema. This implies a local hold on all records involved in a cascade.

2. In a Multi-User environment, the setting of the Multi-User execution parameter LHOLD affects the execution of update commands in immediate mode but not in queue mode. If LHOLD is set to YES (the default) and there is no global hold or exclusive use in effect, a PLEX program must obtain a local hold on a particular data record before giving a MODIFY or REMOVE command referring to that record in immediate mode. Also, a local hold is required for a MOVE TREE for the "from" stack and for the top node for a REMOVE TREE command. If the program does not first obtain the local hold, the update command is ignored and return code 111 results. If, on the other hand, LHOLD is set to NO, the program need not (but can) obtain a local hold before giving an update command in immediate mode.

3. The following discussion applies to immediate mode, not queue mode. A MODIFY or REMOVE command could refer only to non-key items in a data record. It does so if either 1) the specified subschema record contains only non-key items or 2) an inclusion list is used specifying only the non-key items in a subschema record. In a Multi-User environment, such a command can be executed in immediate mode while retrievals are being processed for the same database and while other users have local holds elsewhere in that database. This is possible in two situations:

   - when the Multi-User execution parameter LHOLD is set to YES and the program has already obtained a local hold for the data record to be updated, or

   - when LHOLD is set to NO.

     In either situation, retrievals and updates of non-key values should be coordinated, because a PLEX program might give a MODIFY or REMOVE command for a non-key value that another user expects to retrieve or update. For example, a PLEX program might change a non-key value in a record whose address is in a Locate File for another program. That other program, moreover, might need to retrieve or update the value that was changed.

     With LHOLD set to YES, if a program gives an immediate mode update command applying to one or more key items in a data record for which the program obtained a local hold, the execution of the command is postponed until

     - a current global hold ends

     - any retrieval or update operations currently in progress completes.

However, if LHOLD is set to YES and a program does not obtain a local hold for that data record beforehand, the command is ignored, and return code 111 results.

4. In immediate mode, a program can obtain a global hold on a database with a FRAME command. Otherwise, a global hold is obtained only with the HOLD option in queue mode.

5. The QUEUE command ends all current local holds for the program.

6. If a conflict arises between two users in which each user requests a local hold on the data record held by the other, it is resolved in the manner shown in Illus. 8.4 on page 8-139. This special consideration applies only to immediate mode, not queue mode.

7. A program can acquire a local hold on the parent or sibling of the top node of an insert sequence by giving a retrieval command with the HOLD option before it gives the first INSERT command in the sequence.

8. If the Rollback Log is enabled, queue mode holds establish a database synchpoint in addition to a global hold. A queue mode hold also establishes a user synchpoint, and updates performed by the user since the last user synchpoint, that is, logical unit of work (LUW), are committed (permanently recorded) in the affected databases. For details, see the ENABLE/DISABLE ROLLBACK, COMMIT, ROLLBACK, and RESET ROLLBACK commands. See also the Coordinated Recovery discussion in *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

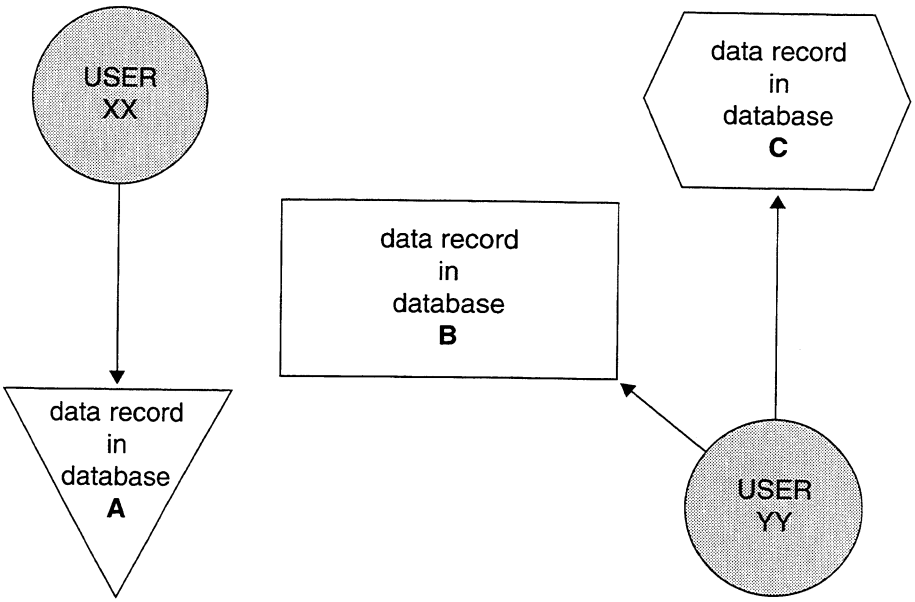
9. LHOLD equals YES is ignored if Rollback is enabled or if the program has a global hold or exclusive use of the database.

10. With Rollback enabled, SYSTEM 2000 software automatically places a local hold on every record being updated by a PLEX program, unless the user has exclusive use or a global hold. For a REMOVE TREE operation, a hold is also placed on every record being removed and on the node of detachment (parent or sibling node). For MOVE TREE, a hold is placed on the top node of the tree being moved, the node of detachment, and the node of attachment (parent or sibling node of insertion).

Illus. 8.4 on page 8-139 shows how SYSTEM 2000 software resolves a deadlock.

**Illus. 8.4** Resolving a Local Hold Deadlock

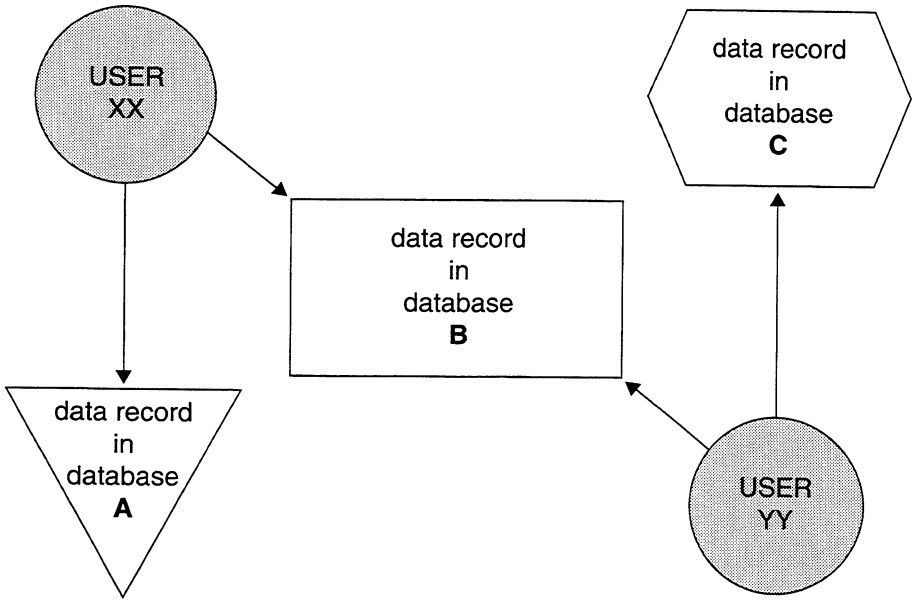
First, suppose two users, user XX and user YY, each have local holds in different databases.




*continued on next page*

**Illus. 8.4** *continued*

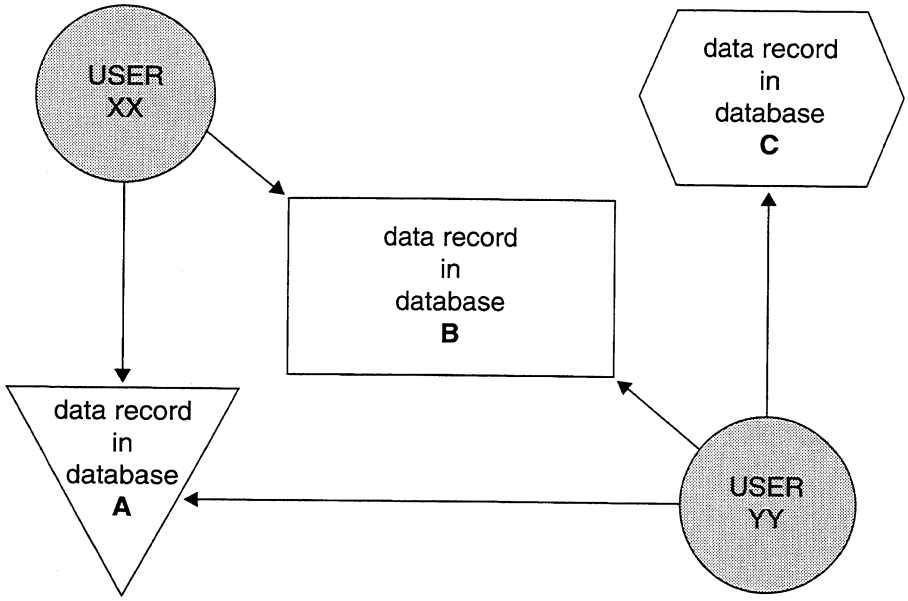
If user XX requests a local hold on a data record already held by user YY, the request is postponed, waiting until user YY's hold on that data record is ended.



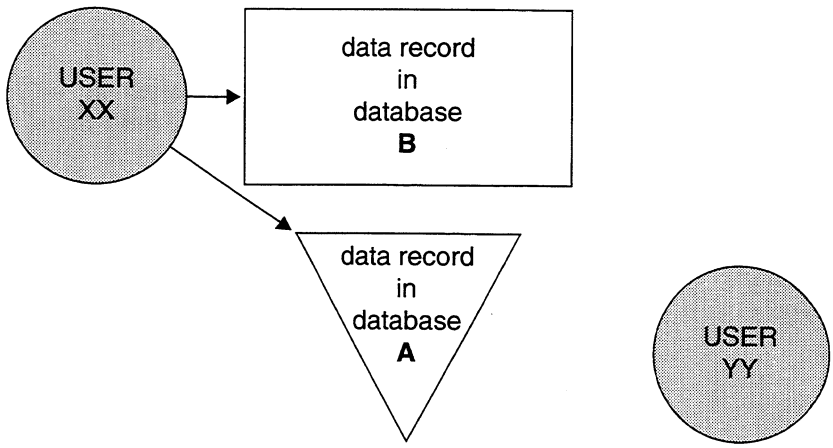

*continued on next page*

**Illus. 8.4** *continued*

Meanwhile, if user YY requests a hold on the data record already held by user XX, a deadlock results. Each user is requesting a hold on a data record held by the other.




Theoretically, user YY caused the deadlock, because user YY's request for a held record was subsequent to user XX's request. SYSTEM 2000 software resolves the deadlock by giving user YY, who caused the deadlock, a non-zero return code and dropping all local holds for that user. User XX then gains a local hold on the data record originally requested, now that it is no longer held by user YY.



## Example of Usage in Immediate Mode

Here is an example of immediate mode with multiple holds enabled.

```
WORKING-STORAGE SECTION.
      :
SSR POSITION-REC OF FIRSTDB.          (subschema record for FIRSTDB)
      :
SSR SALARY OF FIRSTDB.                (subschema record for FIRSTDB)
      :
SSR PLACE OF SECONDDB.                (subschema record for SECONDDB)
      :
PROCEDURE DIVISION.
      :
   GET1/HOLD POSITION-REC NEXT.       (request for retrieval and local hold on a
                                       POSITION-REC data record in FIRSTDB;
                                       local hold granted when a current global
                                       hold or local hold on that record expires,
                                       or when an in-progress update ends)

   GET1/HOLD SALARY NEXT.             (request for retrieval and local hold on a
                                       SALARY data record in FIRSTDB; previous
                                       local hold is still in effect since multiple
                                       local holds are allowed in one database)

   (command to update the SALARY      (After any other user's current retrieval
   record just retrieved)              has ended, the update is performed.)

   GET1/HOLD PLACE NEXT.              (request for retrieval and local hold on a
                                       PLACE data record in SECONDDB; local
                                       hold granted when a current global hold or
                                       local hold on that record expires, or when
                                       an in-progress update ends)

   INSERT commands to add a tree      (after any other user's current retrieval
    of records to SECONDDB.            has ended, the insertion of the tree is
                                       performed)

   GET1/HOLD SALARY NEXT.             (request for retrieval and local hold on a
                                       SALARY data record in FIRSTDB)

   GET1/HOLD PLACE NEXT.              (request for retrieval and local hold on a
                                       PLACE record in SECONDDB)

   GET1/HOLD SALARY NEXT.             (request for retrieval and local hold on a
                                       SALARY record in FIRSTDB; the local hold
                                       in SECONDDB is retained)
```

All records requested with a HOLD are still being held. Only a command causing a synchpoint (for example, COMMIT) can cause the holds to be dropped.

## Example of Usage in Queue Mode

Here is an example of using HOLD in queue mode.

```
WORKING-STORAGE SECTION.
      :
SSR POSITION-REC OF FIRSTDB.          (subschema record for FIRSTDB)
      :
SSR SALARY OF FIRSTDB.                (subschema record for FIRSTDB)
      :
SSR PLACE OF SECONDDB.                (subschema record for SECONDDB)
      :
PROCEDURE DIVISION.
      :
   QUEUE.                             (enter queue mode)

   GET1 POSITION-REC NEXT.            (retrieval without request for global hold
                                      on FIRSTDB)

   GET1/HOLD SALARY NEXT.             (retrieval with request for global hold)

   GET1 SALARY NEXT.                  (global hold expires; retrieval without
                                      request for global hold. If this command
                                      had specified HOLD, the global hold would
                                      have been retained)

   GET1/HOLD POSITION-REC NEXT.       (retrieval with request for global hold)

   (update command for the            (global hold retained)
   POSITION-REC record
   just retrieved)

   INSERT commands to add records     (global hold retained)
   to FIRSTDB

   GET1/HOLD PLACE NEXT.              (global hold for FIRSTDB expires; retrieval
                                      with request for global hold on
                                      SECONDDB)

   TERMINATE.                         (leave queue mode; global hold expires
                                      and the updates are executed)
```

# DROP HOLD

The DROP HOLD command ends the local holds for one database or all databases opened in a program.

In immediate mode, your program can obtain one or more local holds on specific data records. It can have multiple local holds on one database, and it can have local holds for several databases at one time. Your program can end the local hold(s) by giving a DROP HOLD command that specifies either a specific database or the keyword ALL. When the command DROP HOLD ALL is given, all current local holds for all databases for the program are ended. Also, local holds for all databases are dropped when a synchpoint occurs, a FRAME is issued, the program terminates, a global hold occurs, or a QUEUE command is given. All DROP HOLD commands behave like a COMMIT command. That is, even a DROP HOLD for a specific database behaves as a DROP HOLD ALL and causes a synchpoint the same as COMMIT. (The DROP HOLD commands are retained for program compatibility only.) The COMMIT command should be used in all new applications.

## Format

```
DROP HOLD |database-identifier
          |ALL
```

*database-identifier* is the name of a SYSTEM 2000 database or name of the COMMBLOCK.

## Examples of Syntax

```
DROP HOLD EMPLOYEE.
```
*(specifies the EMPLOYEE database)*

```
DROP HOLD ALL.
```
*(specifies all holds for all databases for which the program has local holds)*

## Example of Usage

This example illustrates the DROP HOLD ALL command.

```
WORKING-STORAGE SECTION.
      :
SSR PARTS OF INVENTORY.          (subschema record PARTS for the
                                  INVENTORY database)
      :
SSR ENTRY-REC OF EMPLOYEE.       (subschema records ENTRY-REC and
                                  SALARY for the EMPLOYEE database)
      :
SSR SALARY OF EMPLOYEE.
      :
SSR PLACE OF DISTRIBUTION.       (subschema record PLACE for the
                                  DISTRIBUTION database)
      :
PROCEDURE DIVISION.
      :
(Program is in immediate mode.)

   GET1/HOLD (6) PARTS NEXT.     (retrieves and acquires a local hold on a
                                  PARTS data record in the INVENTORY
                                  database)

   GET1/HOLD (5) PLACE NEXT.     (retrieves and acquires a local hold on a
                                  PLACE data record in the DISTRIBUTION
                                  database)

   GET1 ENTRY-REC  NEXT.         (retrieves an ENTRY-REC data record from
                                  the EMPLOYEE database)

   GET/HOLD SALARY NEXT.         (retrieves and acquires a local hold on a
                                  SALARY data record in the EMPLOYEE
                                  databases)

(PLEX commands for additional databases)

   DROP HOLD ALL.                (ends all local holds and commits the
                                  updates)
```

## Considerations

- The software issues return code 00 for a successful DROP HOLD command. It is sent to the variable S2KRTC and, if a database was specified, to the COMMBLOCK of that database.

- The QUEUE, FRAME, and LOAD commands or the occurrence of a synchpoint end all current local holds for a program.

- Since local holds are acquired only in immediate mode, the DROP HOLD command is intended for immediate mode only. Thus, in queue mode, if a program has a global hold on a database that it names in a DROP HOLD command, the global hold is retained and the command has no effect. But if the program names a different database in a DROP HOLD command, the global hold is ended.

- If a DROP HOLD command refers to a database for which the program has declared a COMMBLOCK but has not yet opened, the database is opened automatically.

- The DROP HOLD command establishes a user synchpoint. User synchpoints are used by the Coordinated Recovery feature when the Rollback Log is enabled. A user synchpoint causes any updates (logical unit of work (LUW)) to databases since the issuing user's last user synchpoint to be permanently recorded (committed) in the affected databases. For details see the ENABLE/DISABLE ROLLBACK, COMMIT, ROLLBACK, and RESET ROLLBACK commands. See also the Coordinated Recovery discussion in *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

- The DROP HOLD command behaves like a COMMIT command. Also, even if a specific database is given in a COMMIT command, it still behaves as a COMMIT (that is, as a DROP HOLD ALL) and causes a synchpoint. The DROP HOLD syntax is retained for program compatibility, that is, all DROP HOLD commands behave exactly like a COMMIT command.

# COPYS AND COPYL

The COPYS command duplicates a stack or cancels its entries. Similarly, the COPYL command duplicates a Locate File or cancels all addresses in it. COPYS and COPYL additionally have the following features:

- You can specify any one or two of 16 stacks.
- You can specify any one or two of up to 16 Locate Files.
- You can use variables to specify the stacks or Locate Files.

A COPYS command changes a specific stack, called the target stack, without the use of PLEX retrieval or update commands. A specific Locate File, called the target Locate File, can likewise be changed with a COPYL command. Using these commands, a PLEX program can make a copy (the target) of a source stack or Locate File and then process the source and target in different ways. However, the addresses in a stack or Locate File are canceled if a COPYS or COPYL command specifies NULL in place of a source stack or Locate File number. The same result is obtained when the program specifies a negative value for the source stack or Locate File number.

## Format

```
COPYS |(source-stack) TO (target-stack).
      |NULL

COPYL |(source-Locate-File) TO (target-Locate-File).
      |NULL
```

| | |
|---|---|
| *source-stack* | is the number of the stack to be copied: a constant or a variable. If the value for the source stack is negative, the result is the same as for specifying NULL. |
| *target-stack* | is the number of the stack to be changed by the command: a constant or a variable. |
| *source-Locate-File* | is the number of the Locate File to be copied: a constant or a variable. If the value for the source Locate File is negative, the result is the same as for specifying NULL. |
| *target-Locate-File* | is the number of the Locate File to be changed by the command: a constant or a variable. |

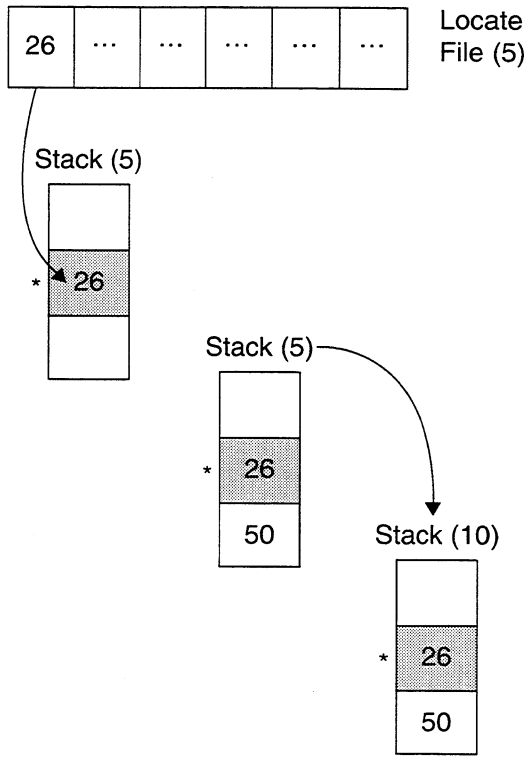
## Examples of Syntax

| | |
|---|---|
| `COPYS (3) TO (14).` | *(specifies copying the contents of stack 3 to stack 14)* |
| `COPYL (VAR1) TO (6).` | *(specifies copying to Locate File 6 the contents of the Locate File whose number is the value of* VAR1, *provided the value is positive. If VAR1 is negative, the addresses in Locate File 6 are canceled)* |
| `COPYS (8) TO (VARX).` | *(specifies copying the contents of stack 8 to the stack whose number is the value of* VARX, *provided the value is positive. If VARX is negative, the command is ignored and return code 40 results)* |
| `COPYL (VAR6) TO (VAR3).` | *(specifies copying of the contents of one Locate File to another, using the values of* VAR6 *and* VAR3. *If* VAR6 *is negative, the addresses in Locate File* VAR3 *will be canceled. If* VAR3 *is negative, the command is ignored and return code 40 results)* |
| `COPYS NULL TO (7).` | *(specifies* NULL, *meaning that the addresses in stack 7 are to be canceled)* |
| `COPYL NULL TO (VAR5).` | *(specifies* NULL, *meaning that the addresses in Locate File* VAR5 *are to be canceled)* |
| `COPYS (-3) TO (6).` | *(specifies a negative value for the source stack, meaning that the addresses in stack 6 are to be canceled)* |

## Examples of Usage

This example includes the COPYS command.

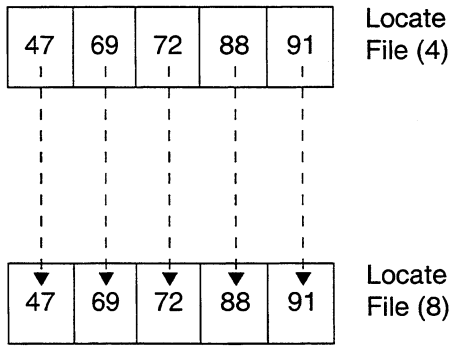
```
START S2K (10).
      :
LOCATE (5) . . .
      :
GET (5) . . . NEXT.
      :
GETD (5) . . . NEXT.
      :
COPYS (5) TO (10).
```




This example includes the COPYL command, and the target Locate File number is specified in a variable.

```
WORKING -STORAGE SECTION.
      :
01  LOCVAR. . .
      :
PROCEDURE DIVISION.
      :
   LOCATE (4). . .
      :
   MOVE 8 TO LOCVAR.

   COPYL (4) TO (LOCVAR).
```



## Considerations

1. Check the return code after a COPYS or COPYL command is given. This return code is placed in the variable S2KRTC, which is part of the S2KDUM control block.

2. For COBOL, variables used to represent stack and Locate File numbers in the COPYS and COPYL commands can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

3. After a COPYL command is executed, the first GET command referring to the target Locate File causes three COMMBLOCK fields to be set prior to execution of the GET command. These fields are

   - the fourth field, which contains the flag for retrieval of the record at the last address. It is set to 0.

   - the sixth field, which contains the number of record addresses in the Locate File.

   - the seventh field, which contains the Locate File pointer. The ordinal position of the address of the most recently retrieved record is set to 0.

     Execution of the GET command changes these values.

4. If the value of the source is negative in a COPYS or COPYL command, the addresses are canceled in the target. The same result is obtained when you specify NULL instead of a source number.

5. If the source stack or Locate File is empty, addresses in the target stack or Locate File are canceled.

6. A COPYS command is ignored and return code 40 results if the value for the source or target stack exceeds 15, since stack numbers range from 0 to 15.

7. If the value for the source or target Locate File in a COPYL command exceeds the subscript in the previous START S2K command, the COPYL command is ignored, and return code 40 results.

8. If the value for the target in a COPYL or COPYS command is negative, the command is ignored, and return code 40 results.

9. A COPYS or COPYL command can cause a target stack or Locate File to be used for a database other than the one for which it was already used by the program. A change from one database to another cancels addresses in the Locate File or stack with the same number as the target, since stacks and Locate File with the same number are always used for the same database.

10. The COPYS command copies ten adjacent levels of the source stack to the target stack. The shallowest level stack entry is the first of the ten copied entries. If the source stack has entries scattered over more than ten adjacent levels, only ten levels are copied to the target, starting from the shallowest entry in the source stack, and return code 69 results. If the primary position in the source stack is not at one of the ten levels to be copied to the target, it will not be part of the target stack.

# GETP

The GETP command (get position) establishes a primary stack position based on values in the Index. The specified value can exist in the database as a single occurrence or as a multiple occurrence, or it might not exist at all. GETP is very efficient when you need to insert a record sequentially within a set of siblings.

- GETP reduces execution time because it uses a unique or near-unique value from the single, key condition where-clause to access the Index. It does not have to build and manipulate lists as in normal where-clause processing.
- You can retrieve a data record from any point, at any level, in the database.
- You can establish a primary stack entry even if a specified value does not exist.
- You can specify any one of sixteen stacks.
- You can retrieve a data record address without retrieving data values, with the USING 0 option.
- You can retrieve values from specific (rather than all) items in a subschema record, using an inclusion list.

The GETP command retrieves a data record for the specific purpose of establishing a desired position in the database. GETP requires a single key condition where-clause. Although the specified value need not be unique for record retrieval, the command is intended for positioning based specifically on a single occurrence value.

The GETP command accesses the Index to select a record that satisfies the key condition. Standard where-clause processing, which involves building and manipulating lists of qualified records, is avoided.

Because the command avoids normal where-clause processing, GETP retrieval is more rapid than retrieval using a GET1 command with a where-clause. For example, suppose you want to insert a new data record below level 0, in a sibling chain consisting of a large number of records. You could use GETP to position immediately ahead of the insertion point, followed by INSERT AFTER.

To retrieve the required record, the sibling chain values must be arranged in a sequential order appropriate to your application. You must assign data values correctly so that the key value selected in the Index is the value that you want selected. For example, values can be grouped together in the Index using a concatenated key, thus ensuring proper positioning. Example 5 illustrates graphically the importance of this requirement.

Although execution time involving multiple occurrence values is slightly higher than execution time involving a single occurrence value, execution time does not increase according to the number of occurrences. In other words, for multiple occurrences, execution time is the same regardless of whether a value occurs twice or two hundred times. However, you have no choice of which record is retrieved when the specified value is not unique. Always, the record retrieved is the one containing the value that both meets the where-clause condition and occurs first in the Index for the specified item.

The subset of binary operators permitted in the where-clause condition works differently for GETP than for other retrieval commands. The operators allow you to establish a primary stack entry even if the specified value is not in the database as long as at least one value exists for the specified subschema item. Valid operators are restricted to LT, LE, EQ, GT, or GE, or the equivalent logical symbols. For the GETP command only, these operators are processed as follows:

LT: retrieves the record that contains the value that is next lower than the value specified in the key condition.

If no values in the database are lower than the value specified in the key condition, GETP retrieves the record with the lowest value in the database for the specified item. This results in a retrieved record with a value equal to or greater than the specified value, and return code 50 is issued.

LE: retrieves the record that contains the value specified in the key condition.

If no equal value exists, GETP retrieves the record that contains the next lower value.

If no values are equal to or lower than the value specified in the condition, GETP retrieves the record with the lowest value in the database for the specified item. This results in a retrieved record with a value greater than the specified value, and return code 50 is issued.

EQ: retrieves the record that contains the value specified in the key condition.

GT: retrieves the record that contains the value that is next higher than the value specified in the key condition.

If no values in the database are higher than the value specified in the key condition, GETP retrieves the record with the highest value in the database for the specified item. This results in a retrieved record with a value equal to or lower than the specified value, and return code 50 is issued.

GE: retrieves the record that contains the value specified in the key condition.

If no equal value exists, GETP retrieves the record that contains the next higher value.

If no values are equal to or greater than the value specified in the condition, GETP retrieves the record with the highest value in the database for the specified item. This results in a retrieved record with a value lower than the specified value, and return code 50.

**Note**: You cannot specify NE or its logical symbols with GETP.

## Format

```
GETP [/HOLD] [(stack-number)] SSR |[inclusion-list]
                                  |[USING 0]
   WHERE single-key-condition
```

*stack-number* is the number of the stack to be used in retrieval: a constant or a variable whose value is greater than or equal to zero and less than or equal to 15, that is, $0 \leq n \leq 15$. The default is 0. For COBOL, the variable can have DISPLAY, COMP-3 or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*SSR* is the name of the subschema record for the data record that is to be retrieved.

*inclusion-list*

```
|item-list

USING |name-table
      |word-table
```

*item-list*

```
item-name  [, item-list]
```

*item-name* is the alias of an item in the specified subschema record.

*name-table* is the name of a table containing component names or C-numbers for items to be included in the retrieval operation. For COBOL, the entry with the OCCURS clause must specify PICTURE X(32).

*word-table* is the name of a table of one-word indicators (1s and 0s), one indicator for each item in the specified subschema record. An item is included in the retrieval operation if its indicator in the table is one. Conversely, an item is excluded from the operation if its indicator is zero. For COBOL, the entry with the OCCURS clause must specify COMP usage.

*single-key-condition*

```
item-name  operator  variable
```

*item-name* is the name of a subschema item corresponding to a schema item that is key. It must be a member of the schema record represented by the subschema record in the action-clause.

*operator* is one of the following: LT, LE, EQ, GT, GE, (or the symbolic equivalents).

*variable* for COBOL, is a variable name of up to 30 characters. It can be in a subschema record. The value assigned to the variable should be unique. See **Item Types and COBOL Data Descriptions of Items in Where-clauses** on page 4-7 for COBOL data descriptions of items in where-clauses.

the short form of the command and its equivalent

```
GETP SSR WHERE item-name operator variable

GETP(0) SSR WHERE item-name operator variable
```

## Examples of Syntax

```
GETP SAL WHERE PAY-SCHEDULE EQ
 MONTHLY.
```
(PAY-SCHEDULE *is a key item contained in the schema record represented by* SAL.)

```
GETP WORK WHERE EMPLOYEE-NUMBER
 EQ NUM.
```
(*The value for* NUM *should be unique in the database for the specified item.*)

```
GETP/HOLD(5) PAYROLL WHERE C121
 GE DATE.
```
(*specifies* HOLD *and stack*)

```
GETP(2) POS USING 0 WHERE START-
 DATE LT TWOYEARS.
```
(*specifies address retrieval only, since positioning is all that is required*)

## Example of Usage

```
WORKING-STORAGE SECTION.
      :
      :
COMMBLOCK OF EMPLOYEE.
      :
      :
SSR SKILLS OF EMPLOYEE.
```
(*Subschema record corresponds to* JOB SKILLS *schema record.)*

```
01 SKILLS.
    03  SKILL-TYPE . . .
        EQ C201.
    03  YEARS-OF-EXPERIENCE . . .
        EQ C203.
      :
      :
PROCEDURE DIVISION.
      :
      :
   MOVE PL/I TO PLI.
      :
      :
   GETP SKILLS WHERE SKILL-TYPE
   EQ PLI.
```
(*assumes that the value of the variable* PLI *is unique for* SKILL-TYPE*; retrieves the one and only data record that contains a* C201 *value equal to* PL/I)

```
   INSERT SKILLS AFTER.
```
(*adds a new* JOB SKILLS *data record with subschema items* SKILL-TYPE *and* YEARS-OF-EXPERIENCE *only, immediately after* JOB SKILLS *data record that has* C201 *equal to* PL/I)

## Considerations

1. Check the return code after each GETP command. The command's return code is placed in the second field of the COMMBLOCK for the database to which the command refers.

2. For the GETP command only, the fourth COMMBLOCK field (last-record flag) is

   - set to 1 if the database contains a value that equals the value specified in the condition, regardless of which operator is used (LT, LE, EQ, GT, or GE).

   - set to 0 if the database does not contain a value that equals the value specified in the condition, regardless of which of the allowed operators is used.

3. For the GETP command only, the sixth field of the COMMBLOCK (number of records) is

   - set to 1 if the value occurs only once in the database for the specified subschema item.

   - set to 0 if the value has multiple occurrences for the specified subschema item.

4. The seventh COMMBLOCK field is not used when a GETP command is executed.

5. GETP is intended for use with a unique value, but it does allow multiple occurrences. If the specified value has multiple occurrences, SYSTEM 2000 software retrieves the record containing the value that both satisfies the condition and occurs first in the Index for the specified item. Concurrently, SYSTEM 2000 software sets the sixth field of the COMMBLOCK to 0.

6. SYSTEM 2000 software issues a non-zero return code for GETP in the following situations:

   - The database did not contain any values that equal the value specified in a where-clause that contained the EQ operator (return code 04).

   - The GETP command was issued, but Security by Entry (when in force) was not established in a previous where-clause or the value selected by GETP was not contained in the specified entry (return code 29).

   - The database value was less than the where-clause GT or GE value but was the highest value in the database, or the value was greater than the LT or LE value but was the lowest value in the database. In either case, a data record is retrieved, and a primary stack entry is created (return code 50).

   - The subschema item corresponds to a non-key schema item (return code 68).

   - The subschema item was not a member of the schema record represented by the subschema record in the action-clause (return code 18).

   - The database contained only null data items for the specified item (return code 45).

- The program's current password lacked R-authority for subschema item retrieval, or the password lacked W-authority for the item mentioned in the where-clause (return code 46).

7. A PLEX processor message is issued if you specify an invalid operator in the where-clause or if you specify more than one condition.

8. Multiple occurrences increase execution time slightly in comparison with a unique value. However, execution time is the same regardless of how many times a value occurs.

9. If the MULTIPLE HOLDS option is enabled, your program can request any number of local holds on different records simultaneously. The MULTIPLE HOLDS option is disabled by default. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for details about setting the MULTIPLE HOLDS option. Local holds remain in effect until one of the following events occurs in the PLEX program:

   - a global hold is attempted for any database
   - a FRAME command is issued for any database
   - a synchpoint occurs
   - the program terminates
   - a DROP HOLD command occurs
   - the QUEUE command is issued.

   Once any of these occurs, all of the PLEX program's local holds on the database(s) are dropped. Also, an update command no longer causes a local hold to be dropped.

   See **HOLD Option** on page 80 for a complete discussion of local holds. See also **HOLD Option** on page 8-131 and **DROP HOLD** on page 8-144.

The following examples show a few of the different retrieval possibilities with GETP. In all situations, GETP establishes a primary stack entry.

**Example 1:** when the condition operator is EQ (or =).

**Example 2:** when the condition operator is GE (or ≥).

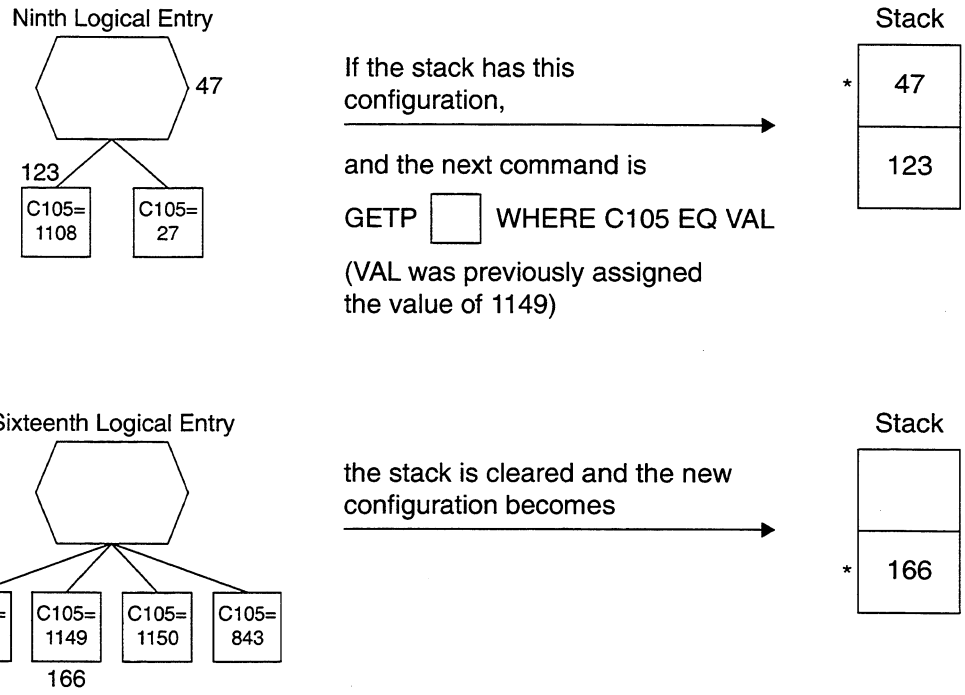
**Example 3:** when the condition operator is LE (or ≤).

**Example 4:** when a record is to be inserted sequentially in a sibling chain.

**Example 5:** shows that the ordering of values is crucial for correct retrieval.

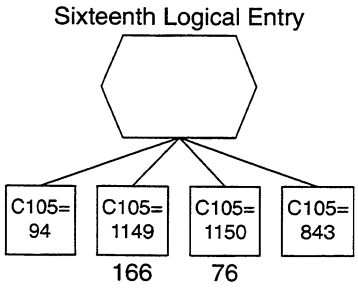
**Example 1**

When the condition contains the EQ (or =) operator, SYSTEM 2000 software searches among specified item values for the one value that equals the right operand value, as shown below.

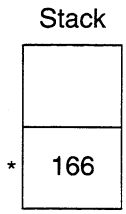


**Example 2**

When the condition contains the GE operator (or ≥) , SYSTEM 2000 software searches among specified item values for the one value that equals the correct operand value. If no equal value exists, SYSTEM 2000 software searches for the next higher value, if none, see next page. In the example below, the record that contains the first occurrence of C105=1150 is retrieved.

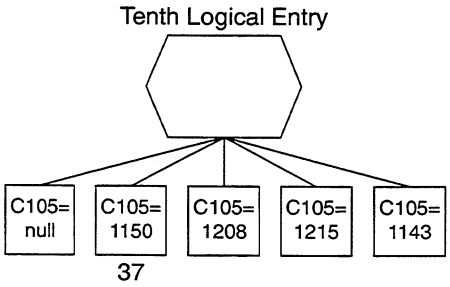



stack configuration →



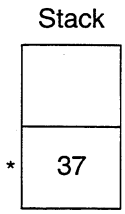

GETP [ ] WHERE C105 GE VAL

(VAL was previously assigned the value of 1150)



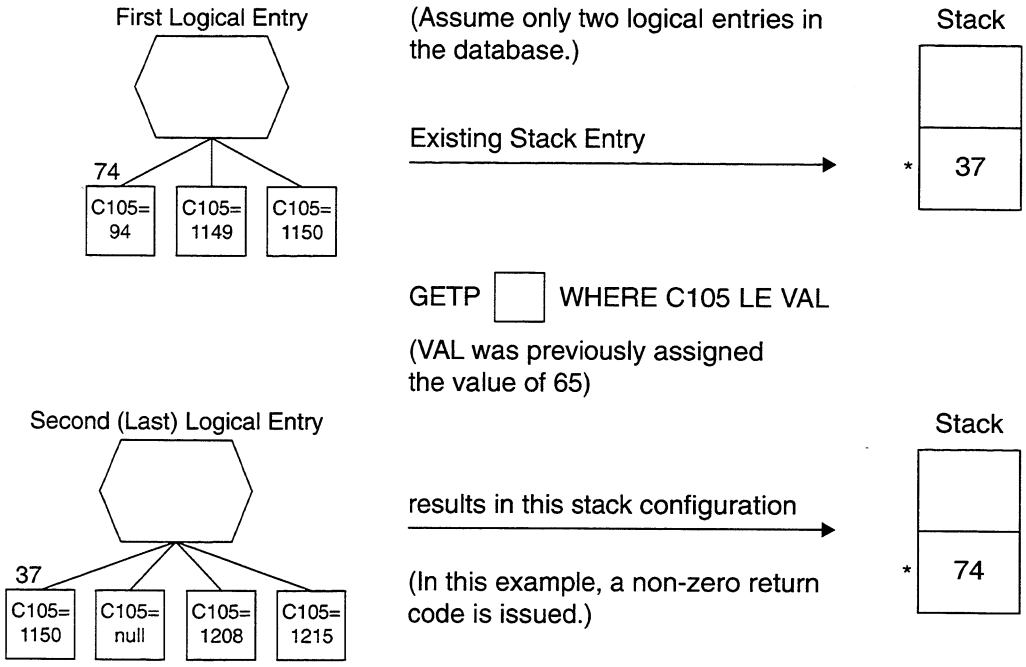

results in this stack configuration →

(In this example, the sixth field of the COMMBLOCK is set to "1", indicating multiple occurences.)



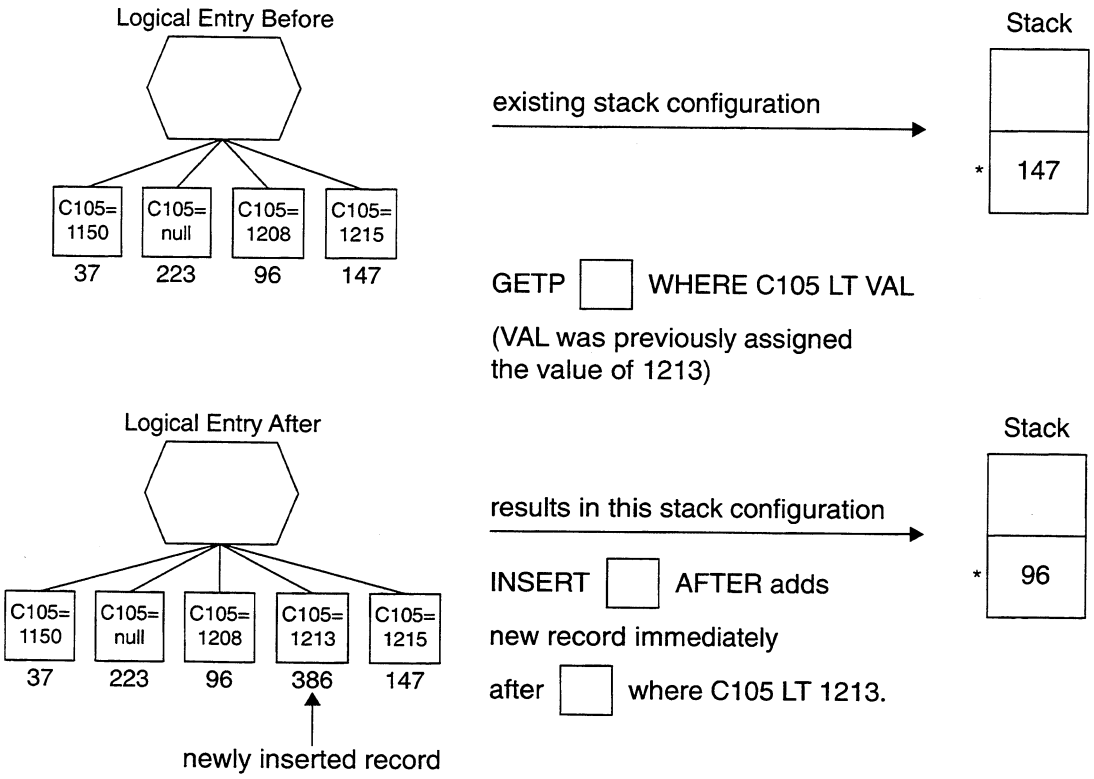
### Example 3

When the condition contains the LE (or ≤) operator, SYSTEM 2000 software searches among specified item values for the one value that equals the correct operand value. If no equal value exists, SYSTEM 2000 software searches for the next lower value. If no lower value exists, SYSTEM 2000 software selects the record with the lowest value in the database for the specified item, and issues return code 50. In the example below, the record retrieved has a C105 value greater than 65.



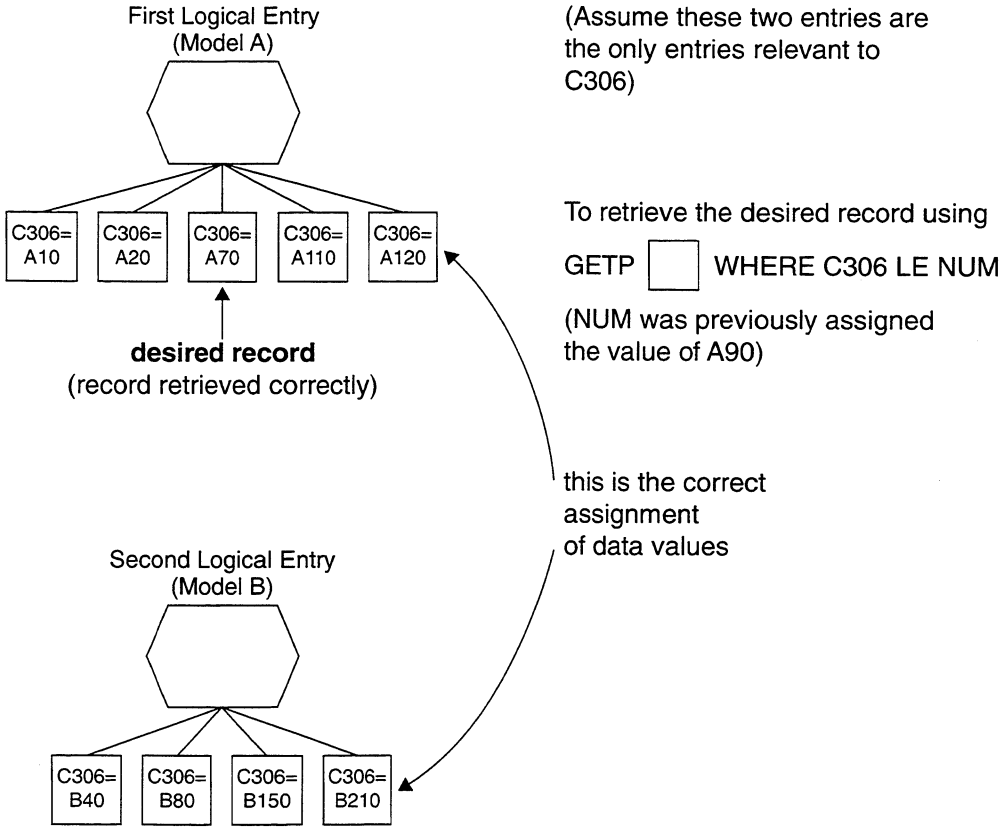
**Example 4**

With GETP, you can position your program logic anywhere in the database regardless of whether a specified value exists. This is useful when you need to insert a record in a sequence, especially with INSERT AFTER.



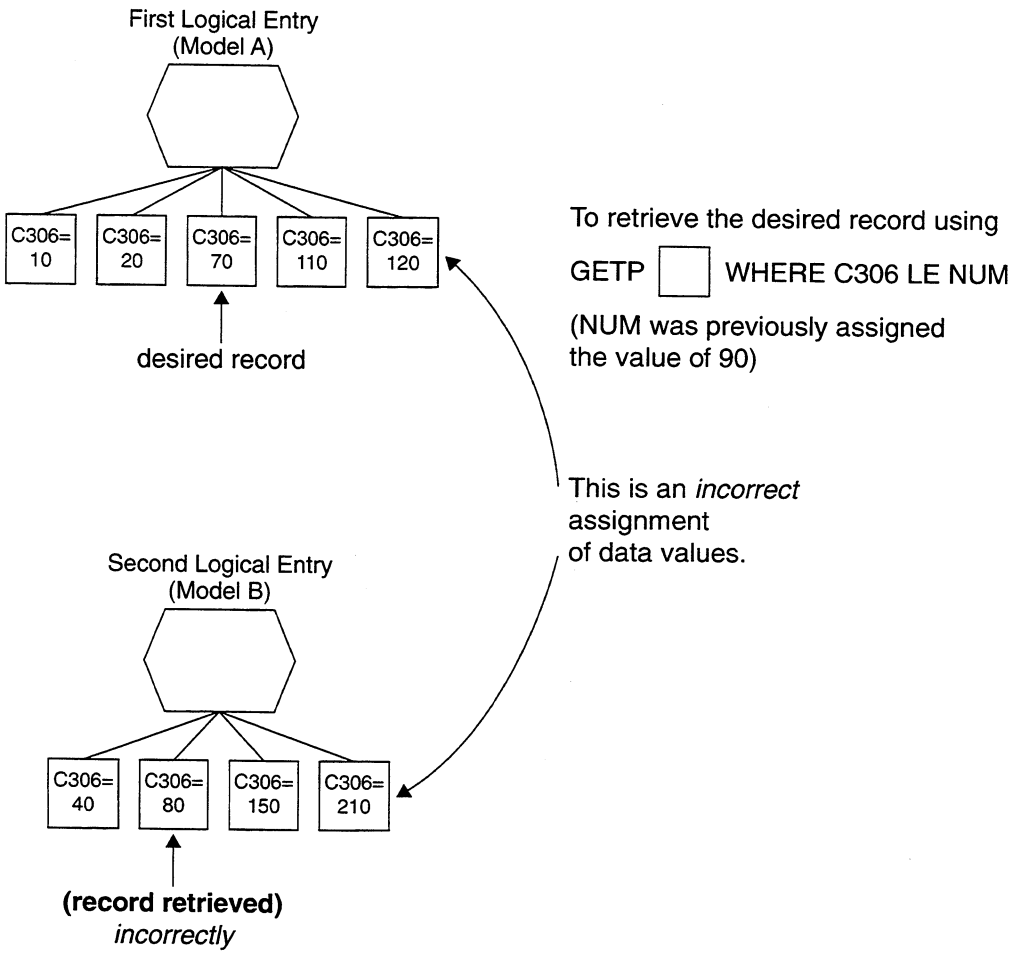
**Example 5**

Correct retrieval requires an appropriate ordering of values in the Index. In the example below, assume that the two entries represent two different car models (Model A and Model B). The C306 values are identification numbers for automotive parts. In this example, a concatenated key ensures correct retrieval.



**Example 5 (continued)**

In the illustration below, an incorrect record is retrieved because the C306 values were not ordered to reflect your required grouping of Model A values separate from Model B values.



# THE LINK COMMAND

The LINK command establishes or cancels a link, which is a symbolic relationship between the source and target subschema records belonging to different databases or to the same database. Once you establish a link, any retrieval into the source subschema record automatically generates a retrieval into the target subschema record. Linking produces a cascade of retrievals. The LINK command has the following features:

- You can establish or cancel a link.
- You can establish up to ten links at any time.
- Your program logic controls initialization, continuation, and termination of a cascade.
- You can specify inclusion lists for the retrievals caused by LINK.
- You can specify any Locate File/stack pair.
- You can establish the link based on more than one condition.

In a LINK command, you specify a link verb, a Locate File or a stack number, a source subschema record, a target subschema record, and a via-clause with one or more conditions. SYSTEM 2000 software stores this information internally. A link is canceled when a source subschema record is retrieved either because another LINK command was issued or because a retrieval command was issued, such as GET, GET1, GETA, GETD, or GETP. From then until the link is canceled, SYSTEM 2000 software performs the following operations, in this order:

- the equivalent of a LOCATE of the target subschema record in the Locate File specified in the LINK command, using a where-clause constructed from the via-clause specified in the LINK command.
- the equivalent of a GET *target-SSR* FIRST, using the stack associated with the Locate File.

The details of LINK processing are given in **LINK Processing** on page 8-166. See **LOCATE** on page 8-81 and **GET** on page 8-100 for details about the LOCATE and GET commands, respectively.

## Format

```
link-verb [[(m)] [source-control] source-SSR TO [/HOLD]
     [target-control] target-SSR [inclusion-list] via-clause]
```

*link-verb* is one of LINK0, LINK1, LINK2, . . . , LINK9.

*m* is the number of the Locate File to be used in the implied LOCATE and number of the stack to be used in the implied GET. This number is either a constant or a variable whose value is equal to or greater than zero and is equal to or less than the number of extra Locate Files requested in the START S2K command. The default is 0. For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*source-control* is either a + symbol or a - symbol, indicating whether a LINK can only initialize or only continue a cascaded retrieval. If you do not specify either symbol, both initialization and continuation are allowed.

*source-SSR* is the name of a subschema record.

*target-control* is either a + symbol or a - symbol, indicating whether a cascade must continue beyond or terminate with the target subschema record. If you do not specify either symbol, continuation is allowed if the return code is zero.

*target-SSR* is the name of a subschema record different from the *source-SSR*.

*inclusion-list*

```
|item-list

|USING |name-table
       |word-table
```

If you do not specify an inclusion list, all items in the subschema record are included. See **Inclusion List** on page 8-121 for details about *inclusion-list*.

*via-clause* **VIA** *association-list*

*association-list* *variable/item* **[**, *association-list***]**

*variable* is a 01-level variable in a COBOL PLEX program. The variable can belong to a subschema record.

*item* is the alias of a database schema item declared in a subschema record. This item must belong to the same database and the same family as the schema record corresponding to the target subschema record.

## Examples of Syntax

```
LINK4.                                    (cancels the link LINK4)
LINK0 SCH1 TO SSR2 VIA V1/E1.
LINK1(1) SCH3 TO SSR4 VIA V2/E2, V3/E3, V4/E4.
LINK2(2) + SCH2 TO - SSR5 VIA V5/E5.
```

## Considerations

1. Check the S2KLKH table in S2KDUM for return codes after a retrieval that produces a cascade (a series of retrievals that occur due to links). See **LINK Return Codes and S2KLKH Table** on page 8-169.

2. A LINK command cancels any previously established link for the same link verb. Therefore, a maximum of ten LINK commands can be active concurrently, one for each link verb.

3. The target subschema record cannot be the same as the source subschema record.

4. The via-clause cannot have more than 26 variable/item pairs.

5. You can open and close databases without affecting links. Links established before you close a database are still in effect when you reopen the database.

6. If you specify the HOLD option in a LINK command, SYSTEM 2000 software executes the equivalent of a LOCATE command followed by a GET/HOLD FIRST command for the specified subschema record. This implies holding all records involved in a cascade if the MULTIPLE HOLDS option is enabled.

7. If different LINK commands specify the same stack, the retrieval due to one LINK is overwritten by the retrieval due to another. Therefore, specify stack numbers carefully when using LINK commands.

8. Following a cascade, your program can issue GET commands to retrieve the located data records using the correct Locate File subscript. That is, the Locate Files produced by a cascade are available after the cascade is completed. They can be used for other retrievals and can be ordered in a different way with an ORDER BY command.

9. If the MULTIPLE HOLDS option is enabled, your program can request any number of local holds on different records simultaneously. The MULTIPLE HOLDS option is disabled by default. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for details about setting the MULTIPLE HOLDS option. Local holds remain in effect until one of the following events occurs in the PLEX program:

   - a global hold is attempted for any database
   - a FRAME command is issued for any database
   - a synchpoint occurs
   - the program terminates

- a DROP HOLD command occurs
- the QUEUE command is issued.

  Once any of these occurs, all of the program's local holds on the database(s) are dropped. Also, an update command no longer causes a local hold to be dropped. See **HOLD Option** on page 80 for a complete discussion of local holds. See also **HOLD Option** on page 8-131 and **DROP HOLD** on page 8-144.

10. If a subroutine that is to be deleted contains any links, they should be canceled before the subroutine is deleted.

# LINK PROCESSING

---

*link-verb* **[[(***m***)]** **[***source-control***]** *source-SSR* **TO [/HOLD]**
**[***target-control***]** **target-SSR [***inclusion-list***]** *via-clause***]**

---

There are ten link verbs: LINK0, LINK1, . . . , LINK9. A link verb is *active* when it is declared in a LINK command with a source subschema record, a target subschema record, and other information. A LINK declaration links the source subschema record with the target subschema record. With each active link verb, SYSTEM 2000 software keeps all the information given in the LINK command, such as the source and target subschema record names, the source and target control information, the via-clause, the inclusion list, the stack number, and so on. When any of the commands GET, GET1, GETA, GETD, or GETP retrieves a source subschema record specified for a link verb, the software uses the information associated with the link verb to initialize a retrieval cascade. The software continues a retrieval cascade if the target subschema record of the link verb appears as a source subschema record for another link verb, using the information associated with the second link verb.

If your program specifies a specific link verb more than once, the software honors only the most recent specification. A LINK command with only a link verb cancels the link associated with that verb, that is, deactivates the link verb.

As a simple example of a link cascade, suppose the following links are active:

```
LINK1 (1) A TO B VIA. . .
LINK2 (2) B TO C VIA. . .
LINK3 (3) A TO D VIA. . .
      :
      :
GET1  A  FIRST.
```

Assuming all the retrievals are successful, data are retrieved into subschema record A by the original GET1, into subschema record B by LINK1, into subschema record C by LINK2, and into subschema record D by LINK3. At the end of the retrieval cascade, the Locate Files 1, 2, and 3 contain data record addresses determined by the corresponding target subschema records and via-clauses. Stacks 1, 2, and 3 contain the addresses of the data records retrieved into subschema records B, C, and D respectively, whereas stack 0 contains the address of the data record retrieved into subschema record A.

Details about the Locate File (such as the number of data records, the last data record pointer, and so on) are stored in the COMMBLOCK. This information changes as the Locate File or stack changes with the cascaded retrievals. Therefore, at the end of a cascaded retrieval, the COMMBLOCK contains the information about the Locate File referenced last in the cascade. But the information about the other Locate Files is saved by SYSTEM 2000 software and is restored in the COMMBLOCK when a reference to that Locate File/stack number is made in another command. See **COMMBLOCK** on page 7 for details about the Locate File and COMMBLOCK.

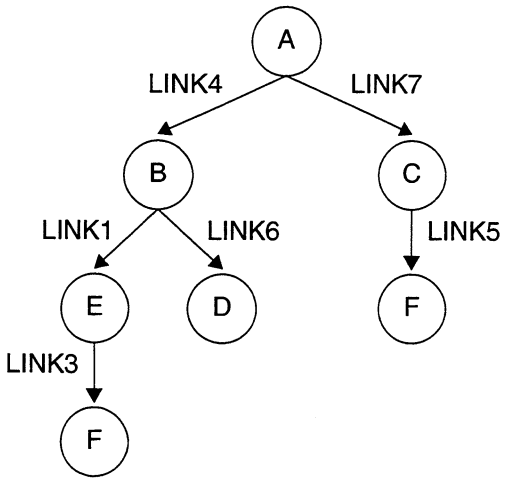
## Order of Retrievals in a Cascade

LINK commands are stored in an array in ascending order by link verb. For each retrieval in a cascade, the link processor searches this array from the lowest to the highest link verb for an appropriate source subschema record.

Suppose the following links are active:

```
LINK1 B TO E
LINK3 E TO F
LINK4 A TO B
LINK5 C TO F
LINK6 B TO D
LINK7 A TO C
```

Then assume that the program issues a GET1 A FIRST command. Starting from A, the retrieval cascade can be represented as a tree as shown below. Note that for the same source, a link with a lower link verb appears on the left side of a link with a higher link verb. The cascade is processed in top-to-bottom, left-to-right order.

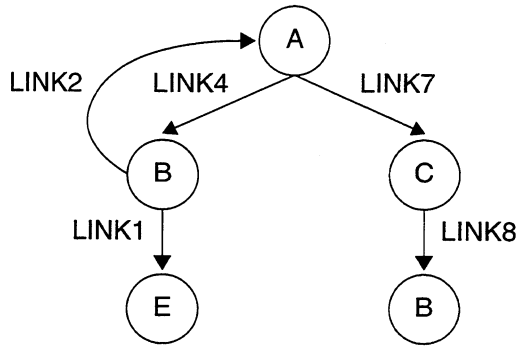



**processing order**

LINK4
LINK1
LINK3
LINK6
LINK7
LINK5

To avoid possible infinite loops, SYSTEM 2000 software does not permit a LINK command to be processed when the following situations exist:

- The source subschema record of a LINK command is the same as its target.
- In the same path of a cascade, a source in a higher part is the same as a target in a lower part, as shown below.



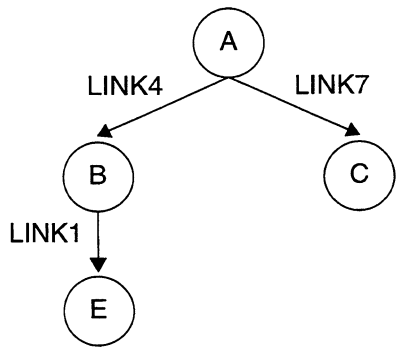

In particular, as soon as a record is retrieved for a target SSR, any other active LINK command with that SSR as a target is not executed during the cascade. Also, if the SSR of the retrieval command initializing the cascade appears as a target in an active LINK command, that LINK command is not executed during the cascade.

Active LINK commands not executed during one cascade remain active until they are explicitly canceled. That is, they can be executed in a subsequent cascade, depending on how that cascade is initialized.

Suppose the following links are active:

```
LINK1 B  TO  E
LINK2 B  TO  A
LINK4 A  TO  B
LINK7 A  TO  C
LINK8 C  TO  B
```

Then, for example, the command GET1 A FIRST would produce the following cascade:




Note that although LINK2 and LINK8 are in the list of active LINK commands, they were not executed during this cascade because of the restrictions previously stated.

- A record for SSR A was already retrieved by the command initializing the cascade. Therefore, LINK2 was not executed because it specifies SSR A as its target.

- A record for SSR B was already retrieved in LINK4. Therefore, LINK8 was not executed because it specifies SSR B as its target.

## LINK Return Codes and S2KLKH Table

A LINK command always gives return code 00. When the LINK command is given, the information in the command is only stored by SYSTEM 2000 software. Therefore, an error return code cannot result. However, errors can occur during the actual retrieval process. The status of each retrieval (link) in a cascade is indicated by a return code for each link verb in the table called S2KLKH in the S2KDUM control block. S2KLKH has ten words. The first word stores the result of LINK0, the second word stores that of LINK1, and so on. A value of -1 in a field of this table indicates that the corresponding link verb did not participate in the cascade. The return code of the original retrieval is left undisturbed in the second field in the COMMBLOCK.

## Via-clause for LINK

---

*link-verb* **[[(***m***)]** **[***source-control***]** *source-SSR* **TO [/HOLD]**
**[***target-control***]** *target-SSR* **[***inclusion-list***]** *via-clause***]**

---

*via-clause* **VIA** *association-list*

*association-list* *variable/item* **[,** *association-list***]**

*variable* is a variable in the PLEX program. The variable can belong to a subschema record.

*item* is the alias of a database schema item specified in a subschema record. This item must belong to the same database and to the same family as the schema record corresponding to the target subschema record.

As mentioned before, the successful retrieval of data into a subschema record, whose name appears as the source subschema record of a LINK command, automatically performs a LOCATE and a GET FIRST of the target subschema record of the LINK command. The where-clause for this implied LOCATE is constructed from the via-clause as follows:

Each *variable/item* pair in the via-clause forms a condition of the form *item* EQ *variable*. All of these conditions are combined by AND operators. A sample via-clause is shown below.

```
VIA   V1/C1, V2/C2, V3/C3
```

implies

```
WHERE C1 EQ V1 AND C2 EQ V2 AND C3 EQ V3
```

Prior to the retrieval, the variables in the via-clause should have appropriate data descriptions for the intended values in the where-clause comparison. See **Item Types and**

**COBOL Data Descriptions of Items in Where-clauses** on page 4-7 for information about comparisons of items and variables in where-clause processing. See Chapter 7, "Where-Clauses" for other details about where-clause processing.

## Source and Target Control

```
link-verb [[(m)] [source-control] source-SSR TO [/HOLD]
   [target-control] target-SSR [inclusion-list] via-clause]
```

*source-control* is either a + sign or a - sign.

*target-control* is either a + sign or a - sign.

If the *source-control* for an SSR is +, the LINK can only initialize a cascaded retrieval, but it cannot continue one. That is, the retrieval corresponding to the LINK would be allowed only when its source subschema record appears as the subschema record name in a successfully executed GET, GET1, GETA, GETD, or GETP command.

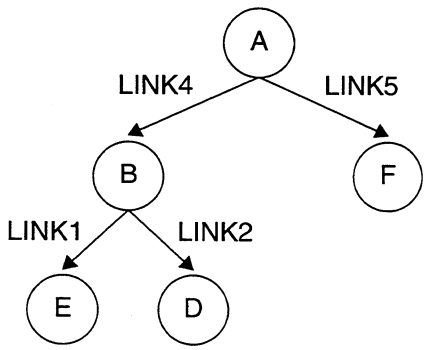
If the *source-control* is -, the LINK can continue a cascaded retrieval, but it cannot initialize one. This means that the retrieval corresponding to the LINK would be allowed only when its source subschema record was retrieved as the target subschema record of some other LINK command.

If you do not specify a *source-control*, you can use the LINK either to initialize or to continue a cascade.

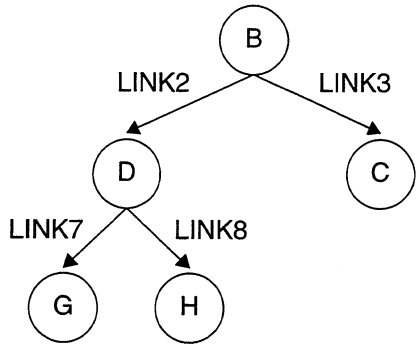
For example, suppose you specify the following LINK commands:

```
LINK1 -B TO E
LINK3 +B TO C
LINK2  B TO E
LINK4 +A TO B
LINK5  A TO F
LINK6 -A TO E
LINK7 -D TO G
LINK8 -D TO H
```

The command GET1 A FIRST produces the following cascade:



GET1 B FIRST results in the cascade shown below.




In the above examples, assume that return code 00 was issued for all the retrievals.

The target control for an SSR determines whether the cascade can continue beyond the target of the LINK command, or whether it should terminate after the retrieval of the target subschema record.

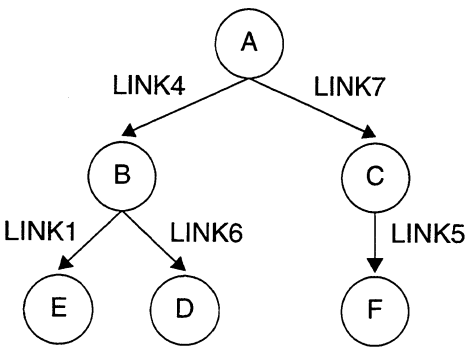
A plus sign indicates that the cascade must continue beyond the target subschema record (regardless of the return code) if such links are active. A minus sign indicates that the branch of the cascade due to the LINK command concerned must terminate at the target subschema record (regardless of the return code).

If you do not specify the target control, the return code determines the continuation of a cascade. Return code 04 terminates a branch.

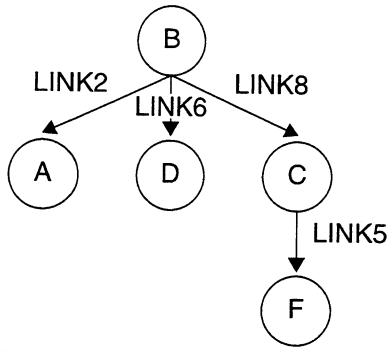
Suppose the following LINK commands are active:

```
LINK1 -B TO -E
LINK2  B TO  A
LINK3  E TO  F
LINK4 +A TO  B
LINK5 -C TO +F
LINK6  B TO  D
LINK7 +A TO +C
LINK8 +B TO  C
```

The command GET1 A FIRST produces the cascade given below (assuming return code 00 for all the retrievals).



The command GET1 B FIRST produces the following cascade (assuming return code 00 for the retrievals):




## Inclusion List for LINK

This discussion pertains to COBOL and FORTRAN PLEX. For PL/I or Assembler PLEX information, see the appropriate appendix.

```
link-verb [[(m)] [source-control] source-SSR TO [/HOLD]
  [target-control] target-SSR [inclusion-list] via-clause]
```

```
inclusion-list    |item-list

                  |USING |name-table
                        |item-list
```

See **Inclusion List** on page 8-121 for details.

The successful retrieval of data into a subschema record, whose name appears as the source subschema record of a LINK command, automatically performs a LOCATE and a GET FIRST of the target subschema record of the LINK. If you specify an inclusion list in the LINK, the software uses it in the implied GET FIRST retrieval. The inclusion list has the same syntax and semantics here as in any retrieval command. If you do not specify an inclusion list, all items in the subschema record participate in the retrieval.

## HOLD Option for LINK

```
link-verb [[(m)] [source-control] source-SSR TO [/HOLD]
  [target-control] target-SSR [inclusion-list] via-clause]
```

You can specify the HOLD option for the target subschema record. In immediate mode, HOLD signifies that the program requests a local hold on the data record retrieved as a result of the link. Also, all records involved in a cascade are held until a synchpoint occurs. In queue mode, on the other hand, the HOLD option indicates a request for a global hold on the database of the target SSR. See **Considerations** on page 8-137 for information about obtaining local holds.

Chapter 9

# Execution Modes

*IMMEDIATE MODE 9-1*

*QUEUE AND TERMINATE 9-2*
*Format 9-2*
*Example of Usage 9-3*
*Considerations 9-3*

*LOAD AND TERMINATE 9-5*
*Format 9-5*
*Considerations 9-5*

*CANCEL QUEUE 9-7*
*Format 9-7*
*Example of Usage 9-7*

**Execution Modes** on page 61 introduced the three execution modes: immediate, queue, and load. This chapter covers primarily the commands for entering and leaving queue mode and load mode. The first section covers aspects of immediate mode. Subsequent sections concentrate on the QUEUE command and how other commands are processed in queue mode. **LOAD and TERMINATE** on page 9-5 treats the same considerations for the LOAD command and load mode. **CANCEL QUEUE** on page 9-7 discusses the CANCEL QUEUE command, which exits either queue mode or load mode.

## IMMEDIATE MODE

A program is in immediate mode until a QUEUE or a LOAD command is given. Upon termination of queue mode or load mode, the program re-enters immediate mode. In contrast to updates in queue or load mode, updates in immediate mode are executed immediately. Each command, whether for update or retrieval, is executed before the next one. The execution of each update (excluding MOVE TREE) includes changing the Hierarchical Table, the Data Table, and the Index to reflect the updated values or structure. A MOVE TREE operation changes only the pointers in the Hierarchical Table. The processing of INSERT commands is delayed only when a series of INSERT commands is given for data records that are all in one tree. In this situation, the processing is delayed so that the entire tree is inserted all at once. Immediate mode is especially useful for occasional, random updates.

Immediate mode makes use of any reusable space when insertions are made. *Reusable space* was used once but was then released (through a REMOVE TREE operation). The use of reusable space is described in item 9 of QUEUE and TERMINATE **Considerations** on page 9-3.

The CANCEL QUEUE command ends a QUEUE/TERMINATE or LOAD/TERMINATE session, and returns the program to immediate mode. If the CANCEL QUEUE command is given in immediate mode, it is ignored.

# QUEUE AND TERMINATE

The QUEUE command changes the mode of execution from immediate to queue. The TERMINATE command initializes the execution of update commands in queue mode, then changes the mode of execution to immediate when execution is complete.

## Format

```
QUEUE
   .
   .
   .
   PLEX commands
        and
   COBOL statements
   .
   .
   .
TERMINATE
```

In queue mode, retrieval commands are processed immediately, but update commands are not. Rather, they are saved in a temporary file to be executed after TERMINATE is issued. At that point, SYSTEM 2000 software makes the updates in a single pass through the Hierarchical Table, the Data Table, and the Index, if necessary. Since update commands are processed all at one time, using queue mode saves processing time.

The update commands that can be issued between a QUEUE command and a TERMINATE command are REMOVE TREE, INSERT, MODIFY, and REMOVE. They must all be issued for the same database.

As SYSTEM 2000 software encounters these update commands, it does not execute them but groups them instead. The REMOVE TREE commands form one group, the INSERT commands form another, and the MODIFY and REMOVE commands form a third group. When the software reaches the TERMINATE command, it executes all the update commands in a batch, first the REMOVE TREE commands, then the INSERT commands, then the MODIFY and REMOVE commands. (Within each group, the software processes commands in the order they were given.) Since the REMOVE TREE commands are processed first, all INSERT, MODIFY, and REMOVE commands based on a removed node are discarded.

When making insertions, queue mode uses reusable space if any is available. Reusable refers to space that was once in use but has now been released. (The other type of free space is available space, which has not been used before.) The software's controls on reusable space are discussed under **Considerations** on page 9-3.

TERMINATE increments the database cycle number by one if any update in the batch changes the database. TERMINATE clears all stacks for the database being updated. Locate Files remain unaffected. TERMINATE also writes updated database pages in main memory to disk (as the CLEAR command does). Pages updated before a QUEUE command

is issued are automatically cleared when the first update command after a QUEUE command is recognized. SYSTEM 2000 software realizes at that point which database will be updated in the QUEUE/TERMINATE session.

## Example of Usage

Suppose the following commands are encountered in the stated order:

```
QUEUE
GET
MODIFY
GETA
REMOVE
GETD
MODIFY
GET1
INSERT
REMOVE TREE
GETP
INSERT
REMOVE TREE
TERMINATE
```

The order of execution of the retrieval and update commands is as follows:

```
GET      \
GETA       \
GETD        |  (processed immediately)
GET1       /
GETP     /
REMOVE TREE\
REMOVE TREE  \
INSERT         \
INSERT          |  (processed at TERMINATE)
MODIFY         /
REMOVE       /
MODIFY     /
```

You must understand the change in order of the update commands before using queue mode. See **INSERT in Queue Mode** on page 10-52 for more examples.

## Considerations

1. Check the return code after the QUEUE command is given. Since QUEUE does not refer to a particular database, its return code is not placed in a COMMBLOCK. Instead, the return code is placed in S2KRTC, a variable in the S2KDUM control block.

2. The QUEUE command should be given only in immediate mode. If it is given during a QUEUE/TERMINATE session or during a LOAD/TERMINATE session, results are unpredictable. If a TERMINATE command was not preceded by a QUEUE or LOAD command, the TERMINATE command is ignored.

3. All update commands between QUEUE and TERMINATE are restricted to one database. However, different databases can be accessed for retrieval. An update command referencing a second database results in return code 28 in the COMMBLOCK of the second database (not the COMMBLOCK of the first database).

4. If a database to be updated is closed by a CLOSE command before TERMINATE, all the updates batched between QUEUE and CLOSE are lost, the stacks and Locate Files for the database are cleared, and the execution mode is changed to immediate mode. The CLOSE command produces return code 27 even if the database is closed successfully.

5. A CANCEL QUEUE command, if issued after QUEUE and before TERMINATE, cancels the batched updates and changes the execution mode back to immediate mode.

6. If the Update Log is active, the QUEUE command does not suspend it. Therefore, if the database becomes damaged and the Update Log remains active, recovery is possible.

7. The QUEUE command ends all local holds for the program.

8. In a Multi-User environment, a database can become damaged by another user before the TERMINATE command is executed. In this situation, the TERMINATE command results in return code 16, and the damage flag for the database is set to 1, signifying damage. The pending queue mode updates are canceled, and the program returns to immediate mode. The database must be restored to an undamaged state before it can be updated again.

9. The space in the Hierarchical and Data Tables that is freed by a REMOVE TREE command is not immediately reusable if more than one user has the database open. Another PLEX program can have a stack or Locate File containing the address of the removed record. It would not be appropriate to insert new, unrelated data into the record and give it a different structural position.

   If no other user has the database open, space from removed records will be reused. The presence of this reused address in a Locate File or stack could produce the following situation. With a REMOVE TREE command, the program using that database causes the deletion of a data record whose address is in a Locate File. While the address for the deleted data record is still in the Locate File, an INSERT command given by the program could cause this address to be reused for a newly inserted record. The presence of this reused address in the Locate File could cause problems for GET and ORDER BY commands using that Locate File. To avoid these problems, the program should coordinate retrievals and deletions of records whose addresses are in Locate Files.

10. Each combination of a QUEUE command and an associated TERMINATE command establishes a user synchpoint for the issuing user if the Rollback Log is enabled. User synchpoints are used by the Coordinated Recovery feature and cause any updates performed since the issuing user's last user synchpoint (logical unit of work) to be permanently recorded (committed) in the affected databases. For details see the ENABLE/DISABLE ROLLBACK, COMMIT, ROLLBACK, and RESET ROLLBACK commands. See also the Coordinated Recovery discussion in *SYSTEM 2000 CONTROL Language, Version 12, First Edition.*

11. TERMINATE clears the stacks but not the Locate Files for all subscripts associated with the QUEUE/TERMINATE database.

12. If a MOVE TREE command is issued between QUEUE and TERMINATE commands, return code 28 results.

# LOAD AND TERMINATE

The LOAD command changes the mode of execution from immediate to load. The TERMINATE command completes the load mode insertions and changes the execution mode back to immediate. Load mode is the most efficient way to populate a database.

## Format

```
LOAD database-identifier
  :
     PLEX INSERT commands only
          and
     COBOL statements
  :
TERMINATE
```

*database-identifier* is the name of a database or name of the COMMBLOCK.

In load mode, only INSERT commands can be issued for the specified database. Insertions are made immediately in the Hierarchical Table and the Data Table of the database, but insertions are made in the Index (for key items) only when TERMINATE is executed. Load mode insertions use available space, that is, space that was allocated for the Hierarchical Table and Data Table but not previously used.

TERMINATE increments the database cycle number by one if any updates are successful. It also clears all stacks for the database being updated. Both LOAD and TERMINATE write updated database pages to disk (as the CLEAR command does).

A valid record is immediately placed in the first sequentially available entry in the location space of the Data Table. However, these entries are not logically in the database until the LOAD/TERMINATE session completes.

For examples of the use of LOAD and TERMINATE, see **INSERT in Load Mode** on page 10-66 and Chapter 21, "Populating a Database."

## Considerations

1. Check the return code after the LOAD command is given. The command's return code is placed in the second COMMBLOCK field for the database.

2. If you do not acquire exclusive use of a database before giving a LOAD command for it, the command is ignored, and return code 91 results.

3. You can give the LOAD command only in immediate mode. The software ignores it if given from any other mode. Also, the software ignores the TERMINATE command without a prior LOAD (or QUEUE) command.

4. If an INSERT command between LOAD and TERMINATE is issued for a database other than the one specified in the LOAD command, the insertion is ignored, and return code 53 is placed in the COMMBLOCK of the database referenced by the INSERT command.

5. If the database specified in LOAD is closed by a CLOSE command before TERMINATE, the incomplete INSERT updates are removed from the database tables, and the execution mode is changed to immediate mode. In addition, the database is closed. The CLOSE command in this situation produces return code 27 even if the database is closed successfully.

6. If the LOAD command specifies no database name or an invalid database name, the PLEX processor gives an error message for the command.

7. The CANCEL QUEUE command, if executed in load mode, removes incomplete updates and restores the immediate mode of execution. (See **CANCEL QUEUE** on page 9-7 for more information about CANCEL QUEUE.)

8. Any command other than INSERT, CANCEL QUEUE, or CLOSE between LOAD and TERMINATE is ignored with return code 55. After such a command is given, every subsequent command except TERMINATE is ignored, including the INSERT command.

9. The first valid insertion cancels the contents of every stack and Locate File for every database, except the stack (or Locate File) last referenced and the Locate File (or stack) associated with it. This stack must contain the correct data record address for the insertions when the LOAD command is given.

10. If a load job ends abnormally before TERMINATE, SYSTEM 2000 software marks the database. When the next user opens the database, an automatic recovery takes place and the incomplete updates are removed from the Hierarchical Table and the Data Table.

11. Each LOAD and each associated TERMINATE command establishes a user synchpoint for the issuing user if the Rollback Log is enabled. In addition, at the end of TERMINATE processing, the Rollback Log is reset. User synchpoints are used by the Coordinated Recovery feature and cause any updates performed since the issuing user's last synchpoint (logical unit of work) to be permanently recorded (committed) in the affected databases. For details, see the ENABLE/DISABLE ROLLBACK, COMMIT, ROLLBACK, and RESET ROLLBACK commands. See also Coordinated Recovery discussion in *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

# CANCEL QUEUE

The CANCEL QUEUE command cancels the effect of a QUEUE or LOAD command. It provides a way to go back to the state prior to QUEUE or LOAD if an error is detected in the updates to be made in the database.

## Format

```
CANCEL QUEUE
```

If the CANCEL QUEUE command is executed after QUEUE and before TERMINATE, it cancels the updates batched since the QUEUE command was issued. The execution mode is changed to immediate.

In load mode, the command removes the incomplete updates from the Hierarchical Table and Data Table and restores the immediate mode of execution. In immediate mode, CANCEL QUEUE is ignored.

All stacks and Locate Files remain unaltered by CANCEL QUEUE.

Queue or load mode updates are not recorded on the Update Log, and their before-images are not recorded on the Rollback Log until TERMINATE is issued. Therefore, if Coordinated Recovery is in effect (that is, the Rollback Log is enabled), a CANCEL QUEUE issued in queue or load mode does not cause recovery to occur. For CANCEL QUEUE, the scratch file that contains the batched updates for QUEUE is evicted, or the updates from the Hierarchical and Data Tables for LOAD are removed.

## Example of Usage

```
      .
      .
      .
   LOAD PERSONNEL.                    (begins LOAD/TERMINATE session to populate
                                      the database called PERSONNEL)
      .
      .
      .
   (Read input record)
   IF error in input data
   GO TO CANCEL-LOAD.
      .
      .
      .
TERMINATE.
      .
      .
      .
CANCEL-LOAD.

CANCEL QUEUE.                         (Control reaches here only if there is
                                      an error in input data.)

GO TO END-OF-PROGRAM.
      .
      .
      .
```

Chapter 10

# Updates


*INTRODUCTION 10-2*

*REMOVE 10-2*
- *Format 10-3*
- *Examples of Syntax 10-3*
- *Example of Usage 10-4*
- *Considerations 10-5*

*MODIFY 10-11*
- *Format 10-12*
- *Examples of Syntax 10-12*
- *Example of Usage 10-13*
- *Considerations 10-14*

*REMOVE TREE 10-21*
- *Format 10-21*
- *Examples of Syntax 10-22*
- *Example of Usage 10-22*
- *Considerations 10-23*

*INSERT: GENERAL INFORMATION FOR ALL MODES 10-28*
- *Format 10-28*
- *Examples of Syntax 10-29*
- *Considerations 10-30*

*INSERT IN IMMEDIATE MODE 10-31*
- *Processing an Insert Sequence 10-32*
- *Ending an Insert Sequence 10-32*
- *BEFORE and AFTER 10-33*
- *Example of Usage 10-34*
- *Considerations 10-35*
- *Examples of Insert Sequences 10-37*

*INSERT IN QUEUE MODE 10-52*
- *Example of Usage 10-54*
- *Considerations 10-55*

*INSERT IN LOAD MODE 10-66*
- *Example of Usage 10-67*
- *Considerations 10-68*

*MOVE TREE 10-75*
- *Format 10-77*
- *Considerations 10-77*
- *Examples 10-78*

*CLEAR COMMANDS: MOVING UPDATED PAGES TO DISK 10-81*
*Format 10-83*
*Example of Usage 10-84*
*Considerations 10-85*

# INTRODUCTION

This chapter describes the five update commands: REMOVE, MODIFY, REMOVE TREE, INSERT, and MOVE TREE. A general understanding of the retrieval commands and a familiarity with the syntax and usage of the retrieval commands is a prerequisite for understanding the update process. Several sections of this chapter are devoted to the INSERT command, which requires an understanding of the three execution modes in PLEX: immediate mode, queue mode, and load mode (see Chapter 9, "Execution Modes").

# REMOVE

The REMOVE command deletes one or more values from a data record associated with the specified subschema record.

- You can remove values from a data record at any level of the database.
- You can specify any one of 16 stacks.
- You can select values for specific (rather than all) items in a subschema record for removal, with an inclusion list.

After a retrieval command establishes a stack entry for the data record to be updated, a REMOVE command can be given. This command specifies a stack and a subschema record associated with the data record.

When SYSTEM 2000 software executes a simple REMOVE command (one without an inclusion list), all values in the data record that correspond to subschema items become null. When a REMOVE command contains an inclusion list, only the values corresponding to the indicated subschema items become null.

A REMOVE command does not cause a change in the data structure, and storage space is reserved in the Data Table for values that are null. Even if all the items in a data record become null, the data record retains its place in the logical data structure.

For the REMOVE command, SYSTEM 2000 software takes care of the display representation for nulls, appropriate for particular data types.

## Format

```
REMOVE [(stack-number)] SSR inclusion-list
```

*stack-number* is the number of the stack to be used in the update: a constant or a variable whose value is greater than or equal to zero and equal to or less than 15. The default stack number is zero. For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*SSR* is the name of the subschema record for the record to be updated.

*inclusion-list*

```
|item-list
|USING |word-table
       |name-table
```

*item-list* *item-name* [, *item-list*]

*item-name* is the alias of an item in the specified subschema record.

*name-table* is the name of a table of component names or C-numbers of items to be included in the update operation. For COBOL, the entry with the OCCURS clause must specify PICTURE X(32).

*word-table* is the name of a table of one-word indicators (1s and 0s), one indicator for each item in the specified subschema record. An item is included in the update operation if its indicator in the table is one. Conversely, an item is excluded from the operation if its indicator is zero. In each indicator, only the low-order bit is checked to see if it is zero or one. For COBOL, the entry with the OCCURS clause must specify COMP usage.

short form of the command and its equivalent

```
REMOVE SSR.   (short form using the default)
REMOVE (0) SSR.
```

## Examples of Syntax

```
REMOVE SALARY.                    (short form, which implies stack 0);
                                  indicates that items specified by the SALARY
                                  subschema record are to become null in the
                                  data record addressed by the stack)

REMOVE (8) POSITION-REC.          (specifies stack 8 and the POSITION-REC
                                  subschema record)

REMOVE SKILL PROFICIENCY.         (specifies that only the value for
                                  the subschema item PROFICIENCY is to
                                  become null)
```

```
REMOVE (5) ENTRY-REC USING          (specifies the name table called
     ENTRY-NAME-TABLE.               ENTRY-NAME-TABLE as the inclusion list
                                     indicating the subschema items whose
                                     corresponding values are to become null)
```

## Example of Usage

Possible sequence of commands, including REMOVE.

```
WORKING-STORAGE SECTION.
      .
      .
COMMBLOCK OF EMPLOYEE.
01  EMPLOYEE.
      .
      .
   03  RETURN-KODE                  (field 2)
      .
      .
   03  NUMBER-OF-ADDRESSES.         (field 6)
      .
      .
SSR POSITION-REC OF EMPLOYEE.       (SSR corresponds to the POSITION
                                     WITHIN COMPANY schema record)
      .
      .
SSR ENTRY-REC OF EMPLOYEE.
01  ENTRY-REC.                     (SSR corresponds to the level 0 SR)
   03  SSN. . .
       EQ SOCIAL SECURITY NUMBER.
   03  CLEARANCE. . .
       EQ SECURITY CLEARANCE.
   03  EMP-STATUS. . .
       EQ EMPLOYEE STATUS.
      .
      .
PROCEDURE DIVISION.
      .
      .
   LOCATE (3) POSITION-REC WHERE . . .(retrieves into Locate File 3 the addresses
                                       of all POSITION-REC data records that meet
                                       the conditions in the where-clause; check
                                       RETURN-KODE and NUMBER-OF-ADDRESSES)

   GET (3) POSITION-REC NEXT.       (retrieves the POSITION-REC data record
                                     whose address comes next in the Locate
                                     File; check RETURN-KODE)

   GETA (3) ENTRY-REC.              (retrieves the ENTRY-REC data record that
                                     is the parent of the POSITION-REC data
                                     record; check RETURN-KODE)

   REMOVE (3) ENTRY-REC CLEARANCE.  (removes only the data value corresponding
                                     to the subschema item CLEARANCE;
                                     check RETURN-KODE)
```

## Considerations

1. Check the return code after the REMOVE command is given. The command's return code is placed in the second COMMBLOCK field for the database.

2. For a discussion of inclusion lists, see **Inclusion List** on page 8-121.

3. If you want to remove and also change values for items in a subschema record, use the MODIFY command. Move the representations for null to the subschema items whose values are to be removed.

4. Although you can include the keyword USING when it refers to an inclusion list, you cannot specify the phrase USING 0 with REMOVE or any other update command. In this situation, the PLEX processor gives an error message for the command.

5. Before the REMOVE command is issued, the specified stack must already contain an entry for the data record to be updated. If there is no entry at the level of the subschema record, the command is ignored, and return code 02 is issued.

6. A program's current password must have U-authority (update) for each item included in the REMOVE operation. If it does not, the command is ignored, and return code 47 results. The eighth field of the COMMBLOCK tells the position in the subschema record of the subschema item corresponding to the item for which the program lacks authority. For example, if the item is the second one declared in the specified subschema record, the eighth COMMBLOCK field has the value of 2.

7. In queue mode, update commands are not executed as they are encountered, but rather are grouped together. All REMOVE TREE commands form one group; all INSERT commands form another group; MODIFY and REMOVE commands form a third group. When a TERMINATE command is encountered, all update commands are executed in a batch -- first the REMOVE TREE commands, then the INSERT commands, then the MODIFY and REMOVE commands. Within each group, commands are executed in the order in which they were given.

8. In queue mode, a REMOVE command must refer to the database named in the first update command of that QUEUE/TERMINATE session. If the REMOVE command refers to a different database, it is ignored, and return code 28 results.

9. In a Multi-User environment, the database can become damaged by another user before a REMOVE command is executed. In this situation, the REMOVE command results in return code 16, and the damage flag for the database is set to 1, signifying damage. The database must be restored to an undamaged state before it can be updated again.

10. The stack entry to be used in a REMOVE command might have been canceled in a REMOVE TREE or MOVE TREE operation by the program or by another user in a Multi-User environment. If this happens, the REMOVE command is ignored, and return code 09 results. Also, the eighth field in the COMMBLOCK contains the database level of the removed stack entry.

11. If the specified stack number is negative or greater than 15, the command is ignored, and return code 40 results.

12. In a Multi-User environment, the setting of the Multi-User execution parameter LHOLD affects the execution of a REMOVE command in immediate mode. If LHOLD is set to YES, a PLEX program must obtain a local hold on a particular data record before giving a REMOVE command for that record in immediate mode. If the program does not first obtain the local hold, the REMOVE command is ignored and return code 111 results. If, on the other hand, LHOLD is set to NO, the program need not obtain a local hold before giving a REMOVE command in immediate mode.

13. A REMOVE command refers only to non-key items in a data record if either 1) the specified subschema record contains only non-key items or 2) an inclusion list specifies only the non-key items in a subschema record. In a Multi-User environment, such a REMOVE command can be executed in immediate mode while retrievals are being processed for the same database and while other users have local holds elsewhere in that database. This is possible in two situations:

    - when the Multi-User execution parameter LHOLD is set to YES and the program has already obtained a local hold for the data record to be updated
    - when LHOLD is set to NO.

    In either situation, retrievals and updates of non-key values should be coordinated, because a PLEX program can give a REMOVE command for a non-key value that another user expects to retrieve or update. A PLEX program can, for example, remove a non-key value from a record whose address is in a Locate File for another program. That other program, moreover, might need to retrieve or update the value that was removed.

    If LHOLD is set to YES and if the program gives an immediate mode REMOVE command applying to key items in a data record for which the program has obtained a local hold, the execution of the command is postponed until both the following conditions are satisfied:

    - a current global hold is ended
    - any retrieval or update operations currently in progress are completed.

    If, however, LHOLD is set to YES and the program does not obtain a local hold for that data record beforehand, the REMOVE command is ignored and return code 111 results.

    See Illus. 8.4 on page 8-139 for the disposition of requests for retrievals, holds, and updates in a Multi-User environment.

The following illustrations show different situations of updating with REMOVE. They are summarized here.

**Example 1:** REMOVE uses an SSR with items for only two of the items in the corresponding data record.

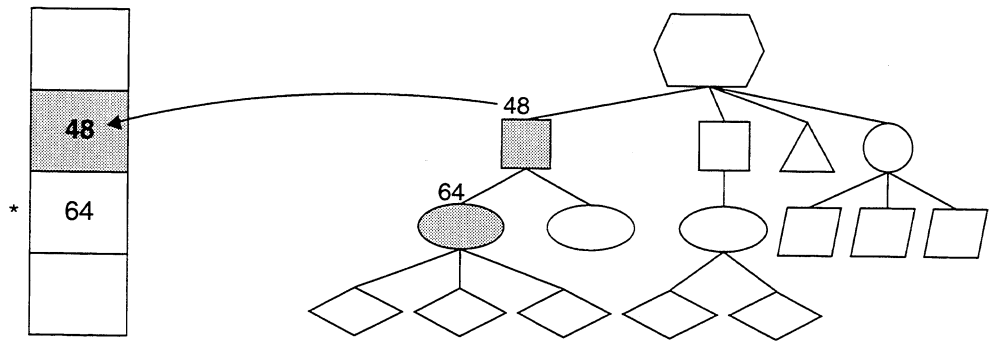
**Example 2:** REMOVE uses an SSR specifying all items in the corresponding data record.

**Example 3:** REMOVE specifies a word table.

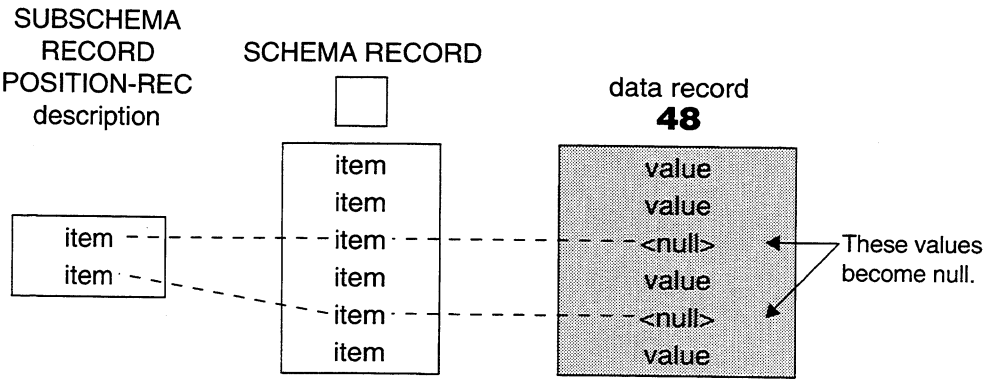
### Example 1

In this example, a REMOVE command specifies a subschema record containing items for only two of the items in the corresponding ☐ data record. Only these two items become null.

GETA POSITION-REC



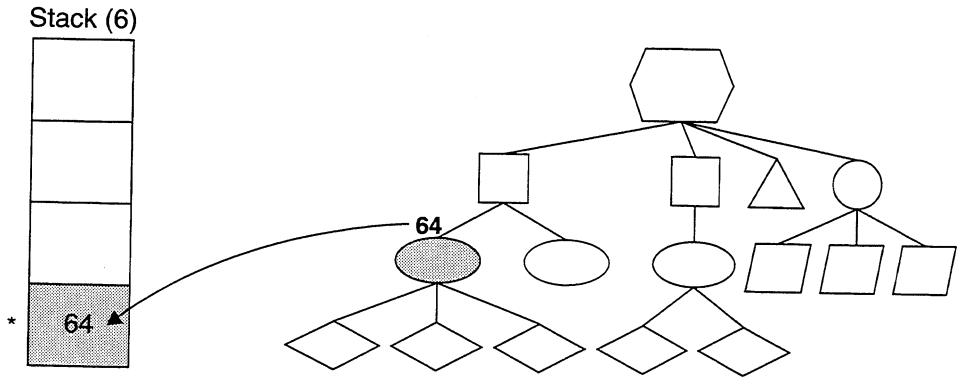

REMOVE POSITION-REC



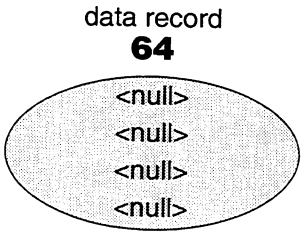
**Example 2**

In this example, the subschema record SALARY contains an item for each of the four items in a ⬭ data record. All four items in the data record become null, but the record remains part of the data structure.

GET1 (6) SALARY NEXT




REMOVE SALARY

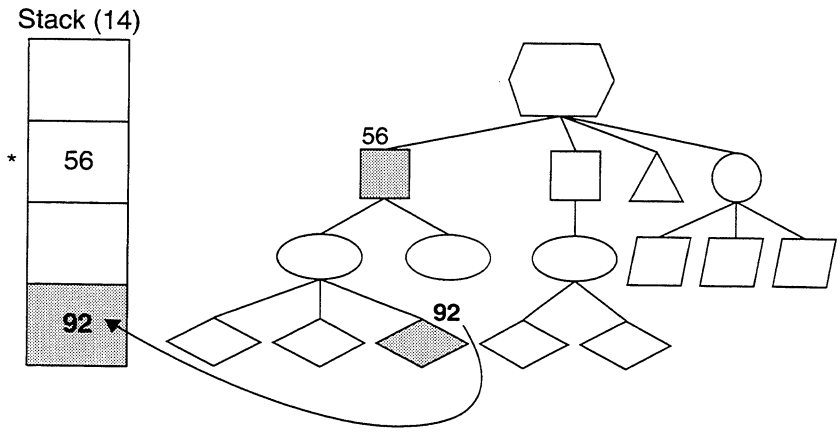



The data record still retains its place in the logical data structure.

### Example 3

Suppose your SSR specified only five items and the corresponding schema record has six. In this example, a word table is defined for the five items, and only three items will become null.

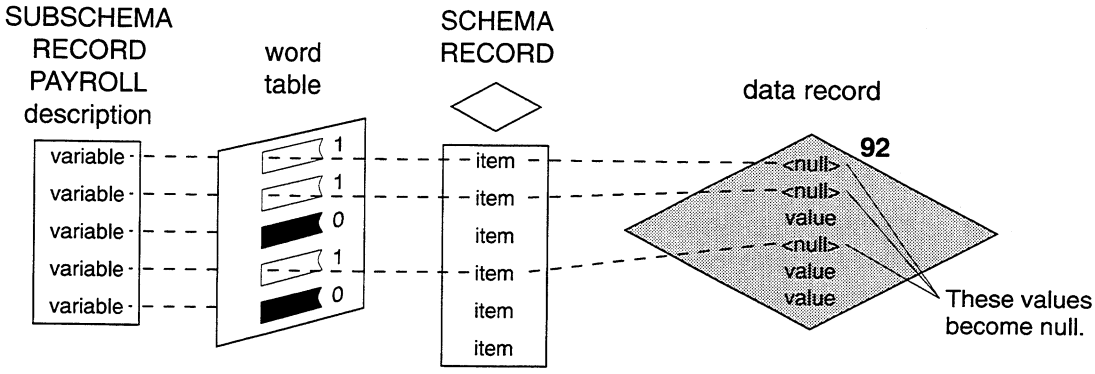
```
WORKING-STORAGE SECTION.
      .
      .
      .
01  PAYROLL-WORD-TABLE.
    05  PAYROLL-WORD-ITEM OCCURS 5 TIMES PIC S9(9) COMP.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
   MOVE 1 TO PAYROLL-WORD-ITEM (1).
   MOVE 1 TO PAYROLL-WORD-ITEM (2).
   MOVE 0 TO PAYROLL-WORD-ITEM (3).
   MOVE 1 TO PAYROLL-WORD-ITEM (4).
   MOVE 0 TO PAYROLL-WORD-ITEM (5).
      .
      .
      .
   GETD (14) PAYROLL NEXT.
```



**Example 3 (continued)**

The following command removes three of the six items in the data record:

```
    :
REMOVE PAYROLL USING PAYROLL-WORD-TABLE.
```



# MODIFY

The MODIFY command changes or deletes existing values in a data record and adds values to a data record.

- You can modify values in a data record at any level of the database.
- You can specify any one of 16 stacks.
- You can modify items in a data record that correspond to specific (rather than all) subschema items by specifying an inclusion list.

Before issuing a MODIFY command, your program should

1. give a retrieval command to establish a stack entry for the data record to be updated and to place the current data values for the record in the subschema record area.

2. in the subschema record area associated with this data record, place the new values or the appropriate representations of null for the data items to be updated. These new data are thus made accessible to SYSTEM 2000 software for the update. The new values in the subschema record area are available to replace corresponding data record values. Also, the representations of null in the subschema record area indicate the corresponding items that will become null.

Then the program can issue a MODIFY command, specifying the stack and the subschema record. When SYSTEM 2000 software executes a simple MODIFY command (one without an inclusion list), all of the subschema items play a role in the update. The values in the subschema record area are transferred to the data record to be updated.

Once the modification is performed, the data record contains new values from the subschema record as well as any other items that might have become null.

Values for which no corresponding subschema items were specified in the SSR description are not changed. But when the MODIFY command has an inclusion list, it can exclude some of the subschema items from the update processing. The inclusion list specifies particular subschema items whose values are to be used in the update; database values are not changed for subschema items that are not specified.

The MODIFY command does not cause a change in the data structure; new values are brought in to replace old ones, and storage space is reserved for values that became null. Even if all of its items become null, a data record retains its place in the logical data structure.

When a program must make a certain value null, it should move to the corresponding subschema item the representation of null that is appropriate for the COBOL data description of that item. Illus. 4.1 on page 4-10 shows the representation of null appropriate for each COBOL data description. For PL/I, FORTRAN, or Assembler representations of null, see the appropriate appendix.

## Format

```
MODIFY [(stack-number)] SSR inclusion-list
```

*stack-number* is the number of the stack to be used in the update: a constant or a variable whose value is greater than zero and equal to or less than 15. The default value of *stack-number* is zero. For COBOL, the variable must be 01-level and can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*SSR* is the name of the subschema record for the record to be updated.

*inclusion-list*

```
|, item-list
|USING |name-table
       |word-table
```

*item-list* *item-name* [, *item list*]

*item-name* is the alias of an item in the specified subschema record.

*name-table* is the name of a table of component names or C-numbers for items to be included in the update operation. For COBOL, an entry with the OCCURS clause must specify PICTURE X(32).

*word-table* is the name of a table of one-word indicators (1s and 0s), one indicator for each item in the specified subschema record. An item is included in the update operation if its indicator in the table is one. Conversely, an item is excluded from the operation if its indicator is zero. In each indicator, only the low-order bit is checked to see whether it is zero or one. For COBOL,the entry with the OCCURS clause must specify COMP usage.

short form of the command and its equivalent

```
MODIFY SSR.  (short form using the default)
MODIFY (0) SSR.
```

## Examples of Syntax

```
MODIFY SKILL.                    (is the short form; implies stack 0;
                                 specifies that the values in subschema record
                                 SKILLS are to be transferred to the data
                                 record addressed by the stack)

MODIFY (11) SALARY.              (specifies stack 11 and the values in the
                                 SALARY subschema record area)

MODIFY EDUCATION SCHOOL.         (specifies that only the value for
                                 the subschema item SCHOOL is to be
                                 transferred to the data record
                                 addressed by the stack)
```

```
MODIFY (3) ENTRY-REC USING          (specifies stack 3 and the name table
   NAME-TABLE.                       called NAME-TABLE as the inclusion list,
                                     indicating the subschema items that
                                     contain values to be transferred to
                                     the data record)
```

## Example of Usage

Possible sequence of commands, including MODIFY.

```
WORKING-STORAGE SECTION.
      :
      :
COMMBLOCK OF EMPLOYEE.
01  EMPLOYEE.
      :
      :
    03 RETURN-KODE.                 (field 2)
      :
      :
    03 NUMBER-OF-ADDRESSES.         (field 6)
      :
      :
SSR POSITION-REC OF EMPLOYEE.
01  POSITION-REC.                  (SSR corresponds to the POSITION
                                    WITHIN COMPANY schema record)
    03  TITLE . . .
        EQ POSITION TITLE.
    03  DEPARTMENT . . .

SSR EMP OF EMPLOYEE.
01  EMP.                           (corresponds to the level 0 schema record)
    03  LASTNAME . . .
        EQ LAST NAME.
    03  SSN . . .
        EQ SOCIAL SECURITY
        NUMBER.
    03  OFFICEPHONE . . .
        EQ OFFICE-EXTENSION.
      :
      :

PROCEDURE DIVISION.
      :
      :
   LOCATE (9) POSITION-REC WHERE . . .(retrieves into Locate File 9 the
                                       addresses of all POSITION-REC
                                       data records that meet the
                                       where-clause conditions; check
                                       RETURN-KODE and NUMBER-OF-ADDRESSES)

   GET (9) POSITION-REC USING 0 NEXT. (establishes the primary stack entry
                                       for the POSITION-REC data record
                                       whose address is next in the
                                       Locate File, but retrieves no
                                       data values)
```

```
(Input new values into SSR POSITION-REC)

MODIFY (9) POSITION-REC.          (replaces old TITLE and DEPARTMENT
                                  values with the new ones now in
                                  SSR POSITION-REC)

GETA (9) EMP.                     (retrieves and establishes a secondary
                                  stack entry for the EMP data record
                                  that is the parent of the POSITION-REC
                                  data record)

(Input new value into OFFICEPHONE
in SSR EMP)

MODIFY (9) EMP OFFICEPHONE.       (replaces only the old OFFICEPHONE
                                  value with the new value in SSR
                                  EMP)
```

## Considerations

1. Check the return code after the MODIFY command is given. The command's return code is placed in the second field of the COMMBLOCK for the database to which the command refers.

2. For a discussion about inclusion lists, see **Inclusion List** on page 8-121.

3. If a value in the subschema record is identical to the corresponding value in the data record to be modified, SYSTEM 2000 software does not write that value again in the data record when the MODIFY command is executed.

4. Although the MODIFY command can include the keyword USING when it refers to an inclusion list, the phrase USING 0 should not be used with MODIFY or any other update command. If you specify USING 0, the PLEX processor gives an error message for the command.

5. Before the MODIFY command is issued, the specified stack must already contain an entry for the data record to be updated. If there is no entry at the level of the subschema record, the command is ignored, and return code 02 is issued.

6. A program should ensure that each subschema item referred to in a MODIFY command has a value conforming to the item type of the corresponding schema item. If it does not, the command is ignored, and return code 10 results. (However, FOR COBOL, if a value in a COMP-3 subschema item field is not in packed-decimal format, a MODIFY command referring to that subschema item results in completion code OC7.)

   For COBOL, MODIFY processing of a numeric value is most efficient when it appears in a subschema item with usage COMP-3 (rather than COMP or DISPLAY).

   When no value was assigned (either by data retrieval or by moving a value) to a certain subschema item, the item contains whatever data were left over in that memory space. If a MODIFY command refers to a subschema item in this

condition, the schema value that SYSTEM 2000 software finds to be transferred to the data record will probably not conform to the data type of the schema item. In this situation, too, the command is ignored, and return code 10 results.

7. The program's current password must have U-authority (update) for each item to which the MODIFY command refers. If it does not, the command is ignored, and return code 47 results. The eighth field of the COMMBLOCK tells the position in the subschema record of the item corresponding to the item for which the program lacks authority. If, for example, the subschema item is the second one in the specified subschema record, this COMMBLOCK field contains the value of 2.

8. In queue mode, update commands are not executed as they are encountered, but rather are grouped together. All REMOVE TREE commands form one group; all INSERT commands form another group; MODIFY and REMOVE commands form a third group. When a TERMINATE command is encountered, all update commands are executed in a batch -- first the REMOVE TREE commands, then the INSERT commands, then the MODIFY and REMOVE commands. Within each group, commands are executed in the order in which they were given.

9. In queue mode, a MODIFY command must refer to the database named in the first update command of that QUEUE/TERMINATE session. If the MODIFY command refers to a different database, it is ignored, and return code 28 results.

10. In a Multi-User environment, the database can become damaged by another user before a MODIFY command is executed. In this situation, the MODIFY command results in return code 16, and the damage flag for the database is set to 1, signifying damage. The database must be restored to an undamaged state before it can be updated again.

11. The stack entry to be used in a MODIFY command might be canceled in a REMOVE TREE or MOVE TREE operation by the program or by another user in a Multi-User environment. In this situation, the MODIFY command is ignored, and return code 09 results. Also, the eighth field in the COMMBLOCK contains the database level of the removed stack entry.

12. If the specified stack number is negative or greater than 15, the command is ignored and return code 40 results.

13. In a Multi-User environment, the setting of the Multi-User execution parameter LHOLD affects the execution of a MODIFY command in immediate mode. If LHOLD is set to YES, a PLEX program must obtain a local hold on a particular data record before giving a MODIFY command for that record in immediate mode. If the program does not first obtain the local hold in this situation, the MODIFY command is ignored and return code 111 results. If, on the other hand, LHOLD is set to NO, the program does not need to obtain a local hold before giving a MODIFY command in immediate mode.

14. A MODIFY command refers only to non-key items in a data record if either 1) the specified subschema record contains only non-key items or 2) an inclusion list specifies only the non-key items in a subschema record. In a Multi-User environment, such a MODIFY command can be executed in immediate mode while retrievals are being processed for the same database and while other users have local holds elsewhere in that database. This is possible in two situations:

    a) when the Multi-User execution parameter LHOLD is set to YES and the program has already obtained a local hold for the data record to be updated

    b) when LHOLD is set to NO.

    In either situation, retrievals and updates of non-key values should be coordinated, because a PLEX program can give a MODIFY command for a non-key value that another user expects to retrieve or update. A PLEX program can, for example, change a non-key value in a record whose address is in a Locate File for another program. That other program, moreover, might need to retrieve or update the value that was changed.

15. With LHOLD set to YES, if the program gives an immediate mode MODIFY command applying to one or more key items in a data record for which the program has obtained a local hold, the execution of the command is postponed until both the following occur:

    a) a current global hold is ended

    b) any retrieval or update operations currently in progress are completed.

The following illustrations show different situations of updating with MODIFY. They are summarized here.

**Example 1:** MODIFY uses a null representation to remove a data value.

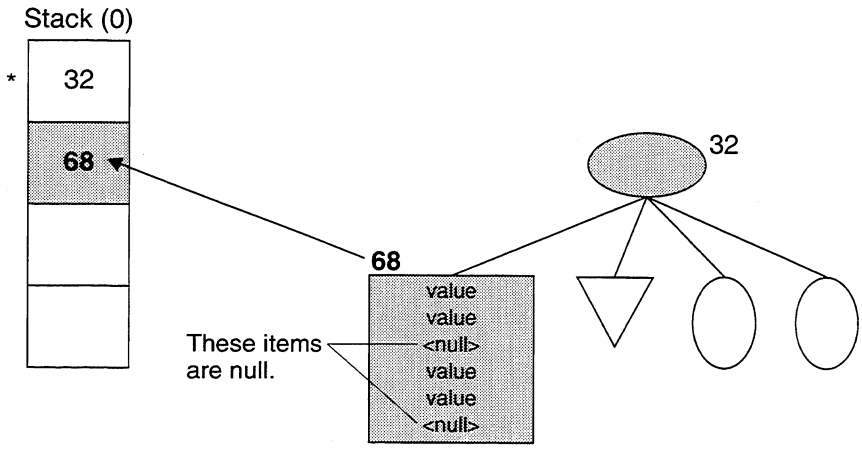
**Example 2:** The SSR for MODIFY specifies by name only three of the six items in the data record to be updated.

**Example 3:** MODIFY specifies a name table.

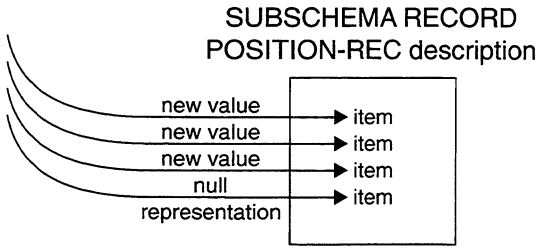
**Example 1**

A MODIFY command specifies a subschema record containing items for only four of the six items in the corresponding data record. New values are moved to three of these values, and a null representation is moved to the fourth value. When the data record is updated, the null replaces the existing value, effectively removing the value from the data record.

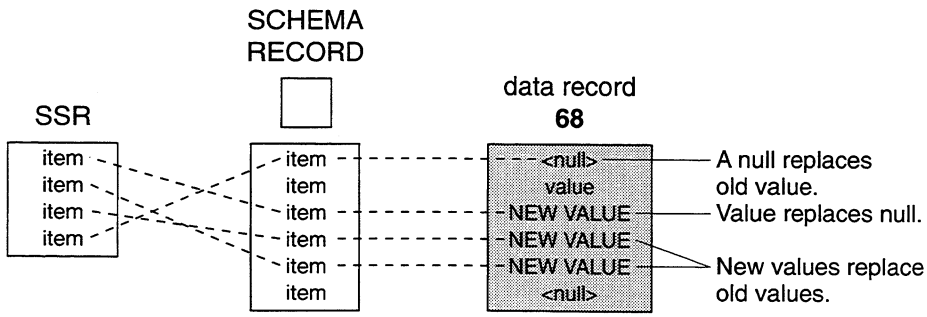
GETD POSITION-REC NEXT




The SSR POSITION-REC only has 4 of the 6 values in the SR. New values are placed in POSITION-REC.



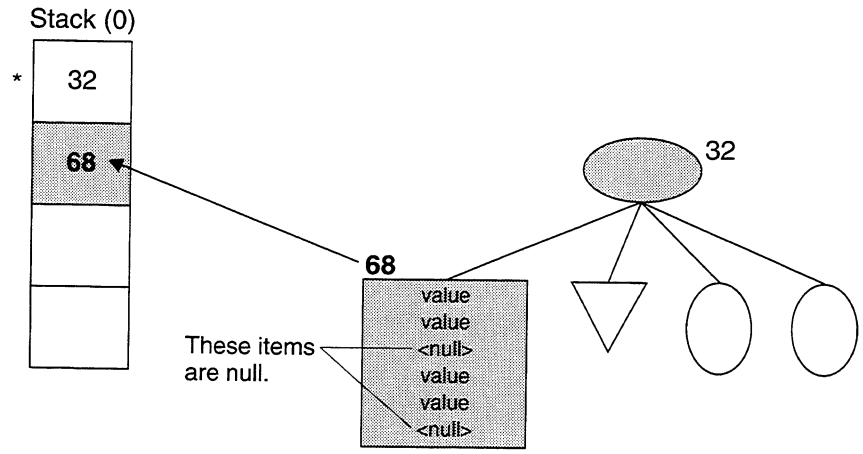

MODIFY POSTION-REC



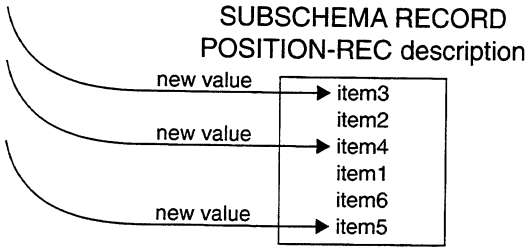
**Example 2**

The subschema record contains all six items in the corresponding schema record, but not in order. The program updates only three values in the SSR, and specifies these items by name in the MODIFY command.

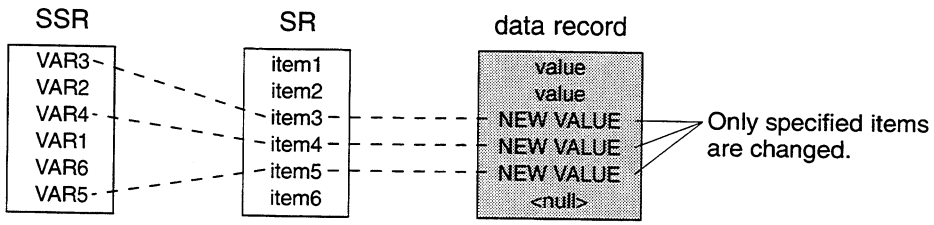
GETD POSITION-REC NEXT




New values are placed in three of the items in the POSITION-REC subschema.



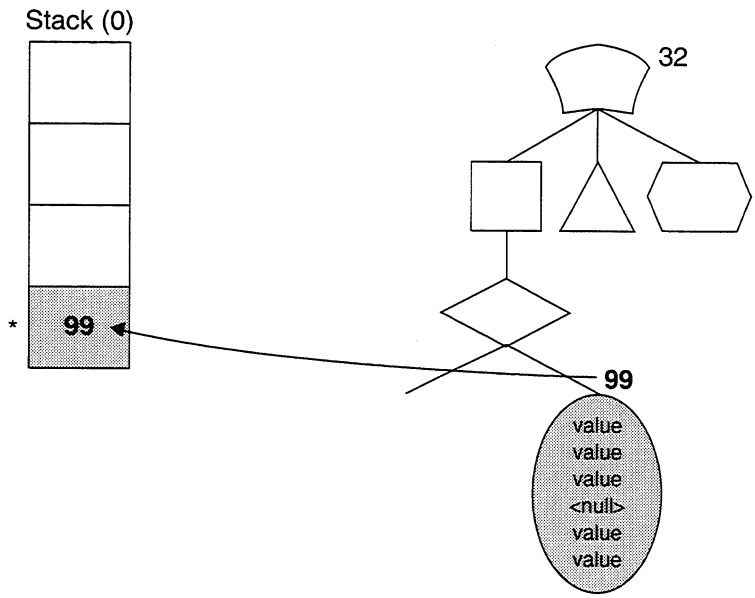

MODIFY POSITION-REC VAR3 VAR4 VAR5



**Example 3**

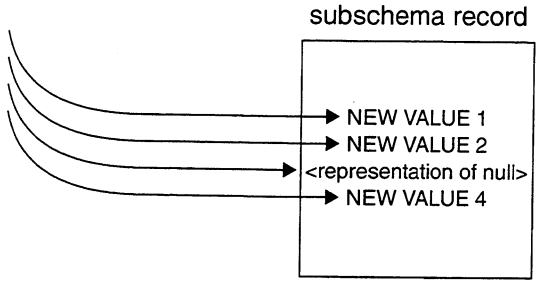
The program moves three new values and the appropriate representation of null to the subschema record. But based on the specifications in the word table for the MODIFY command, only two new values are transferred to the data record.

```
WORKING-STORAGE SECTION.
      .
      .
      .
01  PAYROLL-NAME-TABLE.
    05  PAYROLL-NAME-ITEM OCCURS 3 TIMES PICTURE X(32).
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
   MOVE 'C124' TO PAYROLL-NAME-ITEM (1).
   MOVE 'C125' TO PAYROLL-NAME-ITEM (2).
   MOVE SPACES TO PAYROLL-NAME-ITEM (3).
      .
      .
      .
   GET1 PAYROLL NEXT.
      .
      .
      .
   (commands to move values
   and null into SSR)
```

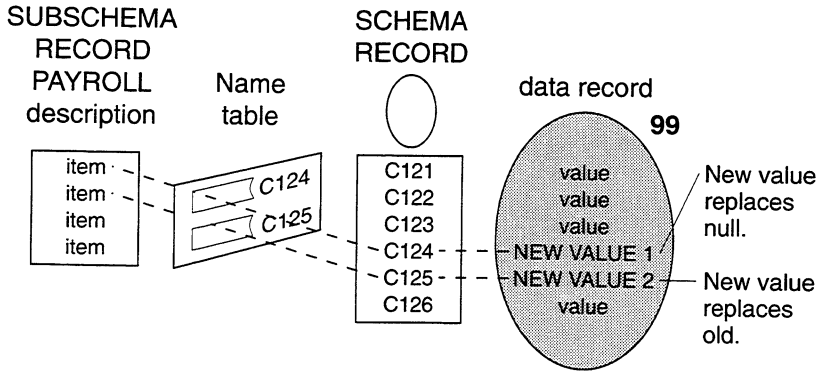


**Example 3 (continued)**

These are the contents of the PAYROLL subschema record:




```
MODIFY PAYROLL USING PAYROLL-NAME-TABLE.
```



If, however, LHOLD is set to YES and the program does not obtain a local hold for that data record beforehand, the MODIFY command is ignored and return code 111 results.

See Illus. 8.4 on page 8-139 for the disposition of requests for retrievals, holds, and updates in a Multi-User environment.

# REMOVE TREE

The REMOVE TREE command deletes a data record and its descendants from a data structure. The REMOVE TREE command has the following features:

- You can remove entire trees and subtrees of data records from a data structure with a single command.
- You can specify any one of up to 16 stacks.

First, a program gives a retrieval command to establish a stack entry (primary or secondary) for the data record that is the top node of the tree or subtree of data records to be removed. A REMOVE TREE command can then be given, specifying the stack and a subschema record for that data record at the top node.

When it is executed, a REMOVE TREE command causes a change in the data structure. SYSTEM 2000 software carries out this change by marking the Hierarchical Table entries for the removed data records so that no other records are related to them. These Hierarchical Table entries can be reused later for the insertion of new data records associated with the same schema records.

In immediate mode only, REMOVE TREE also causes the cancellation of stack entries (primary or secondary) below the level of the specified subschema record. After the command is executed, the entry for the top node just removed remains in the stack as a position for insertions or an initial position for retrievals. The entry remains in the stack until it is canceled by a command that either replaces the top-node entry in the stack or refers to another stack or database.

## Format

```
REMOVE TREE [(stack-number)] SSR
```

*stack-number* is the number of the stack to be used in the update: a constant or a variable whose value is greater than zero and equal to or less than 15, that is, $0 \leq n \leq 15$. The default value of *stack-number* is zero. For COBOL, the variable must be 01-level and can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*SSR* is the name of the subschema record description for the data record to be updated.

short form of the command and its equivalent

```
REMOVE TREE SSR.  (short form, using the default)
REMOVE TREE (0) SSR.
```

## Examples of Syntax

```
REMOVE TREE SALARY.                 (is the short form, which implies stack 0;
                                    specifies SALARY as the subschema for the
                                    record at the top node of the subtree of
                                    data records to be removed)

REMOVE TREE (6) PAYROLL.            (specifies stack 6 and the subschema
                                    record PAYROLL)
```

## Example of Usage

```
WORKING-STORAGE SECTION.
      :
COMMBLOCK OF EMPLOYEE.
01  EMPLOYEE.
      :
    03  RETURN-KODE.               (field 2)

      :
    03  NUMBER-OF-ADDRESSES.       (field 6)

      :
SSR SALARY OF EMPLOYEE.             (corresponds to the SALARY
                                    WITHIN POSITION schema record)

SSR PAYROLL OF EMPLOYEE.            (corresponds to the MONTHLY
                                    PAYROLL ACCOUNTING schema record)

      :
PROCEDURE DIVISION.
      :
   LOCATE (11) PAYROLL WHERE . . .  (retrieves into Locate File 11 the
                                    addresses of all PAYROLL data
                                    records that meet the where-clause
                                    conditions; check RETURN-KODE and
                                    NUMBER-OF-ADDRESSES)

   GET (11) PAYROLL USING 0 NEXT.   (establishes the primary stack entry
                                    for the PAYROLL data record whose
                                    address comes next in the Locate
                                    File, but retrieves no values)
```

```
GETA (11) SALARY USING 0.
```
*(establishes a secondary stack entry for the* **SALARY** *data record that is the parent of the PAYROLL data record, but retrieves no values)*

```
REMOVE TREE (11) SALARY.
```
*(deletes from the data structure the* **SALARY** *data record and all its descendants, including the* **PAYROLL** *data record whose address is the primary stack entry)*

## Considerations

1. Check the return code after the REMOVE TREE command is given. The command's return code is placed in the second field of the COMMBLOCK for the database to which the command refers.

2. The program does not need to obtain exclusive use or a global hold of the database when it gives the REMOVE TREE command. If Rollback is enabled, SYSTEM 2000 software automatically places a hold on every record being removed and on the node of detachment, that is, on a parent or sibling node.

3. The program's current password must have U-authority (update) for the schema record specified in the REMOVE TREE command, that is, for only the top node of the tree being removed. If it does not, the command is ignored, and return code 47 results.

4. Before the REMOVE TREE command is issued, the specified stack must already contain an entry for a data record corresponding to the specified SSR, that is, for the top of the tree to be removed. If it has no such entry at the level of the subschema record, the command is ignored, and return code 02 results.

5. The stack entry to be used in a REMOVE TREE command might have been canceled by a previous REMOVE TREE or MOVE TREE operation by the program or by another user in a Multi-User environment. In this situation, the REMOVE TREE command is ignored, and return code 09 results. The eighth field in the COMMBLOCK indicates the database level of the removed entry.

6. If the specified stack number is negative or greater than 15, the command is ignored, and return code 40 results.

7. In queue mode, update commands are not executed as they are encountered, but rather are grouped together. All REMOVE TREE commands form one group; all INSERT commands form another group; MODIFY and REMOVE commands form a third group. When a TERMINATE command is encountered, all update commands are executed in a batch -- first the REMOVE TREE commands, then the INSERT commands, then the MODIFY and REMOVE commands. Within each group, commands are executed in the order in which they were given.

8. In queue mode, a REMOVE TREE command must refer to the database named in the first update command of that QUEUE/TERMINATE session. If the REMOVE TREE command refers to a different database, it is ignored, and return code 28 results.

9. In a Multi-User environment, the database can become damaged by another user before a REMOVE TREE command is executed. In this situation, the REMOVE TREE command results in return code 16, and the damage flag for the database is set to 1, signifying damage. The database must be restored to an undamaged state before it can be updated again.

10. After a LOCATE command is given, a data record whose address is in the Locate File can be deleted from the database by a REMOVE TREE operation or can be moved by a MOVE TREE operation. Even after the data record is deleted or moved, its address remains in the Locate File until an ORDER BY command or another LOCATE command is given referring to that Locate File. An ORDER BY command causes a resequencing of the Locate File addresses and the removal of the addresses of any data records that were deleted or moved since the Locate File was established.

11. At some point in its processing, a program could give a REMOVE TREE command that causes the deletion of a data record whose address is in a Locate File or give a MOVE TREE command that causes a data record to be moved. While this address remains in the Locate File, it could be retrieved into a stack with a GET command given by the program. In this situation, return code 08 results.

12. The space in the Hierarchical and Data Tables that is freed by a REMOVE TREE command is not immediately reusable if more than one user has the database open. Another PLEX program might have a stack or Locate File containing the address of the removed record. It would not be appropriate to insert new, unrelated data into the record and give it a different structural position.

    If no other user has a certain database open, space from removed records will be reused. The presence of this reused address in a Locate File or stack could produce the following situation. With a REMOVE TREE command, the program using that database causes the deletion of a data record whose address is in the Locate File. While the address for the deleted data record is still in the Locate File, an INSERT command given by the program could cause this address to be reused for a newly inserted record. The presence of this reused address in the Locate File could cause problems for GET and ORDER BY commands using that Locate File. To avoid these problems, the program should coordinate its retrievals and deletions of records whose addresses are in Locate Files.

    If it is not possible for some program to predict and control record removal and reassignment for a particular schema record, the installation can request a special zap to prevent reuse of space until all users have closed a database. (Contact your SYSTEM 2000 Software Representative.) The space is reused when the database is opened again.

The following illustrations show different situations of updating with REMOVE TREE. Each example shows what happens to the stack entry for the top node of the removed tree. The examples are summarized here.

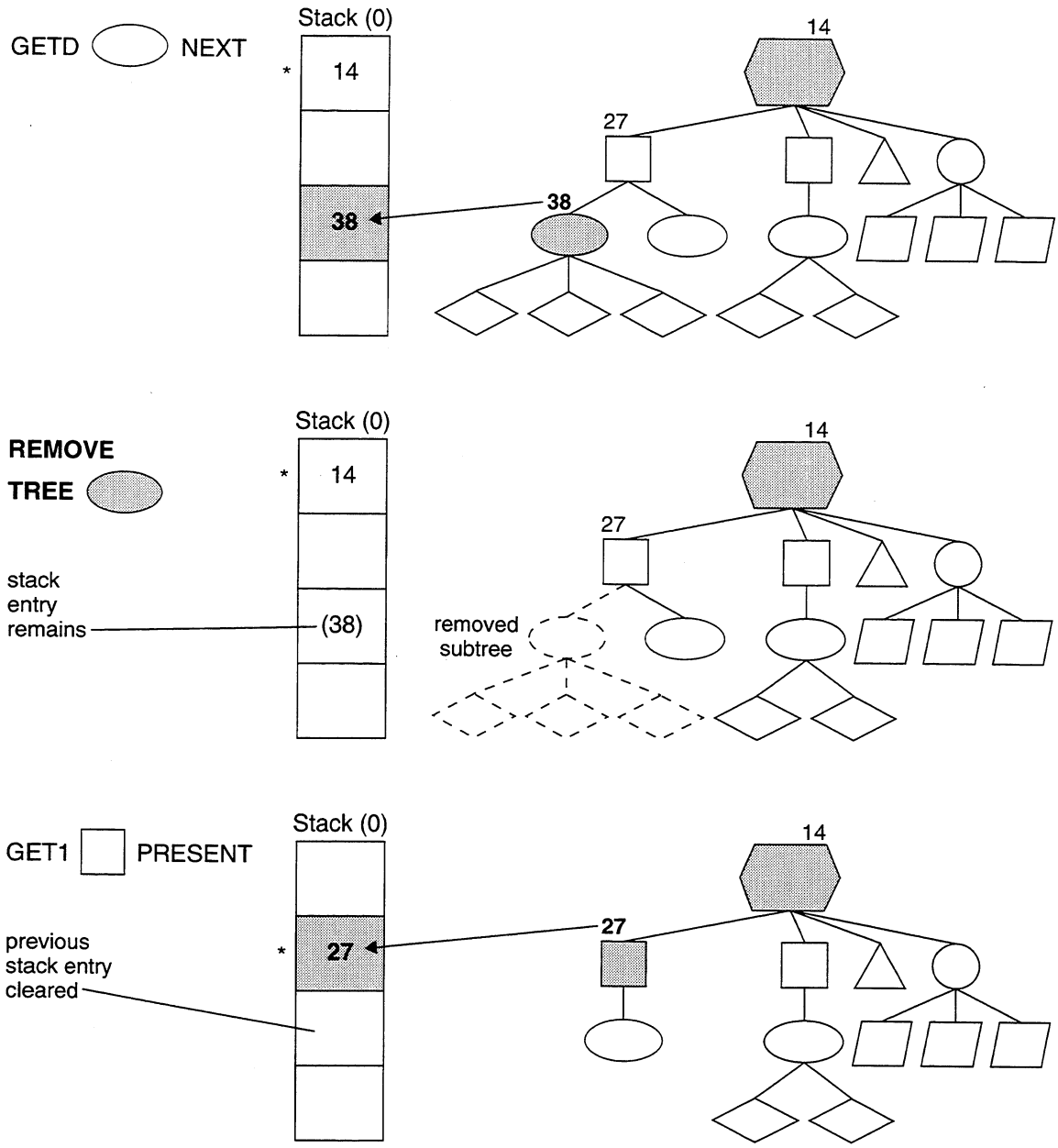
**Example 1:** REMOVE TREE removes a subtree of a logical entry.

**Example 2:** REMOVE TREE removes an entire logical entry.

**Example 3:** REMOVE TREE removes a single record.

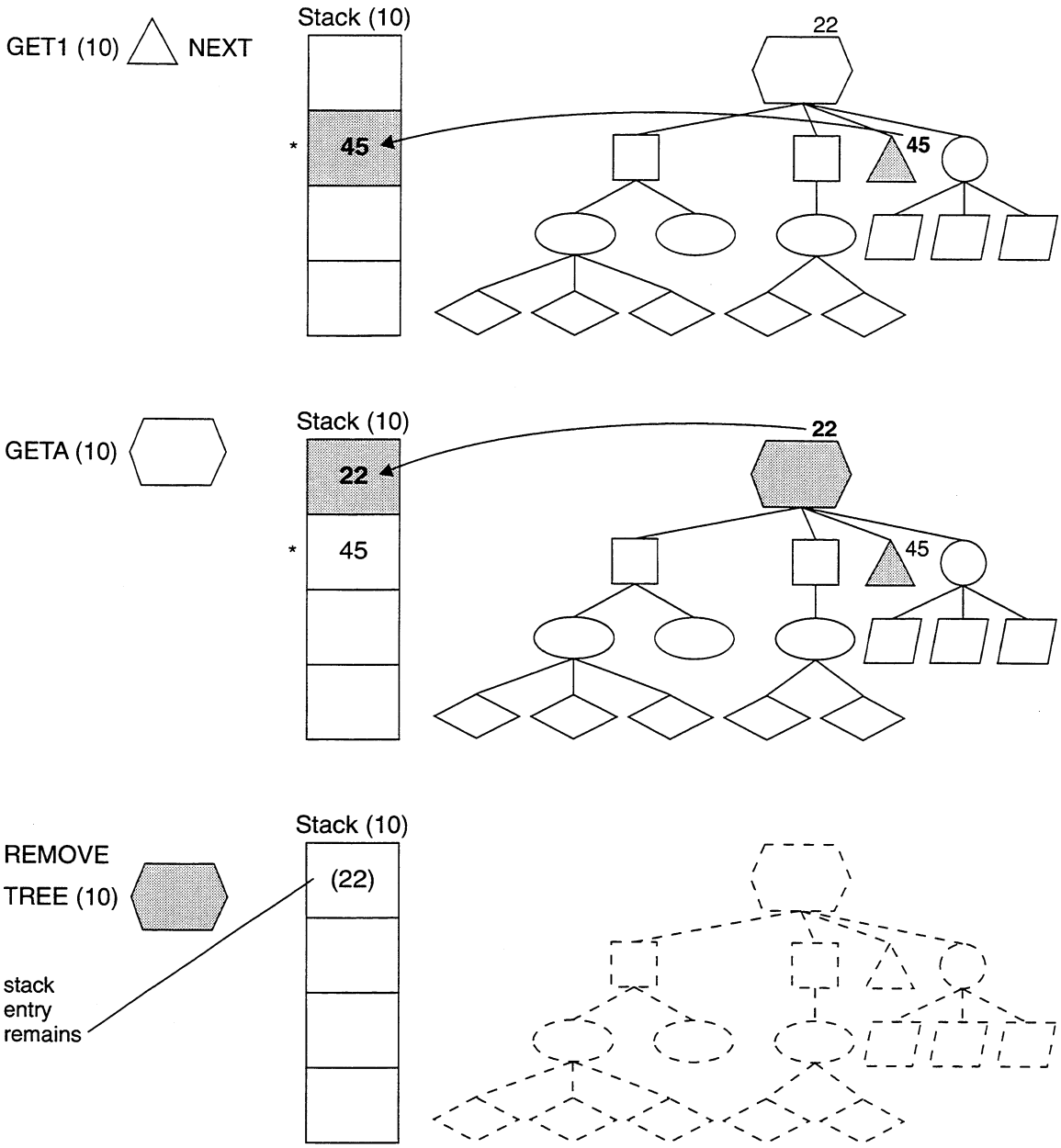
**Example 1**

In this example, a REMOVE TREE command removes a subtree of the logical entry. A GET1 PRESENT command is then issued for the same stack, clearing the stack entry for the top node of the removed tree.



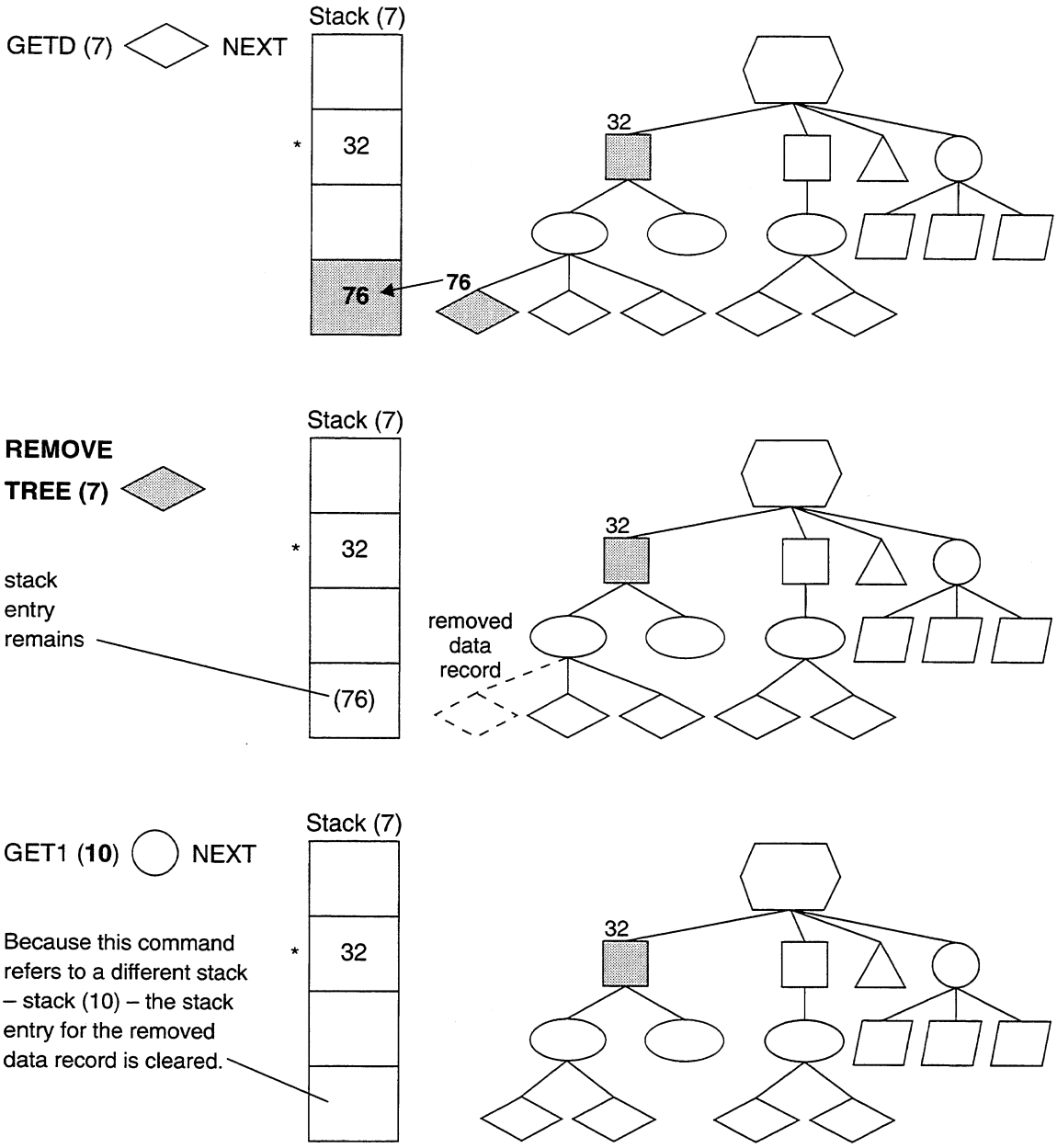
**Example 2**

In this example, a secondary stack entry above the primary is established for the root node of the logical entry. A REMOVE TREE command uses this secondary entry to remove the entire logical entry from the data structure. The entry for the root node remains in the stack.



**Example 3**

In this example, a REMOVE TREE command removes a single record from the logical entry. The stack entry for the removed record remains in the stack until a later command that refers to another stack.



# INSERT: GENERAL INFORMATION FOR ALL MODES

The INSERT command adds a data record to the data structure. The INSERT command has the following features:

- You can insert a data record at any level of the database.
- You can issue INSERT in three execution modes: immediate mode, queue mode, and load mode.
- You can specify any one of up to 16 stacks.
- You can select values for specific (rather than all) items in a subschema record for insertion by specifying an inclusion list.
- You can specify placement of a new data record with respect to a parent or sibling data record by using BEFORE or AFTER.

Before each INSERT command, the subschema record area referred to by the command must contain the values to be inserted. These values can be placed in the subschema record area with higher level language statements or with PLEX retrieval commands.

An INSERT command can refer to an incomplete subschema record, which is a subschema record that has items for some, but not all, of the corresponding items in the data record. In this situation, SYSTEM 2000 software inserts the values from the subschema record, but the values are null for schema items not referred to in the subschema record description. This is true also when an INSERT command specifies an inclusion list; only the values for subschema items indicated in the inclusion list are inserted. The other items in the new data record are null. A null can also be inserted by placing the null representation in a subschema item that is to be inserted.

The manner in which an INSERT command is processed depends on the execution mode. Subsequent topics in this section pertain to INSERT in all three modes. See **INSERT in Immediate Mode** on page 10-31, **INSERT in Queue Mode** on page 10-52, and **INSERT in Load Mode** on page 10-66 for information specific to each execution mode.

## Format

```
INSERT [(stack-number)] SSR [inclusion-list] |AFTER
                                             |BEFORE
```

*stack-number* is the number of the stack to be used in the update: a constant or a variable whose value is greater than zero and equal to or less than 15. The default value of *stack-number* is zero. For COBOL, the variable must be 01-level and can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

*SSR* is the name of the subschema record for the data record to be updated.

*inclusion-list* |*item-list*
|**USING** |*name-table*
|*word-table*

*item-list* *item-name* [, *item-list*]

*item-name* is an alias of an item in the specified subschema record.

*name-table* is the name of a table of component names or C-numbers of items to be included in the update operation. For COBOL, the entry with the OCCURS clause must specify PICTURE X(32).

*word-table* is the name of a table of one-word indicators (1s and 0s), one indicator for each item in the specified subschema record. An item is included in the update operation if its indicator in the table is one. Conversely, an item is excluded from the operation if its indicator is zero. In each indicator, the low-order bit is checked to see if it is zero or one. For COBOL, the entry with the OCCURS clause must specify COMP usage.

short form of the command and its equivalents

**INSERT** *SSR*. *(short form, using both defaults)*

**INSERT (0)** *SSR*.

**INSERT** *SSR* **AFTER.**

**INSERT (0)** *SSR* **AFTER.**

## Examples of Syntax

**INSERT SALARY.**

*(the short form, implies stack 0 and AFTER; specifies that all values in the* **SALARY** *subschema record area are to be inserted as a data record; in immediate mode: the placement of this data record is either after the last sibling if the stack has an entry for the parent only, or after the sibling whose address is a stack entry; in queue mode and load mode: data record is placed after all the children of the parent data record)*

**INSERT (5) POSITION-REC.**

*(specifies stack 5 and the* **POSITION-REC** *subschema record; implies AFTER)*

**INSERT PAYROLL BEFORE.**

*(specifies the* **PAYROLL** *subschema record and BEFORE; implies stack 0)*

```
INSERT (6) SKILL YEARS.
```

*(specifies stack 6 and the* **SKILL** *subschema record; also states that the value of the subschema record variable* **YEARS** *is the only value in the new data record to be inserted; all other items of the new data record are null)*

```
INSERT EDUCATION USING
   NAME-TABLE AFTER.
```

*(specifies the* **EDUCATION** *subschema record and the name table called* **NAME-TABLE;** **AFTER** *implies stack 0)*

## Considerations

1. Check the return code after each INSERT command is given. The command's return code is placed in the second field of the COMMBLOCK for the database to which the command refers.

2. Data records must be inserted according to the hierarchy of schema records in the schema tree. That is, after one data record in the insert sequence for a logical entry, the next data record to be inserted must be either a descendant, sibling, or ancestor of the previous data record. If it is not, the command is ignored, and return code 21 results.

   The data record referred to in the ignored command could be the ancestor of other data records to be inserted afterward. In this situation, the insertion of those other data records would not be allowed since they would not be in logical order according to the schema tree. The INSERT commands for these descendants would also be ignored, and return code 21 would result.

3. The program's current password must have U-authority (update) for each non-null schema item value to which the INSERT command refers. If it does not, the command is ignored and return code 47 results. The eighth field of the COMMBLOCK tells the position in the subschema record of the item for which the program lacks authority. If, for example, the item is the second one declared in the specified subschema record, the eighth COMMBLOCK field will have the value of 2.

4. Each subschema item referred to in an INSERT command must have a value that conforms to the definition of the corresponding item in the database. If it does not, the command is ignored and return code 10 results. (However, for COBOL, if a value in a COMP-3 subschema item field is not in packed-decimal format, an INSERT command referring to that subschema item results in completion code OC7.)

   For COBOL, INSERT processing of a numeric value is most efficient when it appears in a subschema item with usage COMP-3 (rather than COMP or DISPLAY).

   When no value is moved to a certain subschema item, the item contains whatever data were left over in that memory space. If a MODIFY command refers to a subschema item in this condition, the value that SYSTEM 2000 software finds there to be transferred to the data record will probably not conform to the data type of the schema item. In this situation, too, the command is ignored, and return code 10 results.

The tree of data records to be added by an insert sequence will not include a data record with a faulty subschema item value. A subsequent INSERT command in the sequence for a descendant of the rejected data record might then also be rejected because its parent is missing. If the rejected data record is at or above the level of the topmost data record in the new tree, the insert sequence is ended, and an entry for the topmost data record is made in the stack.

5. An INSERT command that is rejected because of an invalid item or SSR may or may not end an insert sequence, depending on the level of the specified SSR. If the SSR of a rejected INSERT command is at or above the level of the topmost data record in the insert sequence, the insert sequence is terminated. However, if the INSERT SSR is below the topmost level of the insert sequence, the insert sequence is not terminated and the stack is not altered.

6. The picture or type of each subschema item referred to by an INSERT command must be compatible with the definition of each corresponding schema item. If it is not, return code 10 results and the command is ignored, with the same consequences for the insert sequence as are discussed in consideration 2.

7. Although you can include the keyword USING in an INSERT command when it refers to an inclusion list, you cannot specify the phrase USING 0 with INSERT or any other update command. If you do, the PLEX processor gives an error message for the command.

8. For a discussion of inclusion lists, see **Inclusion List** on page 8-121.

# INSERT IN IMMEDIATE MODE

This section discusses the INSERT command in immediate mode. It includes the following topics:

- a general discussion of immediate mode INSERT processing
- special considerations for immediate mode
- an example command sequence
- illustrated examples of insert sequences and the placement of insertions as determined by command syntax and stack entries.

For general information on the INSERT command, including command syntax, see **INSERT: General Information for All Modes** on page 10-28.

## Processing an Insert Sequence

A program usually issues a series of INSERT commands to add trees of data records to selected places in the database. The series of commands is called an *insert sequence.* Each insert sequence represents one new tree of data records. Before a program issues the first INSERT command (to insert the topmost data record in the new tree) in an insert sequence, a retrieval command should be given to establish a stack entry for the parent or a sibling of the topmost data record to be inserted. If the topmost data record is at level 0, the stack must have a primary entry for a level 0 sibling. In any case, immediate mode insertions can be made either at or below the level of the primary stack entry, but not above it.

If the program is inserting a number of new trees in succession, without intervening retrievals, the sibling or parent entry in the stack needs to be established only to begin inserting the first tree. Thereafter, as each succeeding tree is being inserted, the stack automatically has an entry for the topmost data record in the most recently inserted tree.

In immediate mode, INSERT commands in an insert sequence are handled differently from all other PLEX commands. SYSTEM 2000 software collects the data to be inserted as the program gives each INSERT command until the insert sequence is ended. At this point, SYSTEM 2000 software actually inserts the entire tree of data records into the database.

## Ending an Insert Sequence

An insert sequence is ended by any of the following situations:

- an INSERT command for the stack used in the insert sequence but referring to a data record that is not part of the same tree, that is, a data record that is either on the same level of the database as the topmost data record of the inserted tree or on a higher level (an ancestor)
- an INSERT command referring to the same stack, but to a subschema record for a different database
- an INSERT command referring to another stack
- any update command, other than INSERT, referring to any stack
- one of these commands, if it refers to the same stack: GET1, LOCATE, GET, GETD, GETA, GETP, REMOVE, MODIFY, REMOVE TREE, MOVE TREE or ORDER BY
- one of these commands, if it refers to the same database: CLOSE, OPEN, KEEP, APPLY, SUSPEND, SAVE, RELEASE, or VALUES PADDING
- one of these commands: CANCEL QUEUE, COPYL, COPYS, CLEAR (all forms), ENABLE/DISABLE ROLLBACK, FRAME/END FRAME, LOAD, QUEUE, START S2K, STOP S2K, or TERMINATE.

As this list shows, an insert sequence is not ended by a GET1, LOCATE, GET, GETA, GETD, or GETP command referring to another stack for the same or a different database. Such a retrieval command is executed at the time it is issued. Thus, a program can retrieve data during an insert sequence from the same or a different database. It can also use that retrieved data to create new data records to be inserted in the same sequence.

**Note**: A retrieval command that does not end the insert sequence cannot retrieve the data being inserted, even though you might logically consider the insert to have taken place. If your program retrieves from one database, then issues a command ending an insert sequence for another database, the password for the database being updated need not be present in the associated COMMBLOCK.

As the list also shows, the FOR, LINK, and FULL PASSES commands do not end insert sequences.

When an insert sequence ends, SYSTEM 2000 software executes the INSERT commands as a group. It updates the database for the entire tree of that insert sequence in one operation by changing the Hierarchical Table, the Data Table, and the Index, if necessary, to reflect the new data values and structure. The database cycle number is incremented by one.

After the INSERT sequence is executed, the stack contains an entry for the topmost data record in the newly inserted tree. Any entries that were at deeper levels in the stack before the tree insertion are canceled.

## BEFORE and AFTER

The placement of a data record to be inserted is specified with BEFORE or AFTER in an INSERT command. AFTER is the default. BEFORE and AFTER indicate the placement of the insertion relative to a current stack entry. Prior to an insert sequence that begins with an insertion at level 0, the stack must contain an entry for a sibling of the new level 0 data record to be inserted. The insertion of that data record will be made at level 0 before or after the sibling, whichever the INSERT command indicates.

When an insert sequence begins with an insertion below level 0, however, the stack must have an entry for either the parent or a sibling of the topmost data record in the insert sequence. If the stack has an entry for only the parent, BEFORE or AFTER in the INSERT command tells where, in logical order among the parent's children, the new data record is to be inserted. INSERT BEFORE inserts the new data record before all other children of that parent in logical order; INSERT AFTER inserts the new data record after the last child (in logical order) that corresponds to the same schema record.

If the topmost data record in the insert sequence is below level 0, the stack can contain an entry for a sibling of that data record. In this situation, INSERT BEFORE inserts the new data record in logical order just before the sibling; INSERT AFTER places the new data record just after the sibling. If the stack contains an entry for both the parent and a sibling, the insertion is made relative to the sibling.

BEFORE or AFTER is ignored if it is specified in an INSERT command for a descendant of the top node of the new tree. Each descendant is inserted in hierarchical order according to the sequence of SSRs in the INSERT commands. However, if an INSERT command specifies an SSR at the level of the most recently inserted top node, BEFORE or AFTER determines placement as described above.

As SYSTEM 2000 software processes an insert sequence, it uses the stack to ensure that each record in turn is an ancestor, sibling, or descendant of the previous record. After the sequence is processed, the stack contains an entry for the topmost record in the sequence, as well as any entries above the level of that entry that were present before INSERT processing. All entries below the level of the topmost record are canceled.

## Example of Usage

```
WORKING-STORAGE SECTION.
      :
      :
COMMBLOCK OF EMPLOYEE.
      :
      :
SSR EMP OF EMPLOYEE.
01  EMP.                          (corresponds to the level 0 schema record)
    03  EMPLOYEENUM . . .
        EQ EMPLOYEE NUMBER.
      :
      :
SSR C100 OF EMPLOYEE.              (corresponds to the POSITION WITHIN
                                   COMPANY schema record)
      :
      :
01  EMPNO . . .                   (is the variable that contains the
                                   required employee number)
PROCEDURE DIVISION.
      :
      :
   (Input value for EMPNO)
      :
      :
   GETP EMP USING 0 WHERE          (establishes a primary stack entry for
   EMPLOYEENUM EQ EMPNO.           the level 0 data record that has the
                                   employee number just entered. Return code
                                   checking must allow for the non-existence
                                   of the employee number)

   (Input new values into SSR C100)

   INSERT C100 BEFORE.             (adds a new position data record
                                   immediately ahead of the C100
                                   data record that is currently
                                   first in logical order)
      :
      :
```

## Considerations

1. See **INSERT: General Information for All Modes** on page 10-28 considerations for the effect of a rejected INSERT command in an insert sequence.

2. Before the program issues an insert sequence, the stack must contain an entry for the parent or a sibling of the topmost record to be inserted. If the topmost record is at level 0, the stack must have a primary entry for the level 0 sibling. If the stack does not have the required entry, or if the level of the specified SSR is above the level of the primary, the INSERT command is ignored and return code 03 results. If the data record was to be inserted at or above the level of the topmost data record in the preceding insert sequence, the sequence is ended, and an entry for the topmost data record is made in the stack.

3. A program can acquire a local hold on the parent or sibling of the top node of an insert sequence by giving a retrieval command with the HOLD option before it gives the first INSERT command in the sequence. If LHOLD=YES, the local hold is required unless the database is under a global hold or exclusive use or unless Rollback is enabled. If the MULTIPLE HOLDS option is enabled, more than one record can be held in a database or in several databases. For a discussion of local holds, see **HOLD Option** on page 8-131.

4. During an insert sequence, the stack entries below the top level retain the tree structure for checking a valid tree. When the user changes stacks, the tree structure stack entries are preserved as regular stack data, that is, for subscript (0), 32 levels are retained, and for each non-zero subscript, 10 levels are retained. The top level of the 10 levels is the highest non-zero entry in the stack, whether it is a position entry or a tree structure entry. If the combination of position(s) plus insert sequence tree structure entries spans more than 10 levels, the insert sequence must be associated with subscript (0) to avoid rejection of lower branches in the INSERT tree after a change in subscript. If your program is inserting without intervening changes in subscript, all 32 levels are available for retaining data for tree structure checking.

5. If the stack for the first INSERT command in an insert sequence does not have an entry for a similar sibling or a parent of the record to be inserted, the command is ignored, and return code 03 results. Even if the stack has such an entry, it might have been canceled by a REMOVE TREE or MOVE TREE command using another stack or issued by another user in a Multi-User environment. If this happens, the INSERT command is ignored, and return code 09 results. Also, the eighth field in the COMMBLOCK indicates the database level of the removed parent or sibling entry.

6. The stack entry for a similar sibling to be used for the first record in an insert sequence might have been canceled by a REMOVE TREE or MOVE TREE command given for the same stack. In the meantime, if no retrieval command for the same stack replaced the canceled or moved entry and the program did not refer to another stack, the canceled or moved entry can still be used for the first insertion. If the first command in the insert sequence is INSERT AFTER or is an INSERT BEFORE command for a level 0 subschema record, the topmost record is placed in the position in the logical data structure formerly occupied by the removed similar sibling. However, if the INSERT BEFORE command is for a non-level-0 subschema record, the canceled stack entry for the removed or moved sibling is ignored, and the insertion is made relative to the parent; the topmost record in the sequence is inserted as the first child under the parent.

7. In a Multi-User environment, even if the stack has an entry to be used when the first INSERT command is issued in an insert sequence, that entry can later be canceled before the insert sequence is ended. This cancellation can result from a REMOVE TREE or MOVE TREE operation by another user, or it can be caused by the same program in a REMOVE TREE or MOVE TREE command referring to another stack. If this happens, the command that ends the insert sequence is ignored, and return code 15 results. The insert sequence itself is also ignored. Although the command ending the insert sequence was ignored, the program can then issue it again.

   Return code 15 applies to the situation in which an otherwise acceptable command is rejected in order to indicate the cancellation of the insert sequence. For this command, certain fields of its COMMBLOCK are set as follows:

   - the eighth field of the COMMBLOCK indicates the database level of the canceled stack entry.
   - the first field of the COMMBLOCK (subschema record name) is cleared.
   - the fourth (last record flag), sixth (number of located records), and seventh (Locate File pointer) fields are for a Locate File referred to most recently before the command that ended the insert sequence.
   - the ninth through the fourteenth fields are set for the database referred to in the insert sequence. These fields might not contain meaningful information in this situation if the command referred to some database other than the one for the insert sequence. In addition, entries in the stack used for the insert sequence are canceled below the entry referred to for insertion of the top node.

8. The space in the Hierarchical and Data Tables that is freed by a REMOVE TREE command is not immediately reusable if more than one user has the database open. Another PLEX program might have a stack or Locate File containing the address of the removed record. It would not be appropriate to insert new, unrelated data into the record and give the record a different structural position.

   If no other user has a certain database open, space from removed records will be reused. The presence of this reused address in a Locate File or stack could produce the following situation. With a REMOVE TREE command, the program using that database causes the deletion of a data record whose address is in a Locate File. While the address for the deleted data record is still in the Locate File, an INSERT command given by the program could cause this address to be reused for a newly inserted record. The presence of this reused address in the Locate File could cause problems for GET and ORDER BY commands using that Locate File. To avoid these problems, the program should coordinate its retrievals and deletions of records whose addresses are in Locate Files.

   If it is impossible for some program to predict and control record removal and reassignment for a particular schema record, the site can request a special zap to prevent reuse of space until all users have closed a database. (Contact your SYSTEM 2000 Software Representative.) The space is reused when the database is opened again.

9. In a Multi-User environment, before an insert sequence ends, the database can become damaged by another user. In this situation, the command ending the insert sequence is rejected. In addition, the damage flag is set to 1 in the COMMBLOCK of the command that ended the insert sequence. The database must be restored to an undamaged state before it can be updated again.

10. In a Multi-User environment, the database can become damaged by another user before the first INSERT command of an insert sequence is issued. The first INSERT command results in return code 16, and the damage flag is set to 1 in the COMMBLOCK for the command.

## Examples of Insert Sequences

The illustrations show examples of insert sequences and the placement of newly inserted records in a logical entry. They are summarized here.

**Example 1:** the insert sequence is ended by another INSERT at tree-top level.

**Example 2:** the insert sequence is ended by commands referring to the tree-top level, but not by a command for another stack.

**Example 3:** the top record insert sequence contains some nulls because of the SSR description.

**Example 4:** the record in the insert sequence contains some nulls because of the specified word table.

**Example 5:** the insert is at the same level as the stack entry.

**Example 6:** the insert is for a sibling of a stack entry.

**Example 7:** the insert occurs after a removed record.

**Example 8:** the insert is before a position in the stack.

**Example 9:** the INSERT BEFORE is for a record that has no similar siblings.

**Example 10:** the INSERT BEFORE is for a record that has a similar sibling.

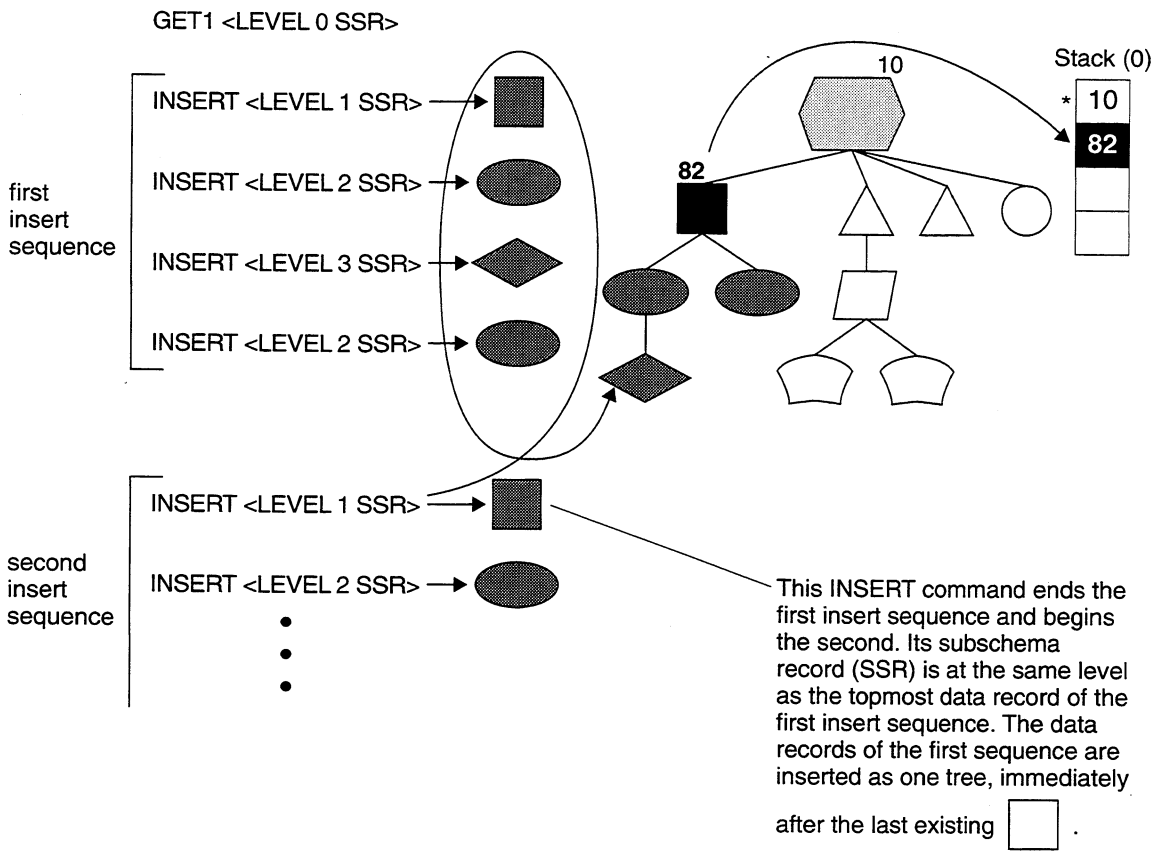
**Example 11:** the INSERT AFTER is for a record that has no similar sibling.

**Example 12:** the INSERT AFTER is for a record that is similar to the primary record in the stack.

**Example 13:** the INSERT BEFORE is for a record that is similar to the primary record in the stack.

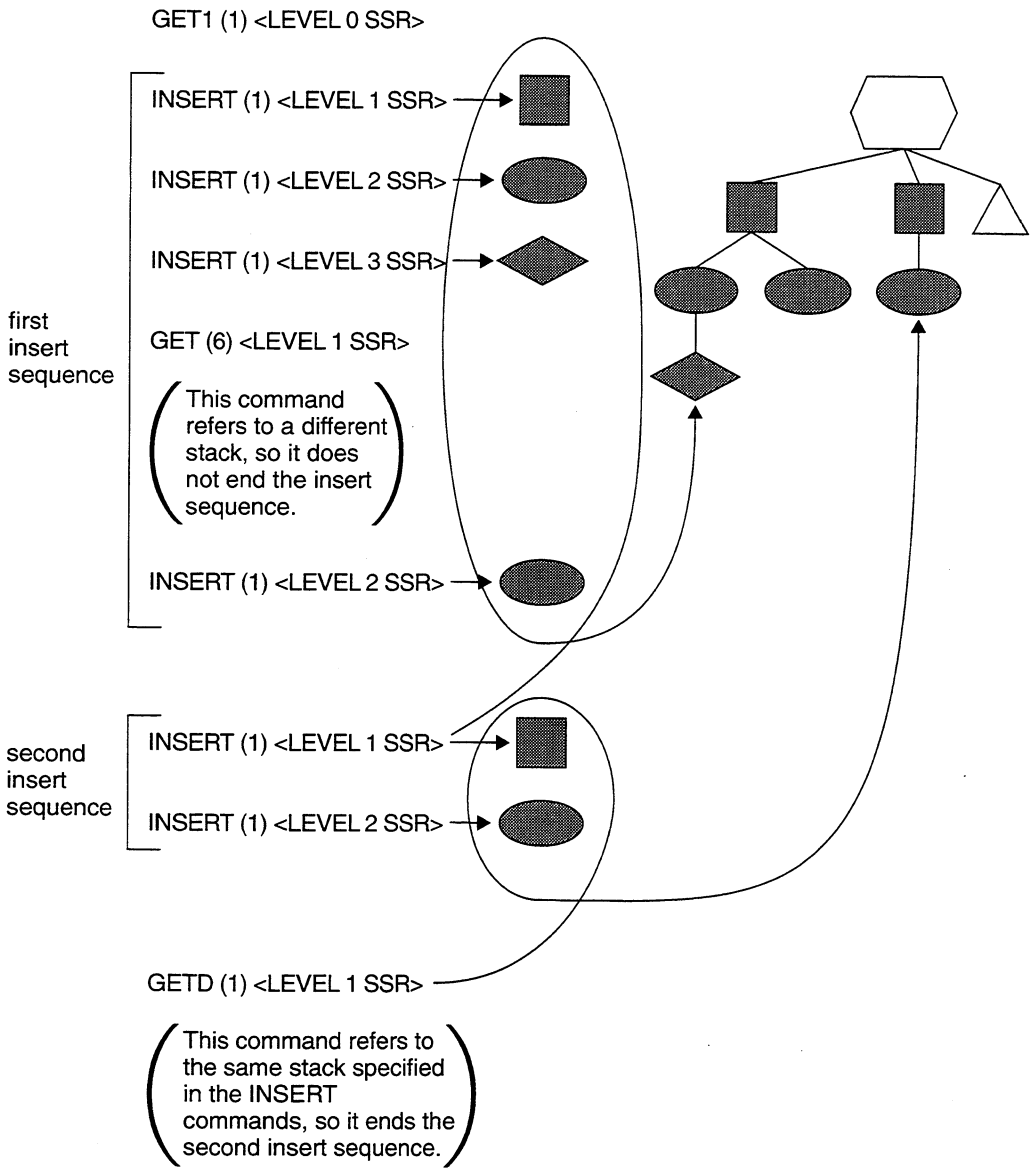
**Example 1**

In this example, the insert sequence is ended by giving an INSERT command for another record at the level of the top node of the first tree.



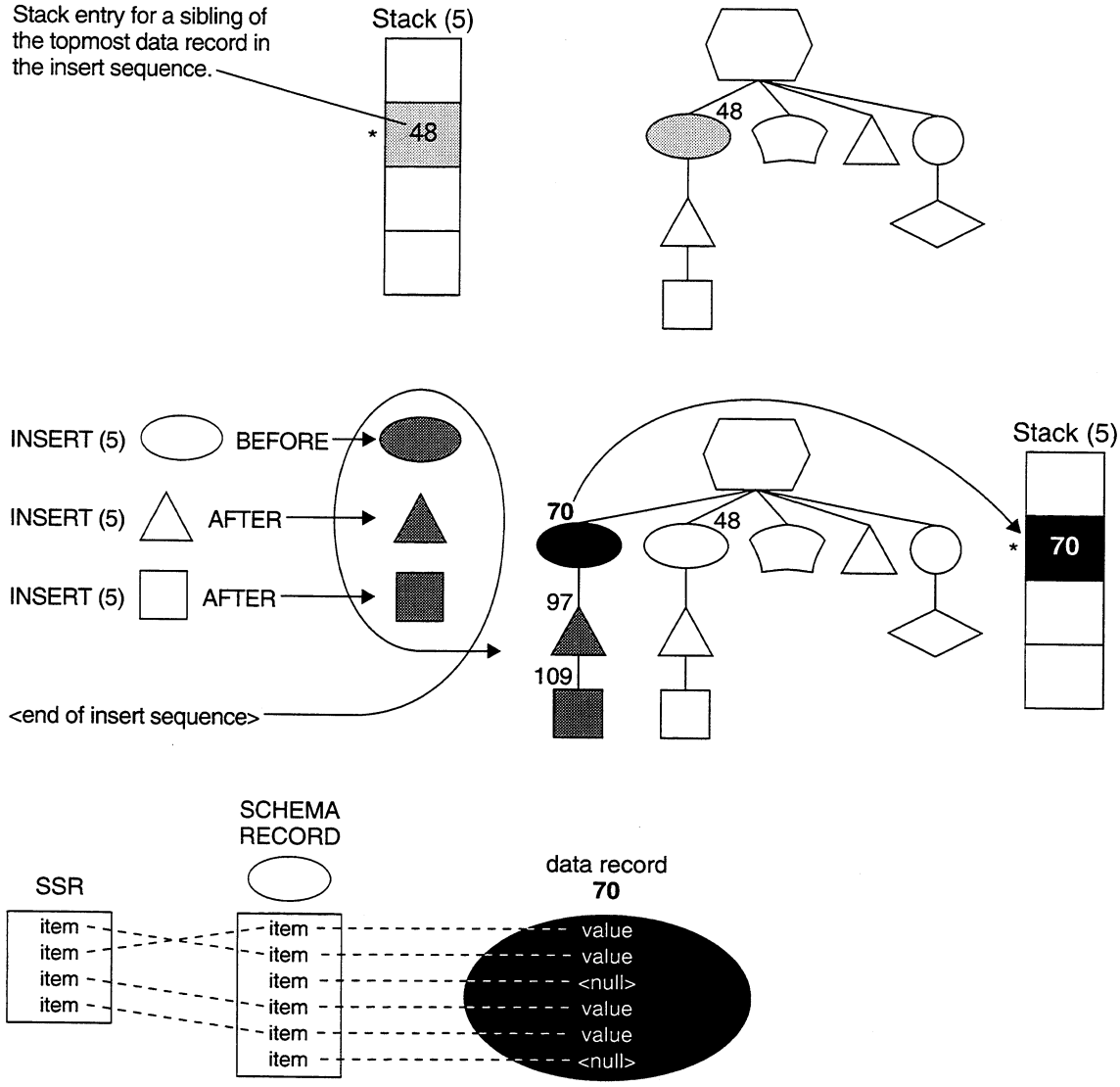
### Example 2

In this example, the insert sequence ends with INSERT and GETD commands specifying an SSR at the level of the tree-top SSR, but not by a GET command referring to another stack.



**Example 3**

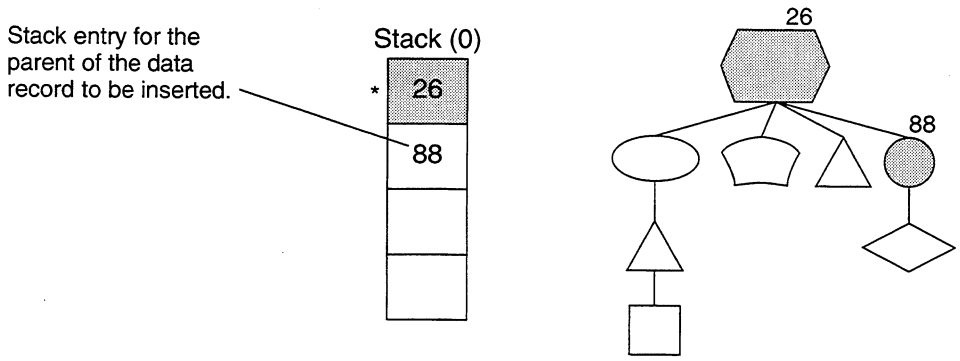
Within the insert sequence in this example, the top node record contains two nulls, because the specified SSR has items for four of the six items in the corresponding schema record.



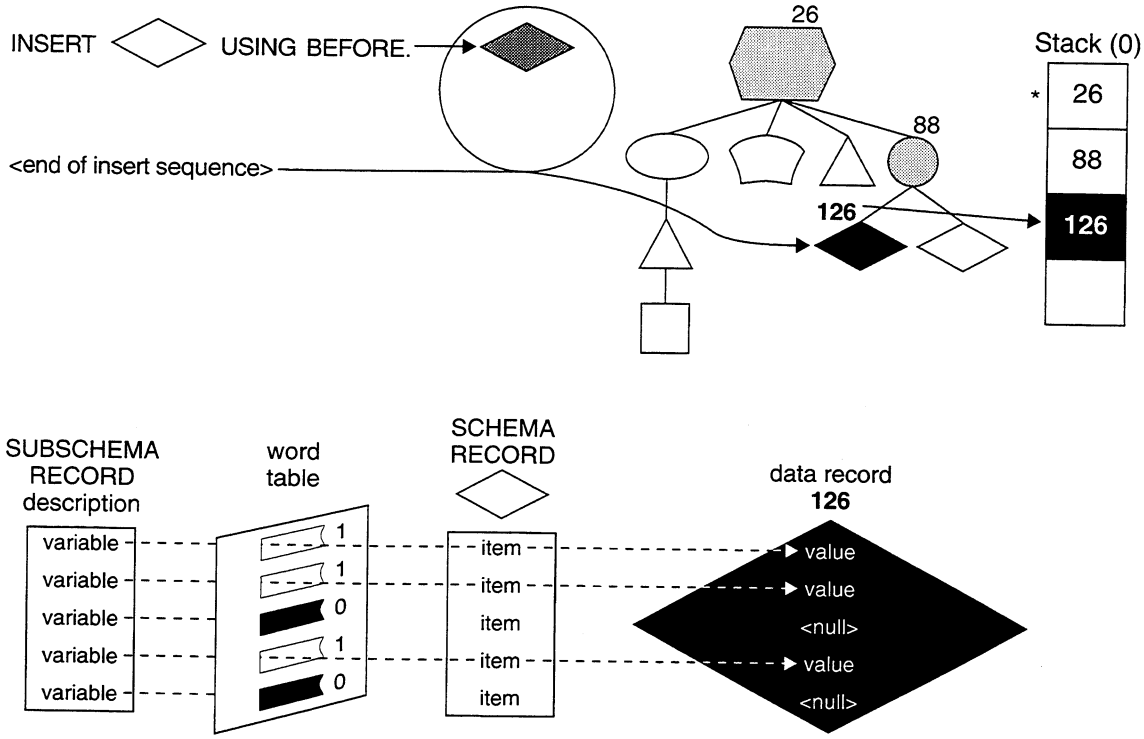
### Example 4

This example shows a one-record insert sequence. Based on the specified word table for the INSERT command, two of the five items in the new record are null.

```
WORKING-STORAGE SECTION.
      .
      .
      .
01  WORD-TABLE.
    05  WORD-ITEM OCCURS 5 TIMES PIC S(9) COMP.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
   MOVE 1 TO WORD-ITEM (1).
   MOVE 1 TO WORD-ITEM (2).
   MOVE 0 TO WORD-ITEM (3).
   MOVE 1 TO WORD-ITEM (4).
   MOVE 0 TO WORD-ITEM (5).
      .
      .
      .
```



**Example 4 (continued)**

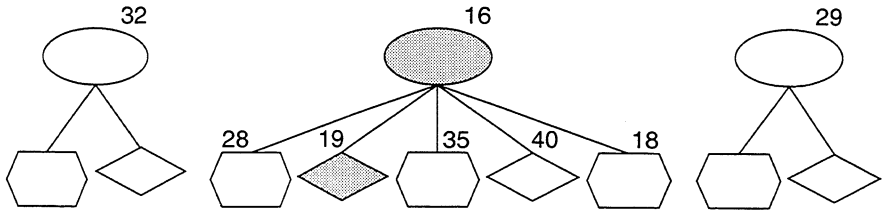


### Example 5

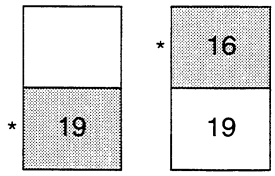
Here, the INSERT is at the same level as the stack entry. A search through the children of 16 is not necessary.

**Note**: If there is no ◇ record in the stack and no primary stack entry, return code 03 is issued.

Given these data trees




and either one of these stacks,

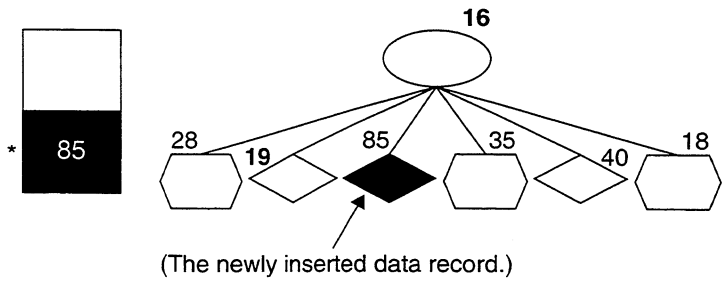



the commands

**INSERT ◇ AFTER**

**GET1 ◇ PRESENT**

give this result:



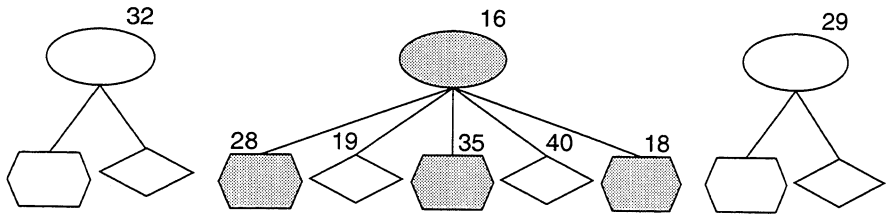
**Example 6**

This INSERT command uses the position in the stack at the level of the INSERT to determine the place in the sibling chain where the insert occurs. If no address exists in the stack at the level of the INSERT, the entire sibling chain is searched until the last child is found.

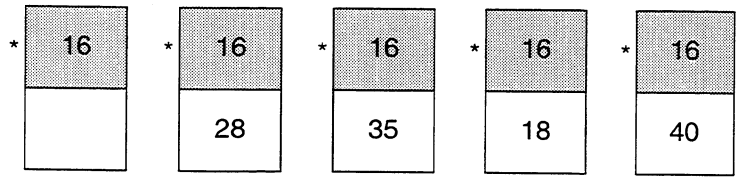
**Note**: With the primary stack at 16, the following applies:

- If there is no secondary position in the stack, the insert occurs after the last ◇ child.
- If the secondary stack is at any child that is not type ◇, the insert occurs after the last ◇ child.
- If the secondary is at a child of type ◇, the insert occurs immediately after the ◇ in the stack.

Given these data trees




and any one of these stacks,

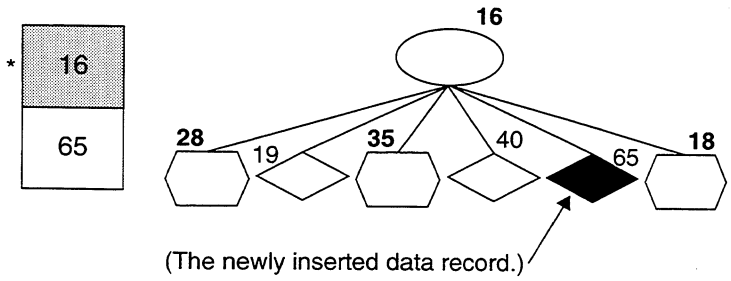



the commands

**INSERT** ◇ **AFTER**

**CLEAR**

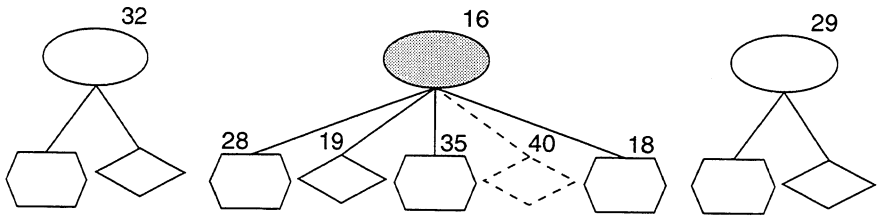
give this result:



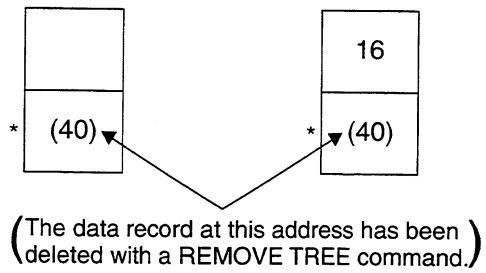
### Example 7

In this example, data record 40 was removed. The INSERT command searches through the children of 16 to find the record that immediately precedes the removed record 40. The insert occurs at that point; in this example, between 35 and 18.

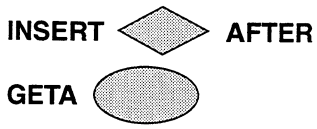
Given these data trees



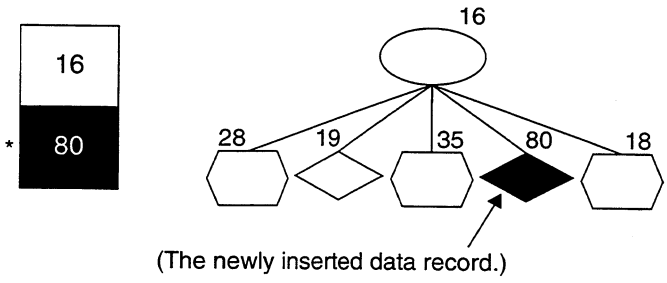

and either one of these stacks,




the commands



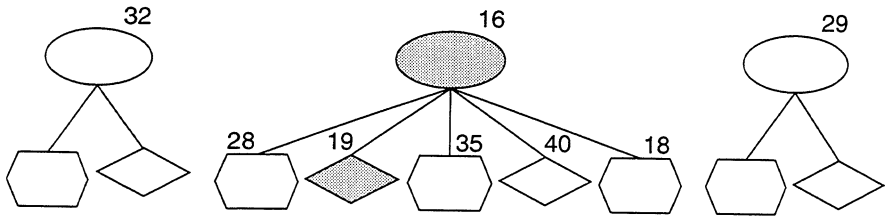

give this result:



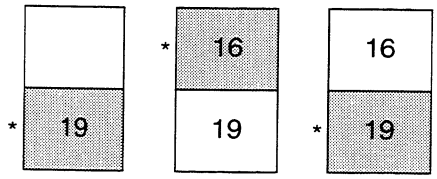
**Example 8**

In this example, the INSERT command inserts the new record immediately before 19, which is a stack entry. The GET1 PRESENT command makes the inserted record the primary stack entry.

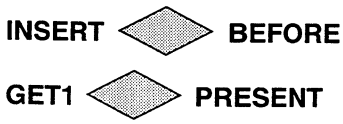
Given these data trees



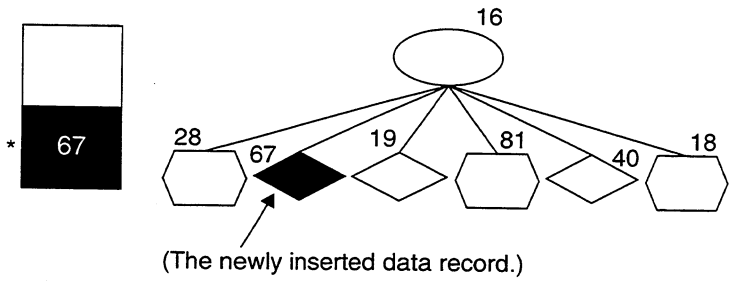

and any one of these stacks,




the commands



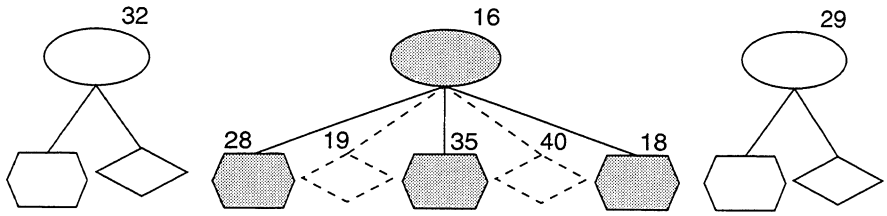

give this result:



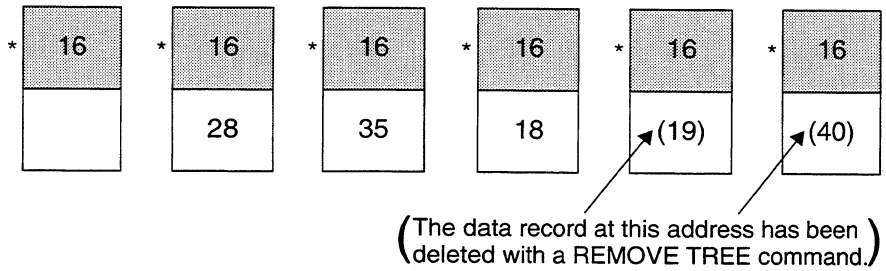
### Example 9

In this example, an INSERT BEFORE is shown with several possible stacks. Because the primary stack entry is always at level 0, and the secondary stacks are either a deleted record or not the type being inserted, the insert always occurs as the first child.

Given these data trees




and any one of these stacks,

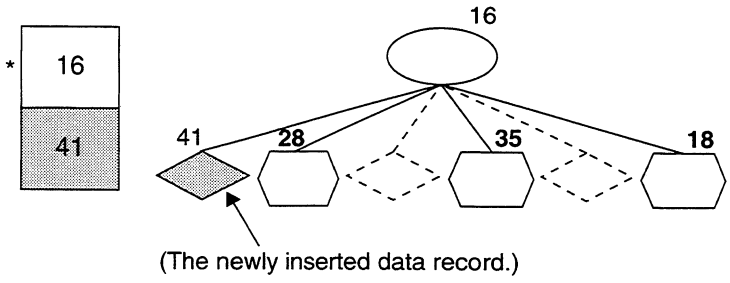



the commands

**INSERT** ◇ **BEFORE**

**CLEAR**

give this result:

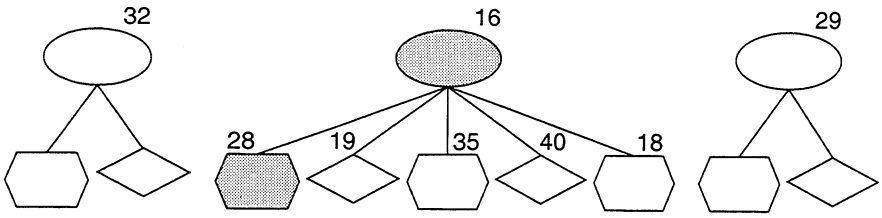



**Note**: If records 19 and 40 had not been deleted, the last two sample stacks would result in an insert before record 19 and before record 40, respectively.

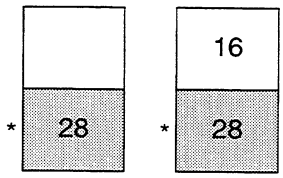
**Example 10**

In this example, an INSERT BEFORE followed by a GET1 PRESENT yields a new primary stack entry.

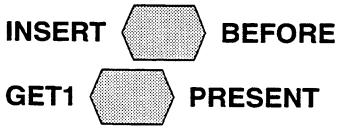
Given these data trees



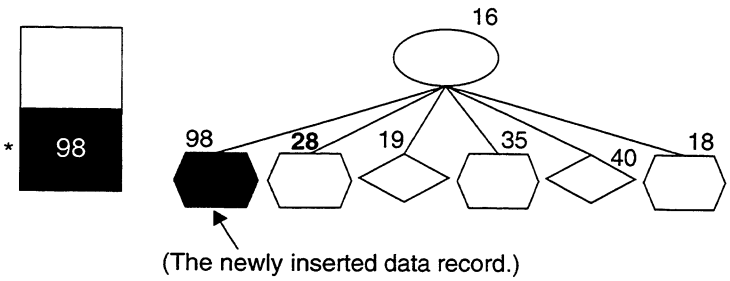

and either one of these stacks,




the commands



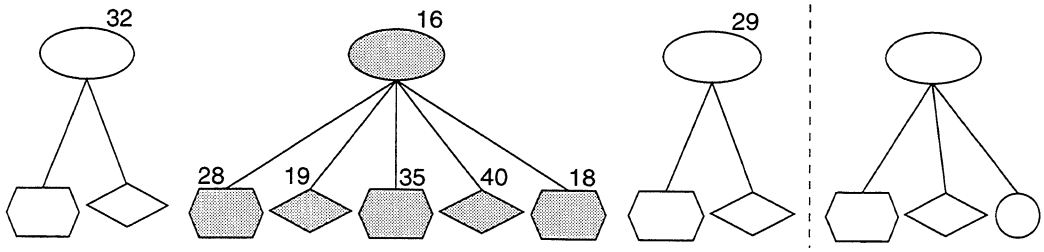

give this result:



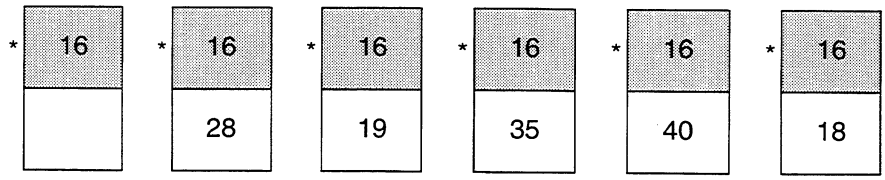
## Example 11

In this example, an INSERT AFTER yields the same result as INSERT BEFORE, because a search through the sibling chain finds no ◯ sibling. It is therefore inserted as the first child.

Given these data trees and this schema tree




and any one of these stacks,

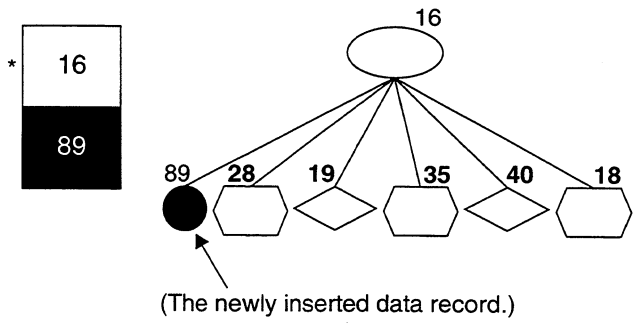



the commands

**INSERT ◯ AFTER** or **INSERT ◯ BEFORE**

**CLEAR**

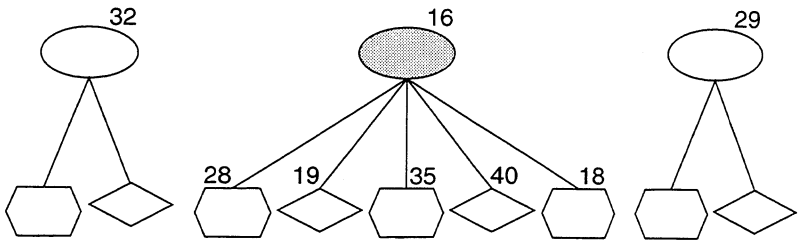
give this result:



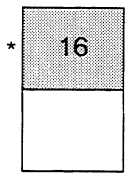
**Example 12**

In this example, no search is performed. The insert occurs after the current stack entry at level 0.

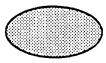
Given these data trees




and any stack with this primary entry,

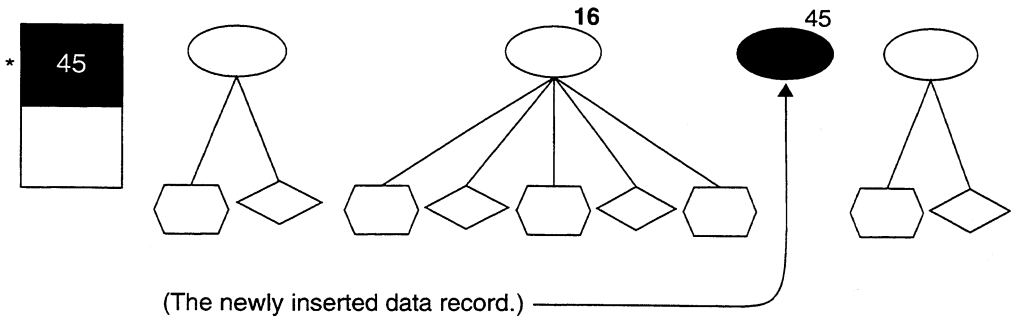



the commands

**INSERT**  **AFTER**

**CLEAR**

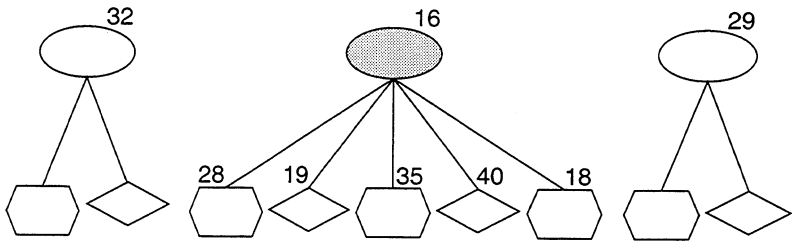
give this result:



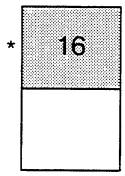
### Example 13

No search occurs in this example either. The insert occurs before the current stack entry at level 0.

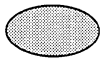
Given these data trees




and any stack with this primary entry,

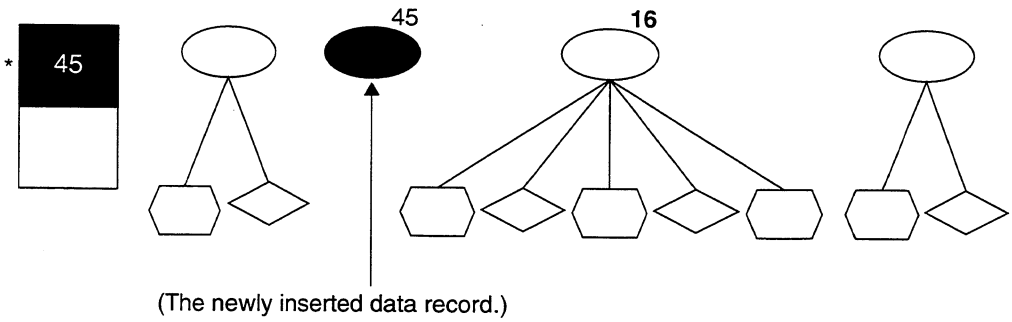



the commands

**INSERT**  **BEFORE**

**CLEAR**

give this result:



# INSERT IN QUEUE MODE

This section discusses the INSERT command in queue mode, including the following topics:

- a general discussion of queue mode INSERT processing
- special considerations for queue mode
- a sample command sequence
- illustrated examples of insert sequences and the placement of insertions as determined by command syntax and stack entries.

For general information on the INSERT command, including command syntax, see **INSERT: General Information for All Modes** on page 10-28.

A program enters queue mode by giving the QUEUE command in immediate mode. In queue mode, a program can repeatedly issue a series of INSERT commands to add trees of data records to certain places in the database. This series of commands is called an insert sequence. Each insert sequence represents one new tree of data records.

The first INSERT command in an insert sequence is given for the topmost data record in the new tree. This data record can be above the level of the primary stack entry. If the topmost data record to be inserted is below level 0, a retrieval command must be given before the insert sequence begins, to establish a stack entry (primary or secondary) for the parent of the topmost data record. If, however, the topmost data record is to be at level 0, no stack entry need be established before the insert sequence begins.

In queue mode, the insertions of new data records are batched, but retrieval commands are processed as soon as the program gives them. Only the TERMINATE command causes the actual insertion of new data records in the database. Thus, new data records referred to by INSERT commands issued in queue mode cannot be retrieved before the TERMINATE command is given.

As in immediate mode, an insert sequence in queue mode consists of INSERT commands, perhaps with other commands interspersed, for the insertion of a tree of data records. Several insert sequences can be given in queue mode before giving the TERMINATE command.

SYSTEM 2000 software detects the end of each insert sequence when it finds any of the following situations:

- an INSERT command for the stack used in the insert sequence, but referring to a data record that is not part of the same tree. The data record referred to could be either on the same level of the database as the topmost data record of the inserted tree or on a higher level (an ancestor)
- an INSERT command referring to the same stack, but to a subschema record for a different database
- an INSERT command referring to another stack
- any update command, other than INSERT or MOVE TREE, referring to any stack

- one of these commands, if it refers to the same stack: GET1, LOCATE, GET, GETD, GETA, GETP, REMOVE, MODIFY, REMOVE TREE, or ORDER BY

- one of these commands, if it refers to the same database: CLOSE, OPEN, KEEP, APPLY, SUSPEND, SAVE, RELEASE, or VALUES PADDING

- one of these commands: CANCEL QUEUE, COPYL, COPYS, CLEAR (all forms), ENABLE/DISABLE ROLLBACK, FRAME/END FRAME, LOAD, QUEUE, START S2K, STOP S2K, or TERMINATE.

As this list shows, an insert sequence is not ended by a GET1, LOCATE, GET, GETA, GETD, or GETP command referring to another stack for the same or a different database. Such a retrieval command is executed at the time it is issued. Therefore, a program can retrieve data during an insert sequence from the same or a different database. It can also use that retrieved data to create new data records to be inserted in the same sequence. Note that a retrieval command that does not end an insert sequence can never retrieve the data being inserted, even though you might think the insert took place.

Also, as the list shows, the FOR, LINK, and FULL PASSES commands do not end insert sequences.

If a program inserts several new trees in succession at the same place in the database (without intervening retrievals that use the same stack), the parent entry in the stack must be established to insert the first of the trees. Thereafter, as each succeeding insert sequence is ended and placed in the batch of queue mode updates, the stack automatically is cleared at levels deeper than the tree top. When the TERMINATE command is executed, this and all other stacks for the database are cleared.

In queue mode, each insert sequence is batched with any other updates in the QUEUE/TERMINATE session. When the TERMINATE command is given, all updates are processed. At TERMINATE time, SYSTEM 2000 software changes the hierarchical structure, first for any REMOVE TREE commands that were given in the QUEUE/TERMINATE session and then for the batched insertions indicated by the queue mode insert sequences, one sequence at a time. An insert sequence is discarded if its parent node was removed by prior REMOVE TREE processing. After the Hierarchical Table is updated, the Data Table and the Index are changed to reflect the new data values for the structure. The database cycle number is increased by one whenever the TERMINATE command is issued, regardless of how many insertions or other updates are processed.

The placement of a newly inserted topmost data record is specified with respect to all its other siblings under its new parent with INSERT BEFORE or INSERT AFTER. AFTER is the default. At the start of a queue mode insert sequence that begins below level 0, the stack must contain an entry for the parent of the topmost data record to be inserted in the sequence. The new data record will be inserted in logical order BEFORE or AFTER all that parent's children, whichever the command specifies.

For an insert sequence that begins at level 0, however, no prior stack entry is necessary. In this situation, INSERT BEFORE means to insert the level 0 data record before all others in logical order; INSERT AFTER means to add the level 0 data record after the last logical entry in logical order.

BEFORE or AFTER is ignored if it is specified in an INSERT command for a descendant of the top node of the new tree. Each descendant is inserted in hierarchical order according to the sequence of SSRs in the INSERT commands. However, if an INSERT command specifies an SSR at the level of the most recently inserted top node, BEFORE or AFTER determines placement as described above.

## Example of Usage

```
WORKING-STORAGE SECTION.
      :
      :
COMMBLOCK OF EMPLOYEE.
      :
      :
SSR ENTRY-REC OF EMPLOYEE.
01  ENTRY-REC.                          (corresponds to the level 0 schema record)
    03  EMPLOYEENUM . . .
        EQ EMPLOYEE NUMBER.

SSR SALARY OF EMPLOYEE.                  (corresponds to the SALARY WITHIN
                                         POSITION schema record)

      :
      :
SSR POSITION-REC OF EMPLOYEE.            (corresponds to the POSITION
                                         WITHIN COMPANY schema record)
      :
      :

01  EMPNO . . .                         (is the variable that holds the required
                                         employee number)
      :
      :
PROCEDURE DIVISION.
      :
      :
   QUEUE.                                (begins the QUEUE/TERMINATE session)

   (Begin loop)

   (Input value for EMPNO)

   GET1 ENTRY-REC USING 0 WHERE          (establishes a stack entry for an ENTRY-REC
   EMPLOYEENUM EQ EMPNO.                 data record -- the parent of the top node
                                         of the subtree, that is, the POSITION-REC
                                         data record, to be inserted; return code
                                         checking must allow for non-existence of
                                         the employee number)

   (Input new values into SSR POSITION-REC)

   INSERT POSITION-REC BEFORE.           (begins a new insert sequence; adds a new
                                         POSITION-REC data record before all
                                         children of the ENTRY-REC data record)
```

```
(Input new values into SSR SALARY)

INSERT SALARY.                    (adds a new SALARY data record under
                                  the POSITION-REC data record;
                                  specifying BEFORE or AFTER makes
                                  no difference here, since the data
                                  record is below the top of tree)

(Check for condition to skip out of loop;
repeat loop if condition not true)

TERMINATE.                        (ends the QUEUE/TERMINATE session and
                                  performs updates in the Hierarchical
                                  Table, the Data Table, and the
                                  Index tables)
```

## Considerations

1. The considerations listed in **INSERT: General Information for All Modes** on page 10-28 pertain to INSERT in queue mode.

2. In queue mode, update commands are not executed as they are encountered, but rather are grouped together. All REMOVE TREE commands form one group; all INSERT commands form another group; MODIFY and REMOVE commands form a third group. When a TERMINATE command is encountered, all update commands are executed in a batch -- first the REMOVE TREE commands, then the INSERT commands, then the MODIFY and REMOVE commands. Within each group, commands are executed in the order in which they were given.

3. A program can acquire a global hold on the database prior to an insert sequence in queue mode by giving a retrieval command with the HOLD option before the first INSERT command in the sequence. The program maintains the hold throughout the QUEUE/TERMINATE session, regardless of how many insert sequences are given in that session, until it either

   a) gives a retrieval command that does not specify HOLD

   b) gives the TERMINATE command.

   For a discussion of global holds, see **HOLD Option** on page 8-131.

4. In queue mode, an INSERT command must refer to the database named in the first update command of that QUEUE/TERMINATE session. If the INSERT command refers to a different database, it is ignored and return code 28 results.

5. Considerations for queue mode are given in **QUEUE and TERMINATE** on page 9-2.

6. Before an INSERT command is issued to begin an insert sequence that adds a data record below level 0, the stack must contain an entry for the parent of that data record. If it does not, the command is ignored and return code 03 results.

7. If the stack entry for the parent to be used by the first INSERT command in an insert sequence was canceled by a previous queue REMOVE TREE or MOVE TREE operation, return code 09 results. If the stack entry for the parent was the primary entry, it remains in the stack. And if the primary stack entry for the parent was canceled by a previous queue REMOVE TREE or MOVE TREE operation in another stack in the program or by another user in a Multi-User environment, the INSERT command is ignored, and return code 09 results. Also, the eighth field in the COMMBLOCK contains the database level of the removed parent entry.

8. The space in the Hierarchical and Data Tables that is freed by a REMOVE TREE command is not immediately reusable if more than one user has the database open. Another PLEX program might have a stack or Locate File containing the address of the removed record. It would not be appropriate to insert new, unrelated data into the record and give the record a different structural position.

   If no other user has a certain database open, space from removed records will be reused. The presence of this reused address in a Locate File or stack could produce the following situation. With a REMOVE TREE command, the program using that database causes the deletion of a data record whose address is in the Locate File. While the address for the deleted data record is still in the Locate File, an INSERT command given by the program could cause this address to be reused for a newly inserted record. The presence of this reused address in the Locate File could cause problems for GET and ORDER BY commands using that Locate File. To avoid these problems, the program should coordinate its retrievals and deletions of records whose addresses are in Locate Files.

   If a program cannot predict and control record removal and reassignment for a particular schema record, contact your SYSTEM 2000 software representative for a special zap to prevent reuse of space until all users have closed a database. (Contact your SYSTEM 2000 Software Representative.) The space is reused when the database is opened again.

9. During an insert sequence, the stack entries below the top level retain the tree structure for a check for a valid tree. When the program changes stacks, the tree structure stack entries are preserved as regular stack data, that is, for subscript zero, 32 levels are retained and for each non-zero subscript ten levels are retained. At those ten levels, the top level is the highest non-zero entry in the stack, whether it be a position entry or a tree structure entry. If the combination of position(s) plus INSERT-sequence tree structure entries spans more than ten levels, the INSERT sequence must be associated with subscript (0) to avoid rejection of lower branches in the INSERT tree after a change in subscript. If the program is inserting without intervening change(s) in subscript, all 32 levels are available for retaining data for tree structure checking.

The following illustrations show different situations of updating with INSERT in queue mode. They are summarized here.

**Example 1:** Two consecutive insert sequences.

**Example 2:** Three insert sequences, one of which ends by a retrieval with the same stack.

**Example 3:** Subtrees inserted in separate logical entries.

**Example 4:** Dissimilar siblings in one insert sequence; a null item in a newly inserted record.

**Example 5:** Logical entries added with INSERT BEFORE.

**Example 6:** Logical entries added with INSERT AFTER.

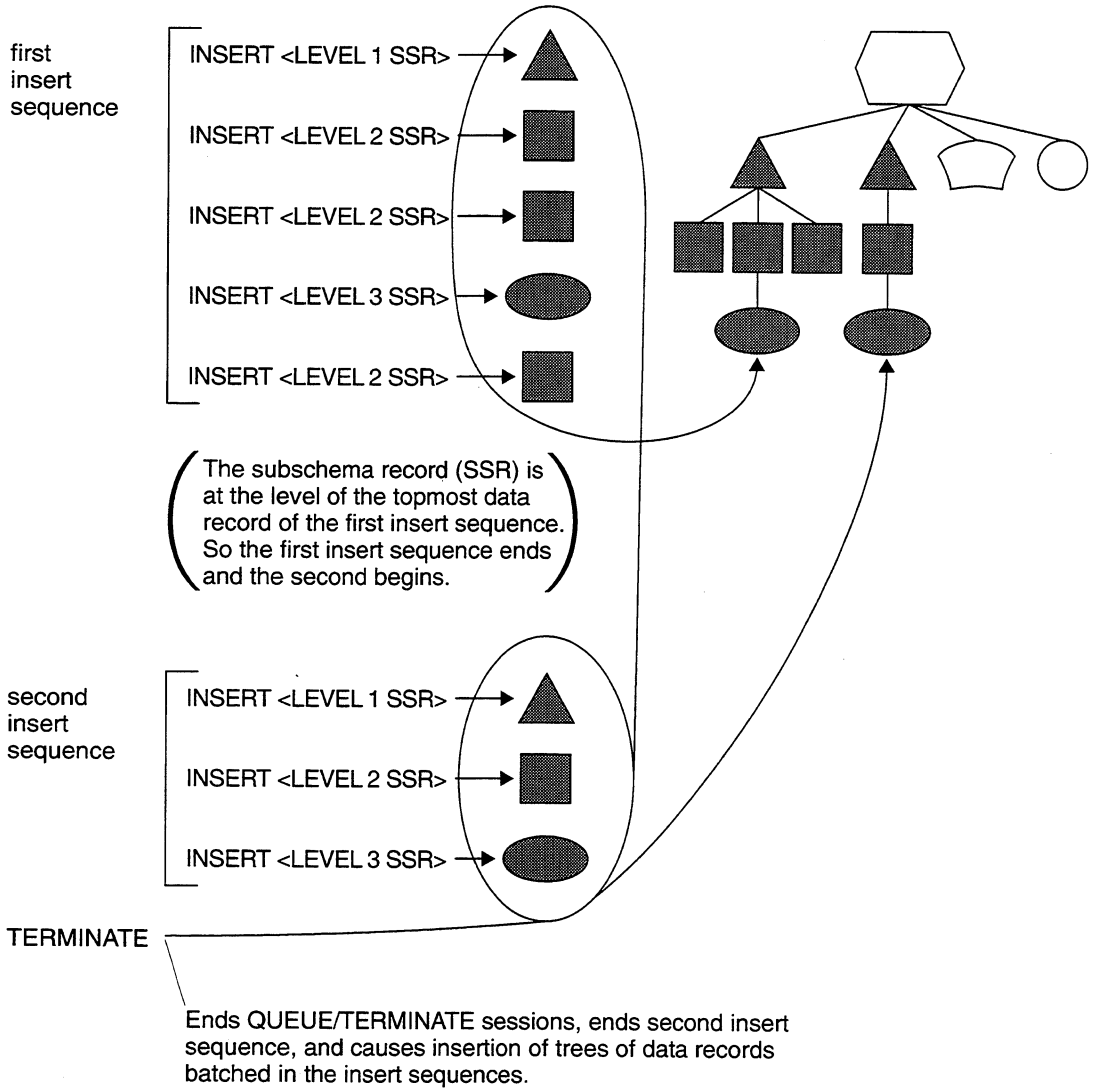
**Example 7:** INSERT with a name table.

**Example 1**

The first insert sequence is ended by the beginning of the second. TERMINATE ends the second insert sequence and causes the trees to be inserted.

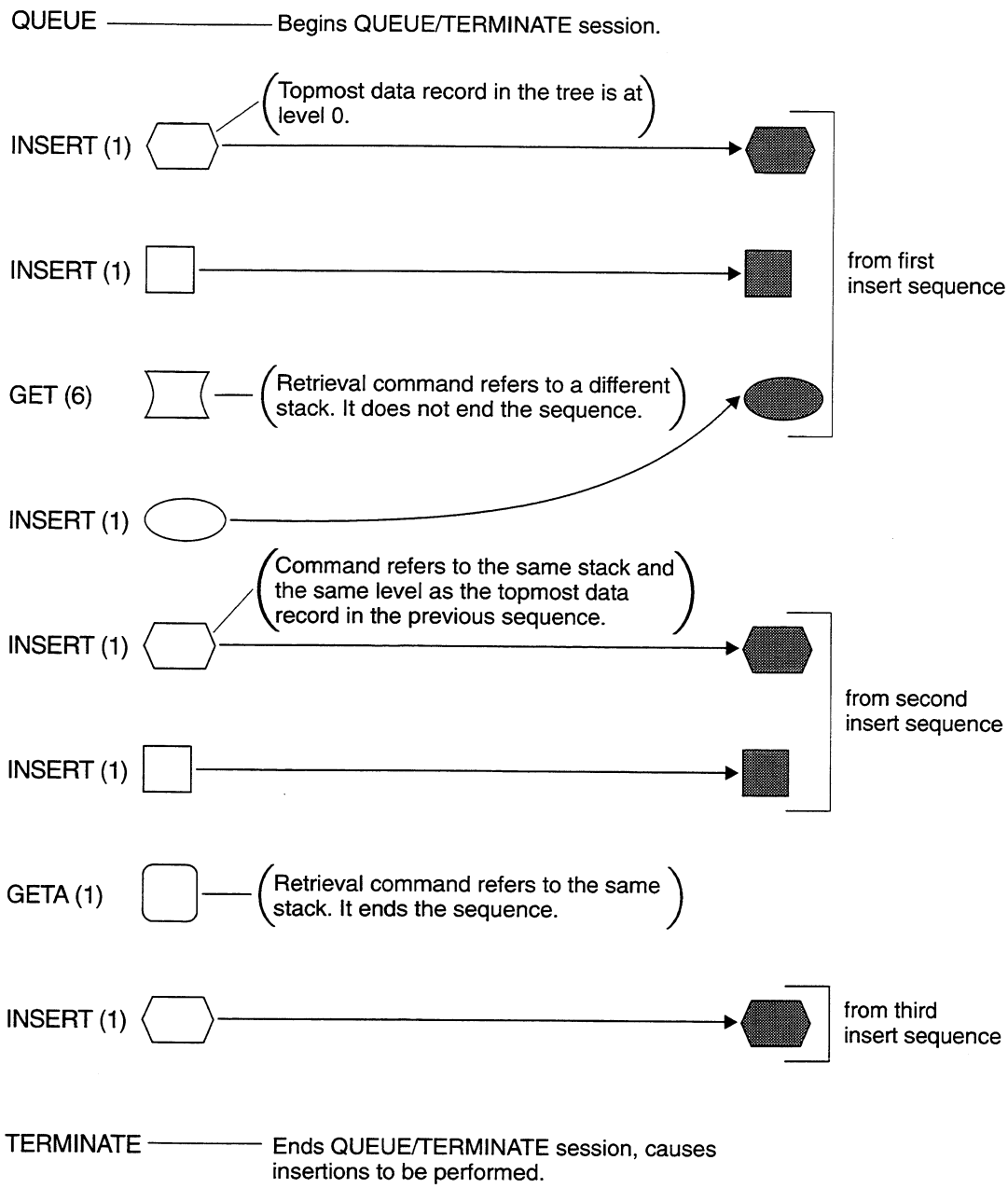
QUEUE ——— Begins QUEUE/TERMINATE session.

GET1 (1) <LEVEL 0 SSR>



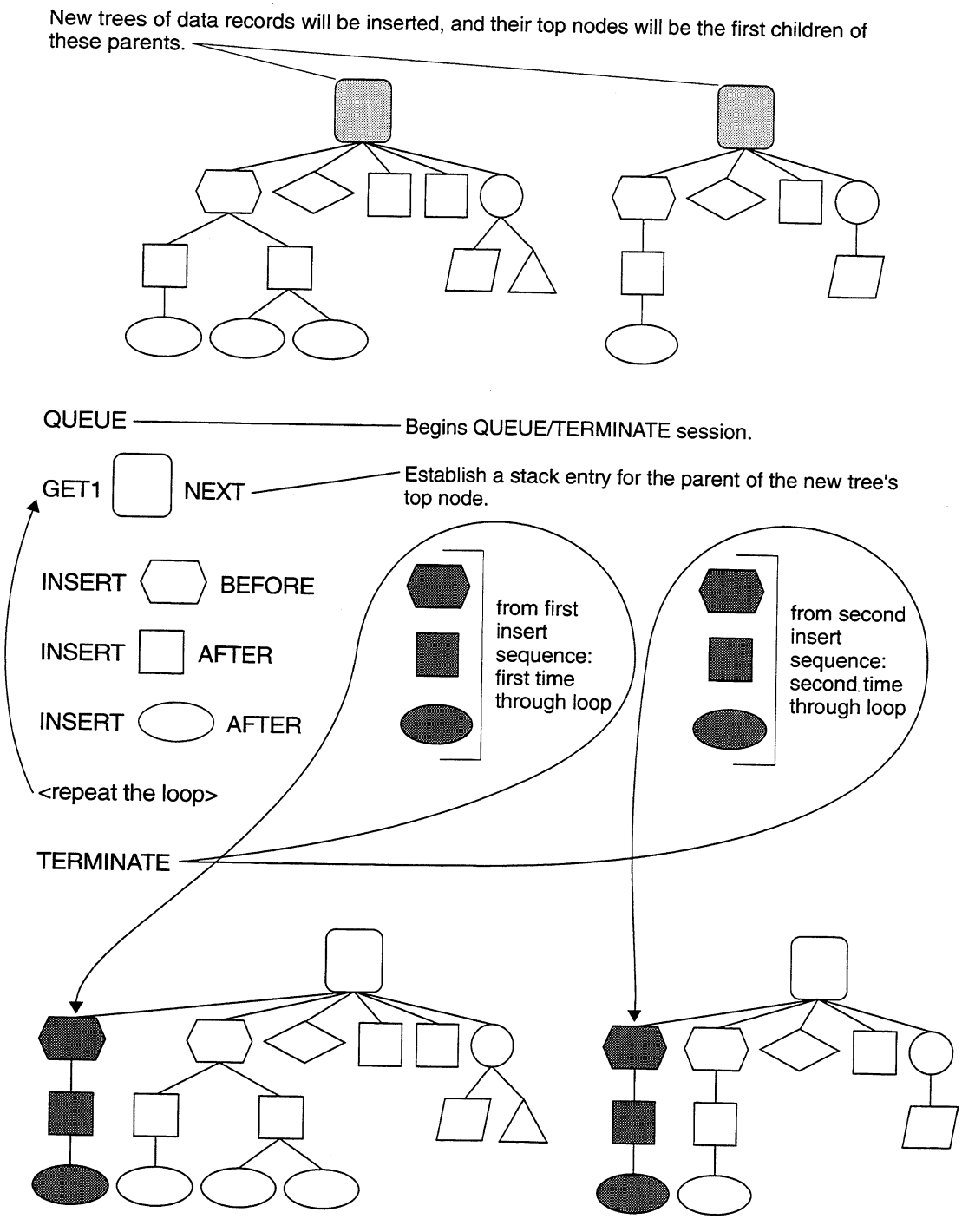
### Example 2

Three insert sequences are shown here. A retrieval command ends an insert sequence only if it refers to the same stack.



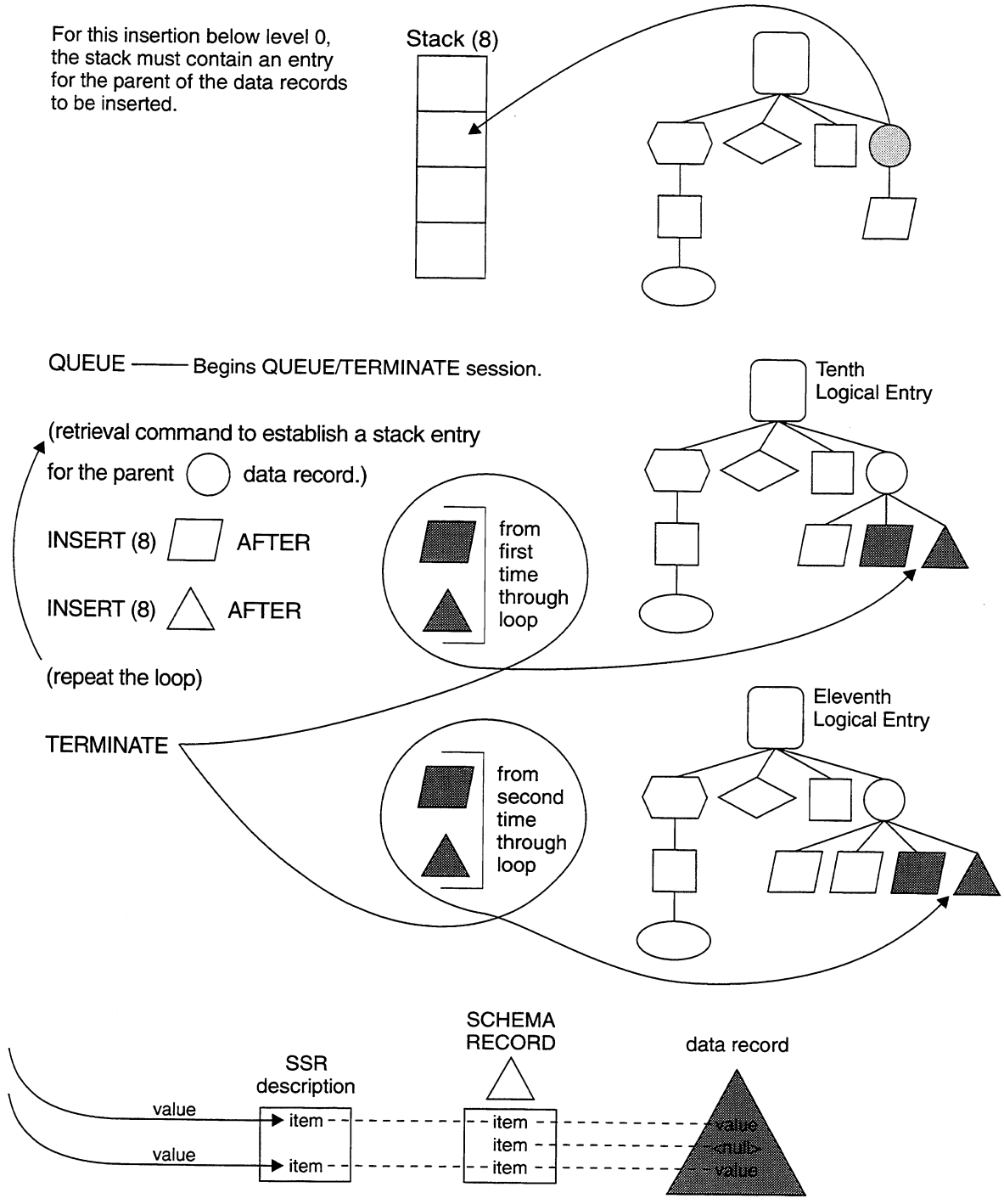
**Example 3**

One subtree is inserted in each logical entry. A GET1 NEXT command establishes a primary stack entry for the parent of the top node of each new subtree.



## Example 4

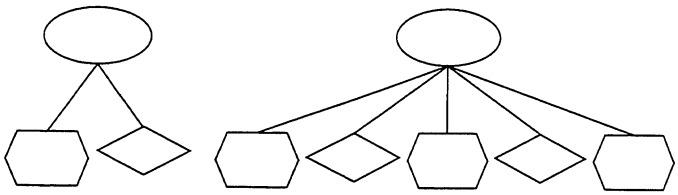
Dissimilar siblings are inserted in each logical entry. Because the SSR for △ records contains two items, rather than three as in the corresponding schema record, one item is null in the newly inserted △ records.



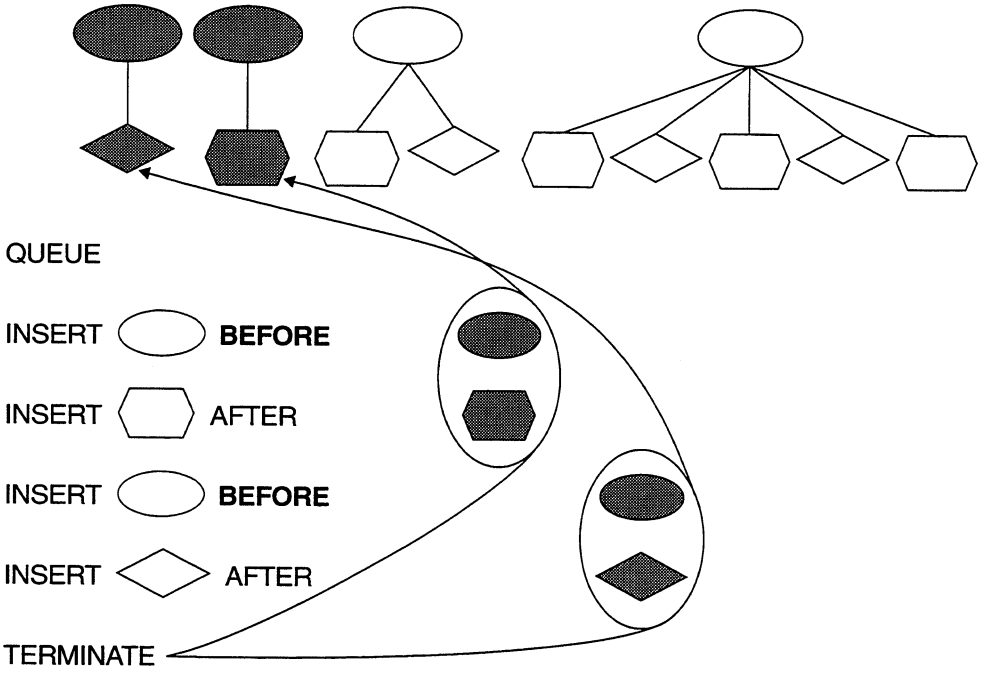
**Example 5**

INSERT BEFORE for a level 0 record specifies that the newly inserted logical entry is placed before all others in the data structure.

Given these data entries,



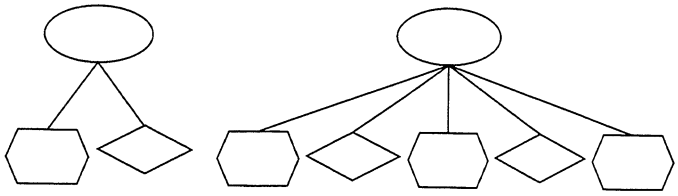
and any stack, regardless of its current entries, this QUEUE/TERMINATE session gives the following result:



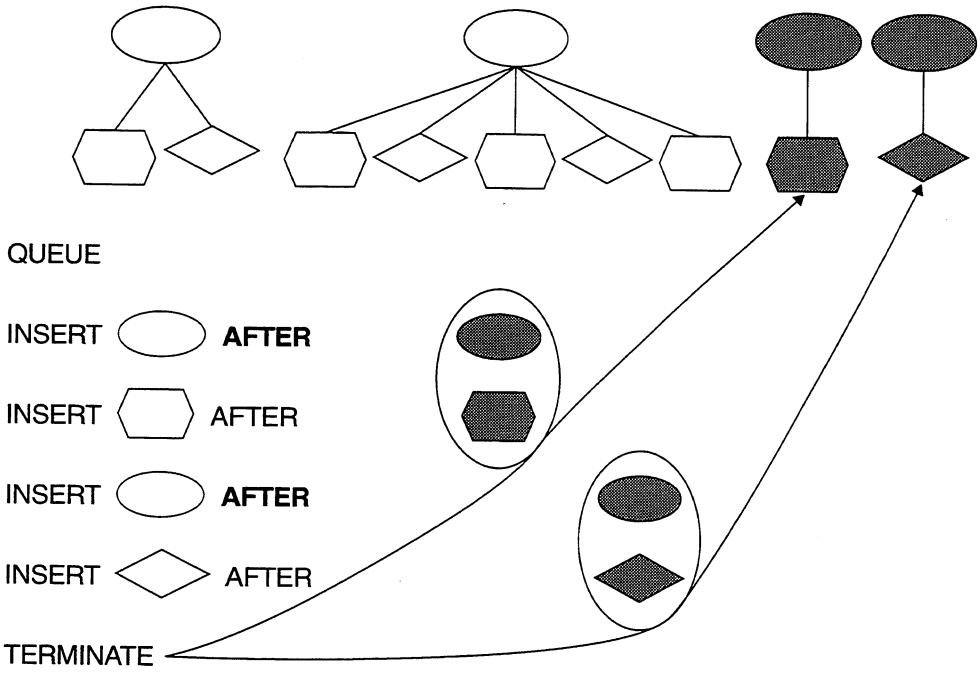
**Example 6**

INSERT AFTER for a level 0 record specifies that the newly inserted logical entry is placed after all others in the data structure.

Given these data entries,



and any stack, regardless of its current entries, this QUEUE/TERMINATE session gives the following result:

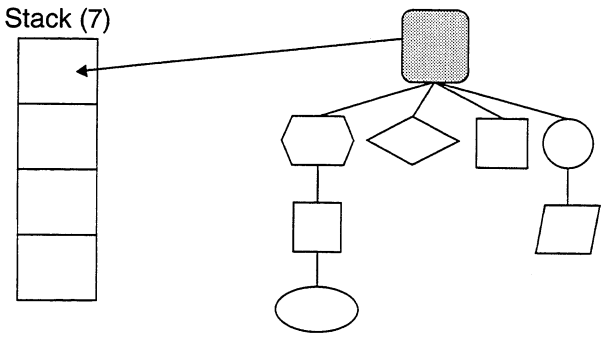


**Example 7**

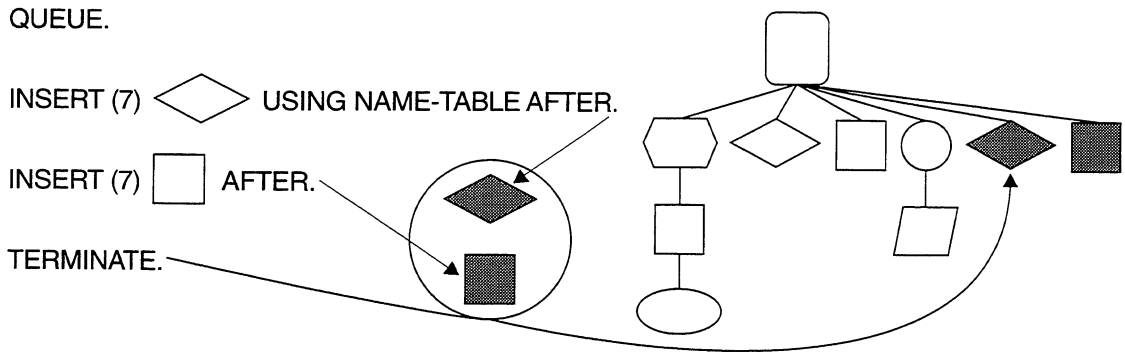
Because the name table used in an INSERT specifies only two of the three items in the schema record, the ◇ record is inserted with one null item.

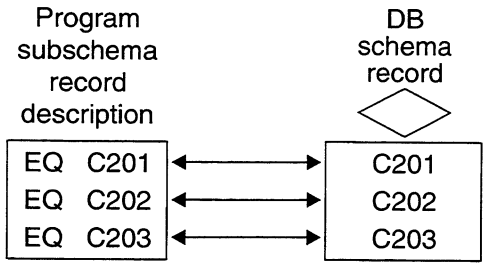
```
WORKING-STORAGE SECTION.
      .
      .
      .
01  NAME-TABLE.
    05  NAME-ITEM OCCURS 3 TIMES PIC X(32).
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
    MOVE 'C201' TO NAME-ITEM (1).
    MOVE 'C203' TO NAME-ITEM (2).
    MOVE SPACES TO NAME-ITEM (3).
      .
      .
      .
```

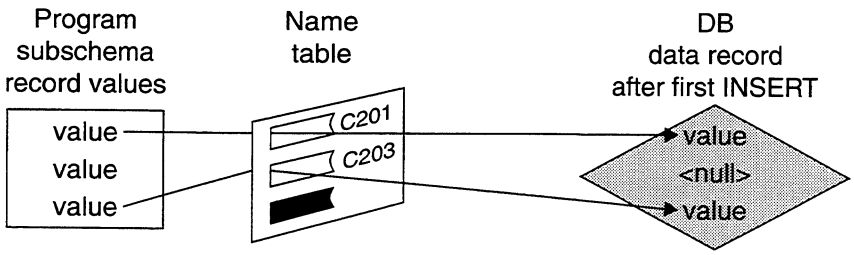
The stack entry has an entry for the parent of the data records to be inserted.



**Example 7 (continued)**



# INSERT IN LOAD MODE

This section discusses the INSERT command in load mode, including the following topics:

- a general discussion of immediate mode INSERT processing
- special considerations for load mode
- a sample command sequence
- illustrated examples of insert sequences and the placement of insertions as determined by command syntax and stack entries.

For general information on the INSERT command, including command syntax, see **INSERT: General Information for All Modes** on page 10-28.

Normally you use load mode to add large numbers of data trees to an existing database or to populate a new database. You can also use load mode to insert a small number of data trees when you want to minimize processing time or to avoid using reusable space, that is, to force the new data records to be stored in consecutive table addresses. A program enters load mode by giving the LOAD command in immediate mode. The program leaves load mode when it issues a TERMINATE, CLOSE, or CANCEL QUEUE command.

While in load mode, the program issues a series of INSERT commands to add trees of data records. The first INSERT command given in the LOAD/TERMINATE session is for a data record at the highest level at which insertions are to be made during the session. If that highest level is level 0, no stack entry needs to be established for reference in the first insertion. But if the highest insertion is to be made below level 0, the program must establish a stack entry for the parent of the topmost data records to be inserted. This entry must be established with a retrieval command before the LOAD/TERMINATE session begins.

The first INSERT command executed in a LOAD/TERMINATE session cancels the contents of all stacks and Locate Files, except for the stack most recently referred to before the LOAD command was executed. This stack remains unchanged throughout the LOAD/TERMINATE session, even after the TERMINATE command is given. A Locate File with the same number as this stack is also preserved.

The INSERT commands must follow logical order. That is, each data record to be inserted in one tree must be hierarchically related to the data record previously inserted. Among data records, the hierarchical relationships are determined by the schema tree. After the program issues an INSERT command for the top data record of a new tree, INSERT commands for descendant data records could be given according to the hierarchical order of schema records to which the data records correspond.

Each valid INSERT command issued in load mode is processed in two steps. An addition is made both to the Hierarchical Table and to the Data Table at the time the command is given, using the available space at the end of those tables. Corresponding changes for the Index, for any key data values being inserted, are batched until the program gives the TERMINATE command to exit from load mode. At that point, the updates for the Index are processed. The database cycle number is then incremented by one.

With INSERT BEFORE or INSERT AFTER (default), the program specifies the placement of a data record in logical order. If the first valid insert is at level 0, it will specify placement of new entries before or after all existing entries. If the first valid insert is not at level 0, the

new topmost data records will be placed before or after all of the parent's children. If a subsequent data record to be inserted is at the same level as the data record in the first valid insertion in the load mode session, INSERT BEFORE means to insert the new data record in logical order before that first and subsequent data records, that is, each new top-level before the preceding top-level. Similarly, in this situation, INSERT AFTER means to add the new data record after the data record that is currently last in logical order.

If, on the other hand, the data record to be inserted is below the level of the first valid insertion in the load mode session, it is inserted after its siblings in logical order, regardless of whether the INSERT command specifies BEFORE or AFTER. The resulting logical order of lower level data records will correspond to the order in which they were inserted.

## Example of Usage

```
WORKING-STORAGE SECTION.
      :
COMMBLOCK OF EMPLOYEE.
      :
SSR  ENTRY-REC OF EMPLOYEE.        (corresponds to the level 0 schema record)
      :
SSR  POSITION-REC OF EMPLOYEE.     (corresponds to the POSITION WITHIN
                                   COMPANY schema record)
      :
SSR  SALARY OF EMPLOYEE.           (corresponds to the SALARY WITHIN
                                   POSITION schema record)

PROCEDURE DIVISION.
      :
   LOAD EMPLOYEE.                  (begins the LOAD/TERMINATE session)

   (Begin the loop)

   (Input new values into SSR ENTRY-REC)

   INSERT ENTRY-REC AFTER.         (adds the top node of a new logical entry
                                   after all other logical entries in the
                                   logical structure)

   (Input new values into SSR POSITION-REC)

   INSERT POSITION-REC.            (adds a POSITION-REC data record as a
                                   child under the ENTRY-REC data record;
                                   AFTER is the default)

   (Input new values into SSR SALARY)

   INSERT SALARY.                  (adds a SALARY data record as a child
                                   under the POSITION-REC data record;
                                   AFTER is the default)

   (Check for condition to skip out of loop)
```

```
TERMINATE.                    (ends the LOAD/TERMINATE session and
                              updates the Index)
```

## Considerations

1. The considerations listed in **INSERT: General Information for All Modes** on page 10-28 pertain to INSERT in load mode.

2. Considerations for load mode are given in **Considerations** on page 9-3. See Chapter 21, "Populating a Database," for a more detailed discussion of populating a database in load mode.

3. A program must open the database for exclusive use in order to operate in load mode on that database. If not, the LOAD command is ignored and return code 91 results.

4. If the database referred to in the LOAD command is damaged, the first INSERT command given in load mode is ignored and return code 16 results. All other commands in load mode, except TERMINATE, are also ignored. When a program opens a database for exclusive use as preparation for load mode, it should check the damage flag to avoid this error during a LOAD/TERMINATE session.

5. The first valid INSERT command in a LOAD/TERMINATE session clears all stacks and Locate Files except the stack (and the Locate File, if there is one, with the same number) last referred to before the LOAD command was issued. Each INSERT command must then refer to the same database specified in the LOAD command. If an INSERT command refers to a different stack or database, the command is ignored and return code 53 is placed in the COMMBLOCK associated with the INSERT command.

6. The database level of the first valid insertion in a LOAD/TERMINATE session is the highest level at which insertions can be made in that session. If an INSERT command is given for a data record above that level, the command is ignored and return code 54 results.

7. The AFTER or BEFORE specification in the first INSERT command that is accepted in a LOAD/TERMINATE session determines the placement of all other data records to be inserted at the level of the first insertion. If a subsequent INSERT command given for a data record at that top level specifies BEFORE or AFTER contrary to the first insertion, the command is processed using BEFORE or AFTER as in the first INSERT, but return code 50 indicates that the processing ignored the change in BEFORE/AFTER.

8. In a LOAD/TERMINATE session, if the program gives a command other than INSERT, TERMINATE, CANCEL QUEUE, or CLOSE, that command is ignored and return code 55 results. Each command given thereafter in the LOAD/TERMINATE session, including each INSERT command, is also ignored, resulting in the same return code 55. The load mode insertions made prior to the first invalid command are canceled.

9. If the first INSERT command in the LOAD/TERMINATE session is for a data record below level 0, a stack entry must be established for the parent of the new data record before giving the LOAD command. The first INSERT command after LOAD must then refer to the stack where the entry was established. If, however, the stack has no entry or a removed entry for the parent of the first insertion, each INSERT command, starting with the first one, is ignored and return code 03 results.

10. The stack entry to be used by the first INSERT command in a LOAD/TERMINATE session might have been canceled before the program acquired exclusive use of the database in preparation for load mode. The entry might have been canceled by another stack in the program or by another user in a Multi-User environment. In this situation, the first INSERT command given after the LOAD command is ignored, as is every subsequent command in the LOAD/TERMINATE session and return code 09 results. Also, the eighth field in the COMMBLOCK contains the database level of the removed entry.

The following illustrations show different situations of updating with INSERT in load mode. They are summarized here.

**Example 1:** Inserting a new logical entry.

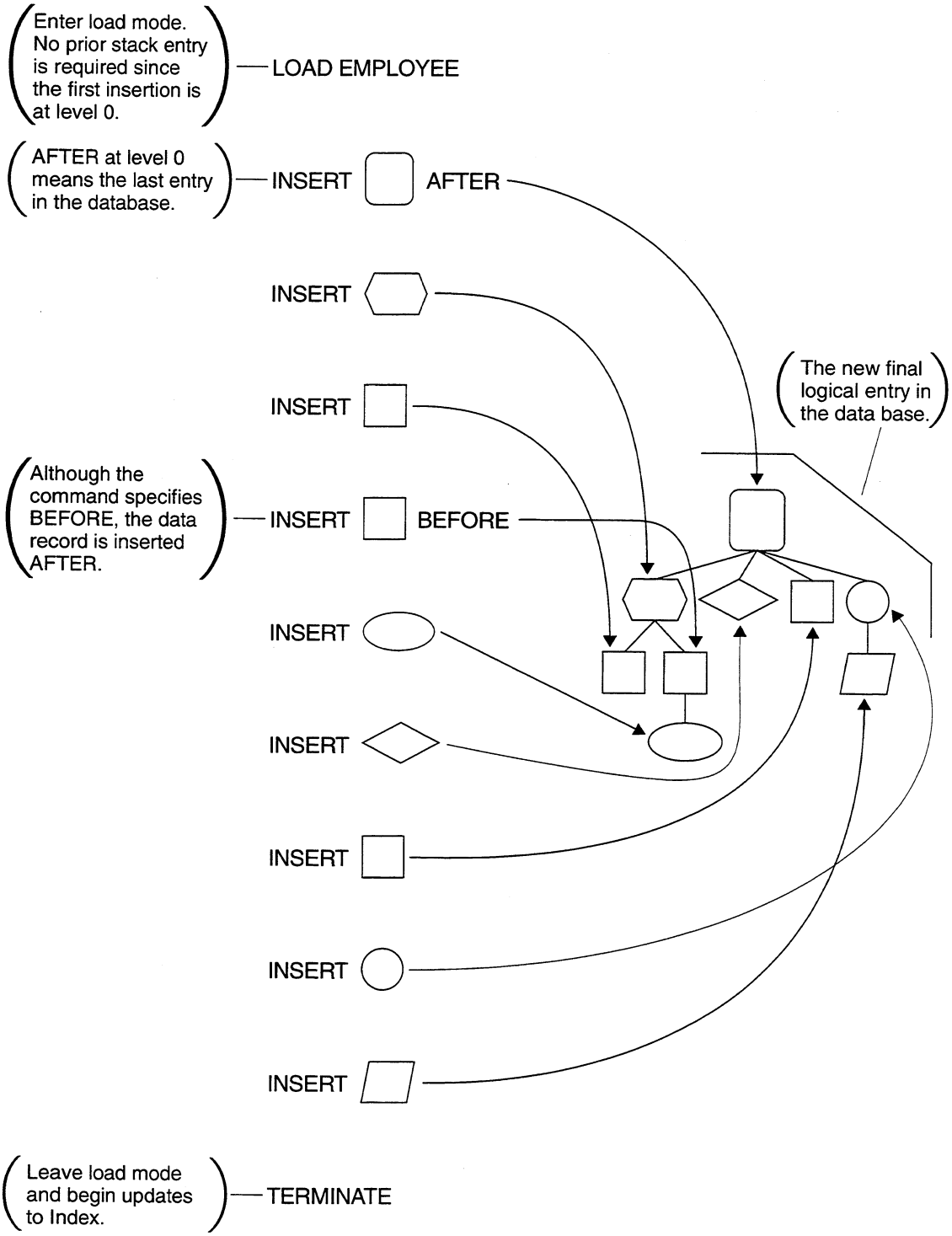
**Example 2:** Inserting two subtrees before all others under a parent.

**Example 3:** Using an incomplete SSR and inserting two subtrees after all others under a parent.

**Example 4:** Using a word table and inserting two subtrees before all others under a parent.

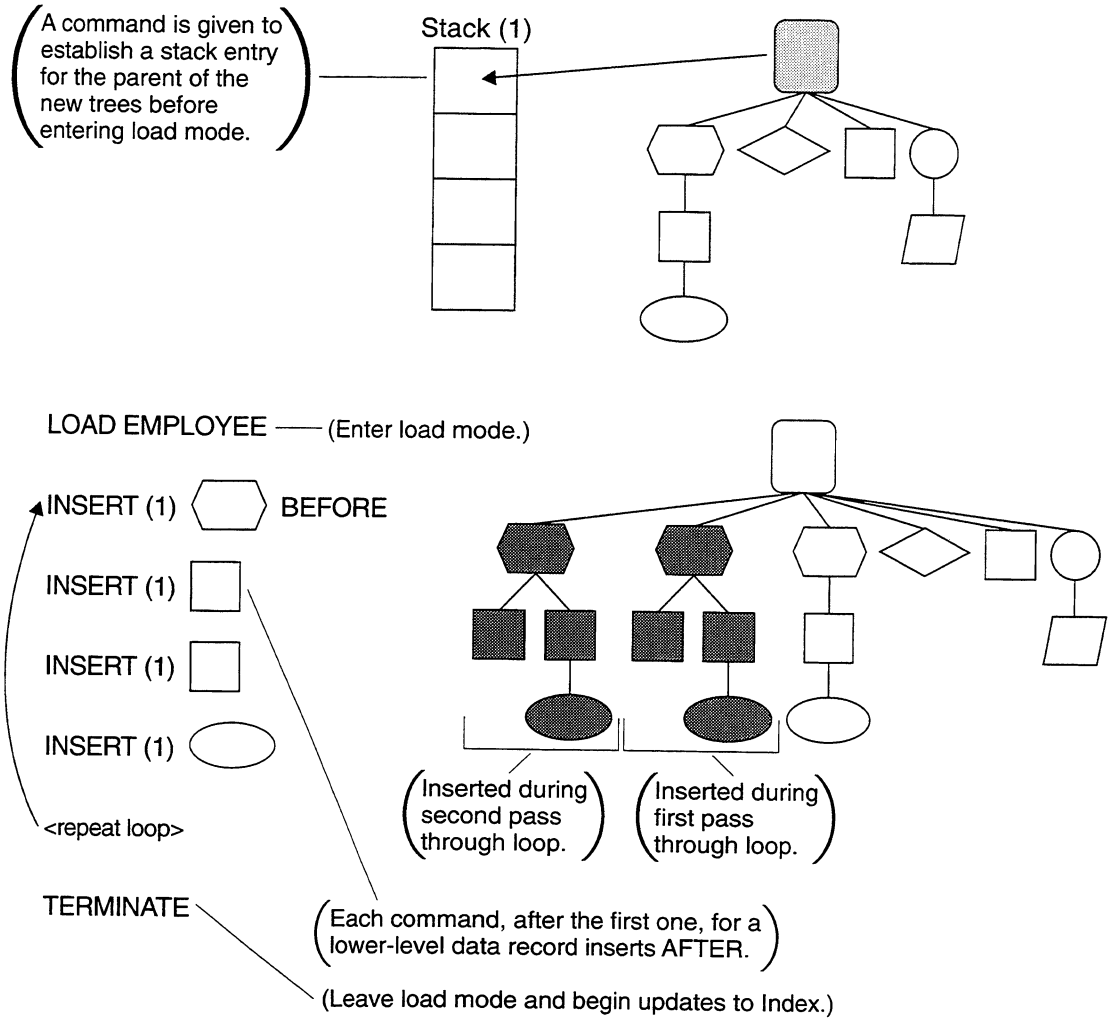
**Example 1**

In this example, a new logical entry is added to the end of the data structure. Specifying BEFORE in an INSERT command for a non-level-0 SSR has no effect.



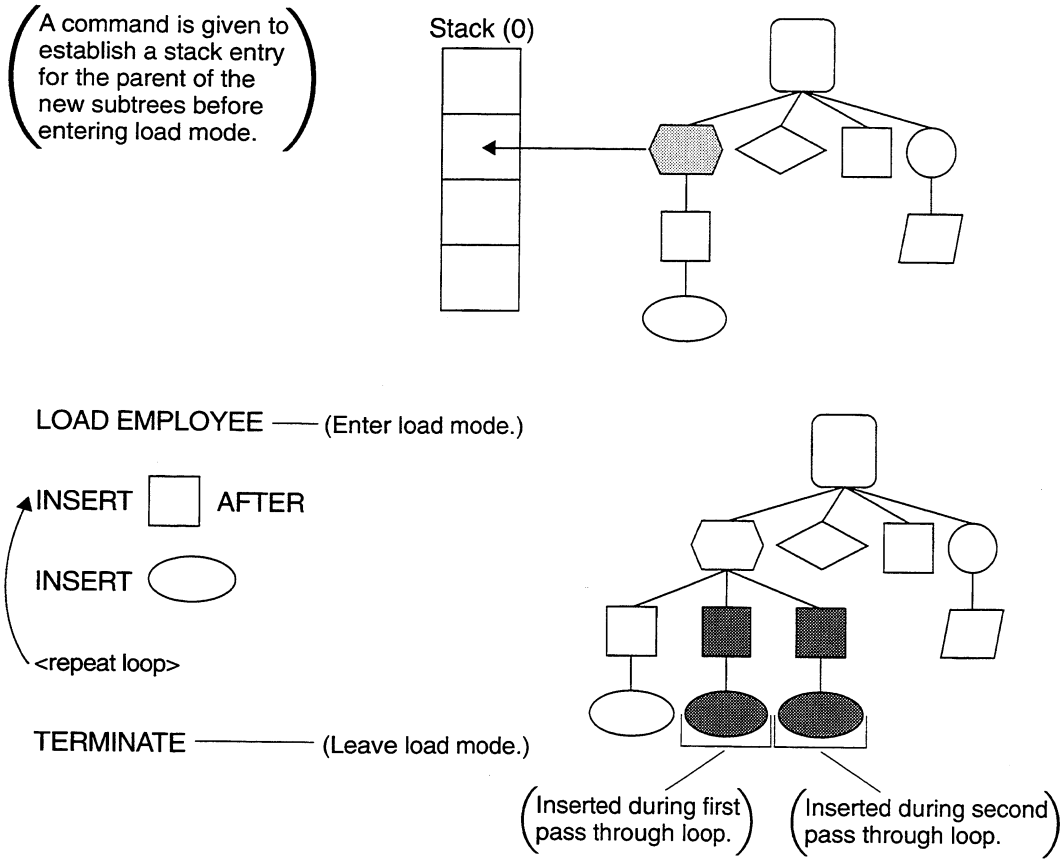
### Example 2

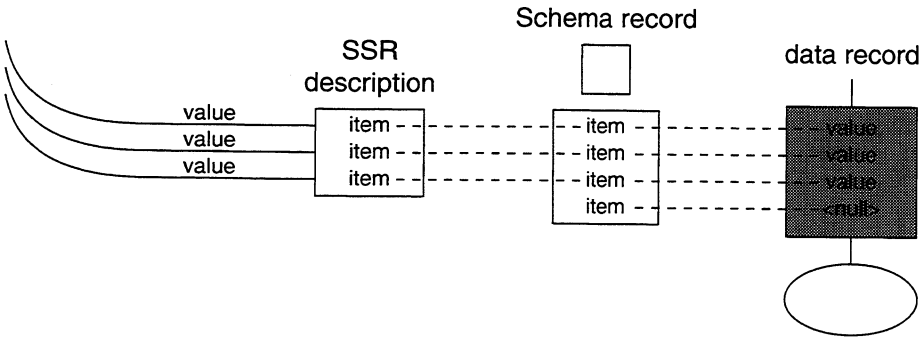
Here, two subtrees are added to the same logical entry. Specifying BEFORE for the top node in a new subtree causes the entire subtree to be inserted before all others under the same parent.



**Example 3**

Here, two subtrees are added to the same logical entry. Specifying AFTER for the top node in a new subtree causes the entire subtree to be inserted after all others under the same parent. Because the specified SSR has items for three of the four corresponding schema items, the □ record is inserted with a null item.



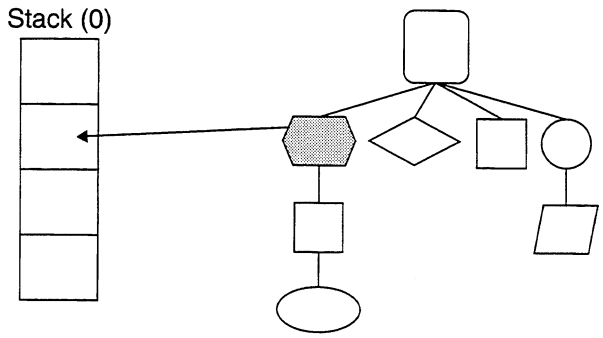



### Example 4

Here, two subtrees are added before all others under the same parent. According to the word table used in the level 3 INSERT command, two items in the ⬭ records are null.

```
WORKING-STORAGE SECTION.
      .
      .
      .
01  WORD-TABLE.
    05   WORD-ITEM OCCURS 6 TIMES PIC S9(9)
    COMP.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
    MOVE 1 TO WORD-ITEM (1).
    MOVE 0 TO WORD-ITEM (2).
    MOVE 0 TO WORD-ITEM (3).
    MOVE 1 TO WORD-ITEM (4).
    MOVE 1 TO WORD-ITEM (5).
    MOVE 1 TO WORD-ITEM (6).
      .
      .
      .
```

A command is given to establish a stack entry for the parent of the new subtrees before entering load mode.



**Example 4** (continued)

```
LOAD EMPLOYEE.                          (enters load mode)

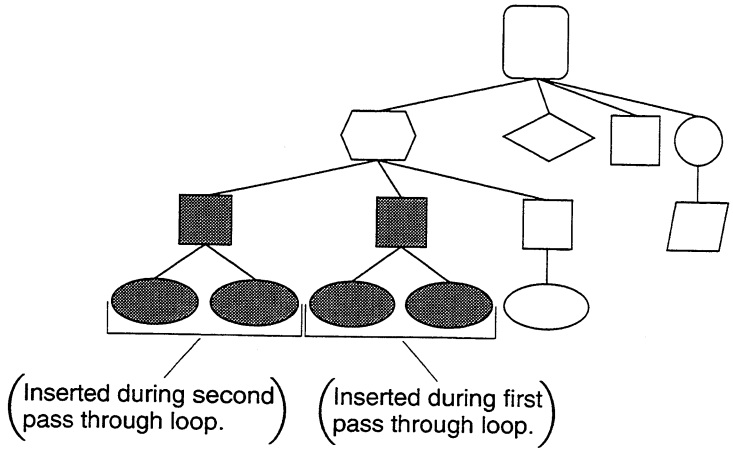
(Start the loop.)

INSERT  □   BEFORE.

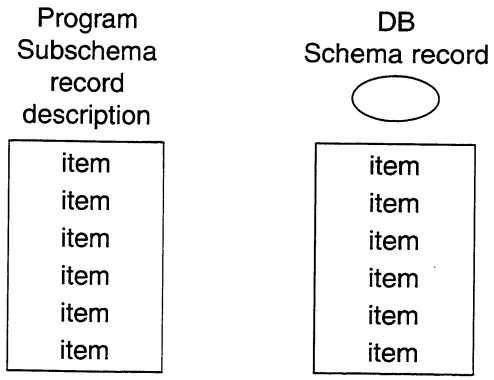
INSERT  ⬭   USING  WORD-TABLE.

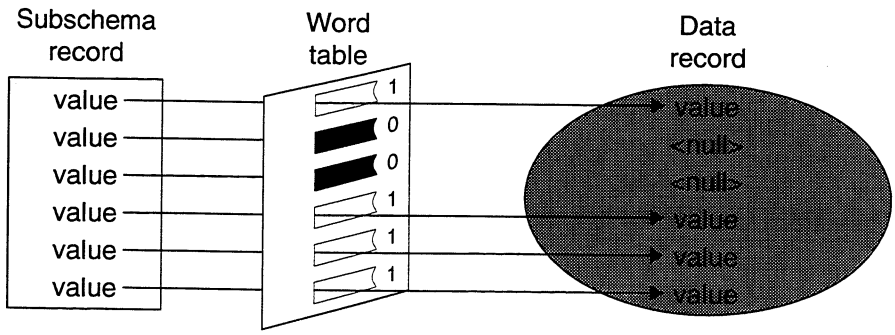
INSERT  ⬭   USING  WORD-TABLE.

(Repeat the loop.)
```




```
TERMINATE.                              (leaves load mode)
```



# MOVE TREE

The MOVE TREE command moves a data record and its descendants from one logical position in the database to another logical position in the same database. The data tree to be moved can be an entire logical entry or any subtree in an entry.

- You can move entire trees or subtrees from one position to another position in the database.
- The software modifies appropriate pointers to accomplish the move without changing the physical position of the data records in the Data Table or the Index.
- MOVE TREE is logically equivalent to REMOVE TREE followed by an insert sequence, but no changes are made to the Data Table.

The MOVE TREE command allows you to move a data tree of records from one logical position to another in a single command without changing the physical structure of the Data Table or the Index. Otherwise, this type of movement is a two-step process that involves a REMOVE TREE command and an INSERT command; this also involves the actual movement of data records and modification of the indexes for key items. The MOVE TREE command is a faster, single operation that involves changing only the appropriate pointers in the Hierarchical Table.

The MOVE TREE capability is a one-to-one relationship. That is, one data tree is logically moved from one position to a new, single point of attachment. The data tree cannot be copied to multiple new positions; only INSERT commands can be used to place a data tree in multiple positions.

Before issuing a MOVE TREE command, your program must establish a stack entry (primary or secondary) for the data record that is the top node of the tree or subtree of data records that is to be moved. This is the "from" stack. Then, another primary or secondary stack entry must be established in a different stack to be used as a position for the place of insertion, that is, the "to" stack. A MOVE TREE command can then be given, specifying the "from" stack and the "to" stack and a subschema record for the data record at the top node.

When it is executed, the MOVE TREE command causes a logical change in the data record pointers, but no physical changes to the Data Table or Index. SYSTEM 2000 software carries out this change by switching the pointers in the Hierarchical Table entries to reflect the removal and the insertion of the top node of the data tree affected.

First, SYSTEM 2000 software checks to see if the MOVE TREE command contains a valid subschema record name. (Because MOVE is a reserved word for COBOL, the keywords MOVE TREE must be followed by a valid subschema record name, or the entire command is passed directly to the COBOL compiler and not treated as a PLEX command.)

Next, SYSTEM 2000 software checks to see if the proper holds are present and if the "from" and "to" stacks are valid according to the subschema record name given in the command. Return code 03 is issued if any run-time errors occur. Return code 00 indicates that the command was successful.

A local hold is required for the "from" data record only if LHOLD equals YES, the database is not under a global hold or exclusive use, and Rollback is disabled. In order to make the hold on the "from" stack effective, establish the "to" stack first; otherwise, the command that establishes the "to" stack will cancel the hold on the established "from" stack. If the multiple holds option is in effect, both the "to" and "from" stacks can be held.

No user hold is required if the database is under exclusive use, if a global hold is in effect, or if Rollback is enabled. If LHOLD equals NO, user holds are optional; you can choose to hold either or both data records, depending upon whether multiple local holds are allowed.

If the MOVE TREE command completes successfully with return code 00, the status of the stacks is as follows:

- The "from" stack is modified to be equivalent to the results of REMOVE TREE command in immediate mode. This means that the "from" stack entries (primary or secondary) below the level of the specified subschema record are canceled. The stack entry for the top node just removed remains in the stack as a position for insertions or as an initial position for retrievals; it remains until canceled by a command that either replaces the top-node entry in the stack or refers to another stack or database.

- The "to" stack is modified by moving the "from" Hierarchical Table entry and all lower entries in the "from" stack to the "to" stack. Then any lower entries still remaining in the "to" stack are removed.

The fifteenth COMMBLOCK variable (the parent pointer) is not set by a MOVE TREE command because it is not a retrieval command.

After completing a MOVE TREE command and before the update status is released for the indicated database, SYSTEM 2000 Multi-User software examines the current state of every PLEX user job, including the user job executing the PLEX MOVE TREE command. The software then checks the stacks for each of these user jobs. Any stack found to contain any data record from the tree that was moved is modified by having all such data record entries removed. Any subsequent reference by such a stack for a moved data record results in return code 03 or 09, similar to that issued for a data record removed by a REMOVE TREE command.

**Note:** If the current user has exclusive use of the database, only his stacks need to be checked and/or modified because no other user can have the database open.

## Format

```
MOVE TREE SSR (from) TO (to) |AFTER
                             |BEFORE
```

*SSR* is the name of the subschema record description for the data record that is the top (or only) node of the tree to be moved.

*from* is the number of the "from" stack; a constant or a variable whose value is greater than or equal to zero and less than or equal to 15, that is, $0 \leq n \leq 15$.

*to* is the number of the "to" stack; a constant or a variable whose value is equal to or greater than zero and equal to or less than 15, that is, $0 \leq n \leq 15$.

**Note**: For COBOL, the *from* and *to* variable can have DISPLAY, COMP-3, or COMP usage, with a picture suitable for a one-digit or two-digit integer.

A MOVE TREE command is, by default, MOVE TREE *SSR* (*from*) TO (*to*) AFTER, if neither BEFORE nor AFTER is specified. The short form of the command and its equivalent are shown below.

```
MOVE TREE SSR (from) TO (to).      (short form, using the default)
MOVE TREE SSR (from) TO (to) AFTER.
```

## Considerations

1. Check the return code after the MOVE TREE command is given. The return code is placed in the second field of the COMMBLOCK for the database to which the command refers. The eighth field (LEVEL) of the COMMBLOCK is set to 1 if there is an error for the "from" stack and is set to 2 if there is an error in the "to" stack. Return code 02 means the entry was missing; return code 03 means the entry was invalid.

2. You do not need exclusive use of the database for a MOVE TREE command.

3. The current password must have U-authority (update) for the schema record specified in the MOVE TREE command, that is, for the top node of the tree being moved. If it does not, the command is ignored, and return code 47 results.

4. Before the MOVE TREE command is issued, the specified "from" stack must already contain an entry for a data record corresponding to the specified subschema record, that is, for the top of the tree to be moved. Also, the "to" stack must contain an entry for a data record corresponding to the specified subschema record or to the parent of the specified subschema record. Return code 02 means the stack entry is missing, and return code 03 means the stack entry is invalid.

---

Abbreviations:
AFTER = AF
BEFORE = BE

5. The MOVE TREE command is not allowed in queue mode or load mode.

6. The MOVE TREE command forces a pending insert sequence to be processed.

7. If LHOLD equals YES and there is no global hold or exclusive use for the database, a hold is required on the "from" data record only, that is, the top node to be moved. If this is the case, establish the "to" stack before establishing the "from" stack (unless the multiple holds option is in effect) to preserve the hold on the "from" stack.

8. The rules for immediate mode INSERT commands govern the insert processing for MOVE TREE.

9. Upon completion of a MOVE TREE command, SYSTEM 2000 Multi-User software checks all the stacks of all PLEX user jobs (including the user job executing the MOVE TREE command). Any stack containing a reference to any data record that was moved is modified. This guarantees that any attempt to use any of the moved data records causes return code 03 or return code 09 to be issued, similar to that given for the REMOVE TREE command.

10. If the integer given for the "from" or "to" stack is negative or greater than 15, the command is ignored, and return code 40 is issued.

11. If Security by Entry is in effect for the database, a secondary password is allowed to move a subtree from one logical entry to another, provided the user knows both entry key values. The last where-clause specified in setting up the "from" and "to" stacks determines the current entry key after the MOVE TREE command completes processing.

12. If Rollback is enabled, SYSTEM 2000 software holds the following records: the top node of the tree being moved, the node of detachment, and the node of attachment.

13. The stack entry to be used in a MOVE TREE command might have been canceled in a previous REMOVE TREE or MOVE TREE operation by the program or by another user in a Multi-User environment. In this situation, the MOVE TREE command is ignored, and return code 09 results. The eighth field in the COMMBLOCK indicates the database level of the canceled entry.

## Examples

The following illustrations show different possibilities for updating with MOVE TREE. They are summarized here.

**Example 1:** MOVE TREE moves a subtree.

**Example 2:** MOVE TREE moves a logical entry.

## Example 1

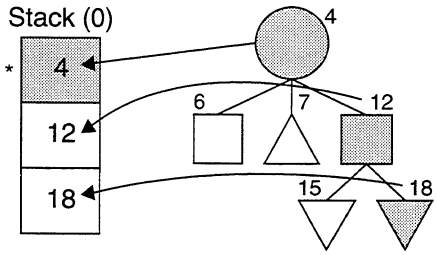
**1** Establish a "from" stack.

GET1 (0) ◯ WH ...

GETD (0) □ NEXT.

GETD (0) □ NEXT.

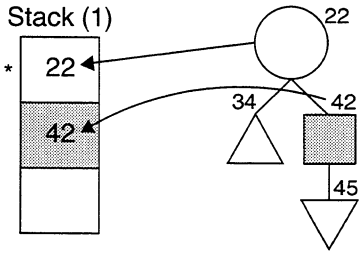
GETD (0) ▽ LAST.




**2** Establish a "to" stack.

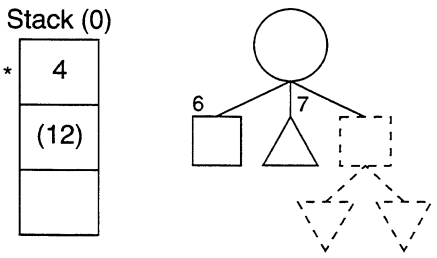
GET1 (0) ◯ WH ...

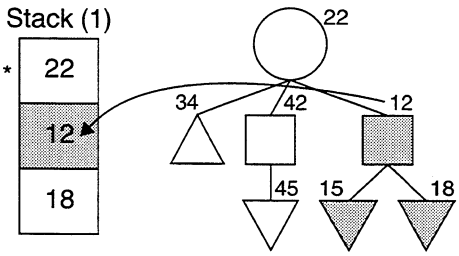
GETD (0) □ NEXT.




**3** Give a MOVE TREE command for the □ subschema using the default of AFTER with stack (0) as the "from" stack and stack (1) as the "to" stack.

MOVE TREE □ (0) TO (1).

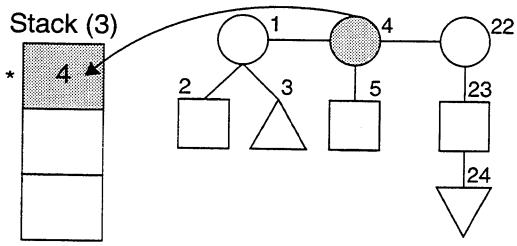


## Example 2

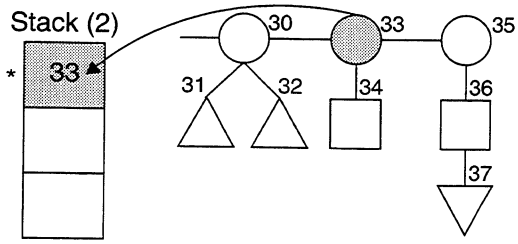
**1** Establish a "from" stack with a level 0 record.

GET1 (3) ◯ WHERE ...



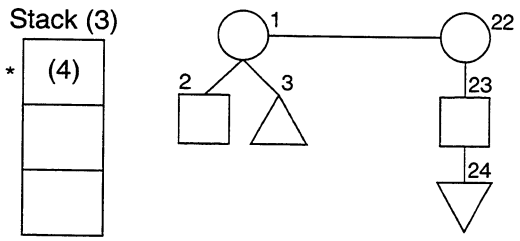

**2** Establish a "to" stack.

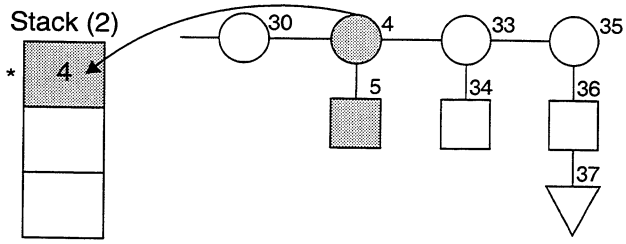
GET1 (2) ◯ WHERE ...




**3** Give a MOVE TREE command for the ◯ subschema using, for example, the BEFORE option.

MOVE TREE ◯ (3) TO (2) BEFORE



# CLEAR COMMANDS: MOVING UPDATED PAGES TO DISK

The various CLEAR commands protect the integrity of the database tables during update operations. The commands allow you to specify when modified database pages and/or Update Log pages are written from main memory to disk.

To alter a database, the pages of the database tables that need modification must be brought from disk to main memory. Ordinarily, an updated page is written back onto the database disk files by the buffer manager when the database is closed, the Rollback Log is reset, or the main memory occupied by that page must be made available for other processing. PLEX provides four forms of commands to give the user some control over the clearing of updated database pages from main memory to disk. These commands also allow the user to control the clearing of buffers that hold information to be written to an active Update Log (File 7, BDAM file). The various forms of the CLEAR commands not only clear pages from main memory to disk, but also set the mode of clearing, as described below.

SYSTEM 2000 software maintains an internal damage flag to indicate whether updated pages were written back to disk. The flag shows a damaged status as long as the modified pages are in main memory and are not written back to disk. It is marked clean after the pages are written to disk. There is also a main-memory damage flag that indicates the status of the database residing on disk and in main memory. This flag indicates a damaged status only during actual updating. The fourteenth COMMBLOCK field indicates the status of the database residing on disk and in main memory.

Any time you issue one of the CLEAR commands there is both an immediate result and a continuing effect. The immediate result is that either 1) all modified database pages and Update Log pages are cleared from main memory to disk, or 2) only Update Log pages are cleared. The continuing effect is the mode of clearing set by the CLEAR command, which is honored by SYSTEM 2000 software until end-of-job or until another CLEAR command is issued.

The immediate result of CLEAR and CLEAR AUTOMATICALLY is the clearing of all modified database pages and all Update Log buffers. If the keywords UPDATE LOG are specified in either command, that is, CLEAR UPDATE LOG or CLEAR UPDATE LOG AUTOMATICALLY, only Update Log buffers are cleared to the Update Log.

As mentioned, the continuing effect of any CLEAR command is that it sets the mode of clearing for subsequent updates in the job session. User-specified clear mode means that the PLEX program will issue a CLEAR or CLEAR UPDATE LOG command at specific intervals in the job session. Automatic clear mode means that a CLEAR AUTOMATICALLY or CLEAR UPDATE LOG AUTOMATICALLY command was issued and that SYSTEM 2000 software will automatically clear all buffers or just the Update Log buffers after each update operation.

The CLEAR commands affect only the user job that issues them. When the PLEX program has more than one database open at a time, the clearing and the mode of clearing affect all open databases. If an opened database does not have an active Update Log, return code 72 is issued. See *SYSTEM 2000 Messages and Codes, Version 12, First Edition* for a complete discussion of all PLEX return codes.

In addition to CLEAR commands given by users, SYSTEM 2000 software might be clearing buffers in order to obtain buffer space during processing of the user's commands. Also, SYSTEM 2000 software ensures that all buffers are cleared to disk after major operations such as RESTORE, RELOAD, CREATE/REMOVE INDEX, and so on, and before major processes, such as the SAVE command, QUEUE, TERMINATE, and REPORT. The CLEAR commands issued by the user clear buffers not cleared by these main processes and not cleared at the end of processing individual update commands.

The occasional use of a CLEAR or CLEAR UPDATE LOG command entails less I/O than clearing all buffers in automatic clear mode. Also, clearing only the Update Log automatically causes no I/O to modified database table pages and ensures that the database can be recovered in the event of a system failure.

The following illustration summarizes the immediate results and continuing effects of CLEAR commands. In a Multi-User environment and in single-user jobs, Update Log buffers are cleared to the Update Log (database File 7, a BDAM file).

**Illus. 10.1** Effects of CLEAR Commands

| | Memory-Resident Modified Database Pages* | | Memory-Resident Update Log Pages* | |
|---|---|---|---|---|
| Command | Immediate Results | Continuing Effects | Immediate Results | Continuing Effects |
| CLEAR | written to disk | user-specified clear mode | written to disk | user-specified clear mode |
| CLEAR UPDATE LOG | none | user-specified clear mode | written to disk | user-specified clear mode |
| CLEAR AUTOMATICALLY | written to disk | automatic clear mode | written to disk | automatic clear mode |
| CLEAR UPDATE LOG AUTOMATICALLY | none | user-specified mode | written to disk | automatic clear mode |

* For all databases that are currently open in the PLEX program.

## Format

```
CLEAR [UPDATE LOG]
CLEAR [UPDATE LOG] AUTOMATICALLY
```

short form of the commands and equivalents

```
CLEAR.          (CLEAR UPDATE LOG)
CLEAR AUTO.     (CLEAR UPDATE LOG AUTOMATICALLY)
```

## Example of Usage

The following example shows the immediate results and continuing effects of various user-specified CLEAR commands. Regardless of the clearing mode that is currently in effect, SYSTEM 2000 software will always clear modified database pages and Update Log buffers if more space is required during update processing. Also, all CLEAR commands and their continuing effects pertain to all databases currently open during the execution of a PLEX program.

| | |
|---|---|
| *(Begin job)* | |
| *(update commands)* | *(clearing done by buffer manager when the space occupied by the modified database pages or Update Log buffers are needed for other pages, the default)* |
| **CLEAR UPDATE LOG AUTO.** | *(clears Update Log buffers only)* |
| *(update command)* | *(clears Update Log buffers only)* |
| *(update command)* | *(clears Update Log buffers only)* |
| **CLEAR.** | *(clears database pages and Update Log buffers)* |
| *(update command)* | *(clears nothing unless space is needed)* |
| **CLEAR UPDATE LOG.** | *(clears Update Log buffers only)* |
| *(update command)* | *(clears nothing unless space is needed)* |
| **CLEAR AUTO.** | *(clears database pages and Update Log buffers)* |
| *(update command)* | *(clears database pages and Update Log buffers)* |
| **CLEAR UPDATE LOG AUTO.** | *(clears Update Log buffers only)* |
| *(update command)* | *(clears Update Log buffers only)* |
| **QUEUE.** | |
| *(update commands)* | |
| **TERMINATE.** | *(clears database pages and Update Log buffers)* |
| *(update command)* | *(clears Update Log buffers only)* |
| *(end of job)* | *(clears database pages and Update Log buffers)* |

If you do not issue a CLEAR command, clearing is done by the buffer manager when space occupied by the updated database pages and/or Update Log pages needs to be made available. SYSTEM 2000 software clears the database and Update Log pages in certain situations, as noted in item 3 of considerations.

## Considerations

1. Abnormal job termination (for example, telephone disconnect, system error, time limit) can be trapped by SYSTEM 2000 software as described in *SYSTEM 2000 Messages and Codes, Version 12, First Edition*. The following situations are possible:

   - For a database with rollback enabled, any CLEAR command issued within a frame has no effect other than to change the clearing made if rollback is disabled subsequently in the job.

   - A trapped abnormal termination of a user's job in a single-user environment clears database pages and Update Log pages from main memory to disk. Damage occurs only if the job terminates during an update.

   - In a single-user environment, if database pages in main memory were updated but are not yet cleared to disk before an untrapped termination of a user's job, the database on disk is in a damaged state. Frequent clearing of database pages reduces the chance of having a damaged database but incurs I/O. Clearing Update Log pages involves less I/O and affords recovery through the APPLY command.

2. In a Multi-User environment, CLEAR does not guarantee a clean database since other users might be updating the database.

3. To protect against damage from an untrapped abnormal termination, SYSTEM 2000 software clears the database pages and Update Log pages as a result of the execution of certain commands. Only the database associated with one of those commands is affected by the clearing. SYSTEM 2000 software ensures a clean database in the following situations:

   - when LOAD is issued

   - when the first update command in a QUEUE/TERMINATE session is issued (not when it is executed)

   - after completing all cycles applied by an APPLY command

   - after completing the TERMINATE process in both queue and load modes

   - after RESTORE from tape, if the archival database was clean

   - when CLOSE is issued.

     **Note**: If Rollback is enabled for a database, the database is only cleared when the Rollback Log is reset (see Coordinated Recovery in *SYSTEM 2000 CONTROL Language, Version 12, First Edition*).

4. Any form of a CLEAR command ends a pending insert sequence.

5. The CLEAR commands affect all databases opened in the PLEX program and thus do not set a return code in a COMMBLOCK. They always set a return code of zero in S2KRTC.

6. CLEAR commands are not accepted in load mode, that is, between LOAD and TERMINATE.

7. CLEAR commands do not perform any clearing in a QUEUE/TERMINATE session because clearing takes place automatically upon completion of the TERMINATE process. However, a CLEAR command issued in a QUEUE/TERMINATE session will reset the mode of clearing for updates performed after TERMINATE.

8. When a CLEAR command is issued in a SYSTEM 2000 session, the continuing effects that the command produces remain in effect until either another CLEAR command is issued or the job exits, even if the user changes databases within the session. See Illus. 10.1 on page 10-82 for the continuing effects of each CLEAR command.

Chapter 11

# Saving and Restoring


*SAVE 11-3*
- *Format 11-3*
- *Dynamically Allocating the Savefile for SAVE 11-4*
- *Examples of Syntax 11-5*
- *Example of Usage 11-6*
- *Considerations 11-7*

*RESTORE 11-8*
- *Format 11-8*
- *Dynamically Allocating the Savefile for RESTORE 11-9*
- *Examples of Syntax 11-9*
- *Example of Usage 11-10*
- *Considerations 11-10*

*KEEP 11-12*
- *Format 11-12*
- *Dynamically Allocating the Keepfile 11-12*
- *Specifying the Keepfile with JCL 11-13*
- *Example of Syntax 11-13*
- *Example of Usage 11-14*
- *Considerations 11-15*

*APPLY 11-16*
- *Recording Updates 11-16*
- *Update Cycles 11-16*
- *Using the Archival Database and the Keepfile 11-16*
- *Executing the APPLY Command 11-17*
- *APPLY with Coordinated Recovery 11-17*
- *Format 11-17*
- *Examples of Syntax 11-17*
- *Examples of Usage 11-18*
- *Considerations 11-20*

*SUSPEND 11-22*
- *Format 11-22*
- *Example of Syntax 11-22*
- *Example of Usage 11-23*
- *Considerations 11-24*

*RELEASE 11-25*
- *Format 11-25*
- *Example of Usage 11-25*
- *Considerations 11-26*

*ENABLE ROLLBACK AND DISABLE ROLLBACK 11-27*
- *Format 11-27*

*Considerations 11-28*

*FRAME AND END FRAME 11-29*
*Format 11-29*
*Frames, Update Cycles, and LUWs 11-30*
*Frames and Coordinated Recovery 11-30*
*Opening a Database and Coordinated Recovery 11-31*
*UPDATE and READ Options 11-31*
*UPDATE Option and the Damage Flag 11-31*
*FRAME and END FRAME Processing 11-32*
*IMMEDIATE and CONDITIONAL Options 11-32*
*Acceptable Commands in a Frame 11-32*
*Storing the Return Codes 11-33*
*Return Codes and the FOR Command 11-34*
*FRAME, END FRAME, and the Execution Modes 11-34*
*Effects of a Rejected FRAME Command 11-35*
*Effects of a Rejected END FRAME Command 11-35*
*Effects of Enabling and Disabling Rollback 11-35*
*Considerations for Opening Databases with FRAME 11-36*
*Holds and Exclusive Use of Databases 11-36*
*Clearing Database Pages 11-37*
*FRAME Command Continuation 11-38*
*Examples of Syntax 11-38*
*Examples of Usage 11-39*

*COMMIT 11-42*
*Format 11-42*
*Example 11-43*
*Considerations 11-44*

*ROLLBACK 11-45*
*Format 11-45*
*Example of Usage 11-46*
*Considerations 11-47*

*RESET ROLLBACK 11-48*
*Format 11-48*
*Examples of Usage 11-51*
*Considerations 11-53*

This chapter describes the administrative functions associated with saving and restoring a database, including update logging. A general understanding of the Update Log is assumed. The last two sections describe the commands for using the rollback feature of SYSTEM 2000 software. The Coordinated Recovery process and the philosophy of using the Update Log and Rollback Log are not covered in this manual. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for more information about these topics.

Most of the commands described in this chapter require the master password or the DBA password for successful execution. Also, many of these commands deal with the Update Log.

# SAVE

The SAVE command copies the specified database from disk to a Savefile (tape or disk). It produces an archival backup copy. Also, it can activate or suspend update logging for the database.

## Format

---

**SAVE** *database* **ON** *Savefile* [/ [*Keepfile*]]

---

*database* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

*Savefile*
|*volume-list* [**DS** = *sequence-number*]
|**DSN** = *data-set-name*

*volume-list*
*volume* [,*volume-list*]
(Up to ten volumes can be specified.)

*volume* is a variable containing a six-character, left-justified, blank-filled standard volume identification label.

*sequence-number* is a variable containing the data set sequence number for the identified file, so that $1 \leq n \leq 255$. If DS = *sequence number* is not included, the default is DS = 1. This specification applies only to the first volume of a multi-volume data set. Recording continues at the beginning (DS = 1) of any additional specified volumes. For COBOL, the usage is COMP.

*data-set-name* is a variable containing a 44-character, left-justified, blank-filled data set name conforming to IBM syntax.

/ activates update logging. If you omit the slash, update logging is suspended.

*Keepfile*
|*volume* [**DS** = *sequence-number*]
|**DSN** = *data-set-name*

(Only one volume can be specified.)

In the SAVE command, you specify a database name. SYSTEM 2000 software copies the disk files for that database onto a Savefile. This Savefile is a tape or QSAM disk file declared in the JCL to hold the archival database. On the Savefile, the individual files of the database constitute one file in a different format than the database disk files.

If you omit the optional slash, update logging (if already activated by a previous SAVE or RESTORE) is suspended; that is, no further recording of update commands occurs. If you include the optional slash, update logging is activated. It remains active for all subsequent jobs accessing the database until it is suspended.

The Update Log is database File 7, a preformatted BDAM file. Updates are copied onto the Keepfile (tape or disk) only when you issue the KEEP command. For more information about the Update Log and the commands associated with it, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition.*

A SAVE command copies database Files 1 through 6 to the specified volumes, regardless of your JCL. It also causes a request for additional volumes after exhausting those specified.

If you specify DSN for the Keepfile, the software stores the specified data set name for later use. When a subsequent KEEP command is given for that database, the software uses that Keepfile information.

## Dynamically Allocating the Savefile for SAVE

SYSTEM 2000 software dynamically allocates the Savefile if it is not already allocated in the JCL or CLIST. The Keepfile is dynamically allocated when you issue a KEEP or APPLY command. If the software dynamically allocates either the Savefile or Keepfile, it dynamically deallocates the file after SAVE, RESTORE, KEEP, or APPLY processing completes.

**Note**: If the data set is already cataloged, the data set information given in the SAVE command (for example, volume or DS) is ignored.

SYSTEM 2000 software performs the following steps before opening the Savefile:

1. Determines whether the DDname is specified in the JCL or CLIST, and, if so, uses that file.

2. Gives user exit EXIT09 a chance to dynamically allocate the file.

3. Looks for a cataloged file, using the default data set name or the DSN specified in the SAVE command.

4. Allocates a new Savefile on disk as specified in the SAVE command.

The default data set name for the Savefile is:

**DSN=***prefix.database***S**

When you activate update logging, SYSTEM 2000 software dynamically allocates database File 7, provided it is not already allocated in the JCL or CLIST. For information about dynamically allocating the Keepfile, see **KEEP** on page 11-12.

For a SAVE command, if the software does not find an existing Savefile data set, it creates the Savefile data set dynamically using the following attributes:

```
DSN=prefix.databaseS        (or the DSN specified in the SAVE command)
VOL=SER=volume-list         (or the default for SYSDA)
UNIT=SYSDA
RECFM=VB
LRECL= largest block size + 4
BLKSIZE= depends on type of disk
                            (The 3380 default is 23472.)
SPACE= primary space large  enough to save the database
                            (SYSTEM 2000 software knows
                             the total pages in use.)
```

## Examples of Syntax

```
SAVE EMPLOYEE ON SAS280.

SAVE EMPLOYEE ON SAS280/SAS40.

SAVE EMPLOYEE ON SAS40, SAS50, SAS60/SAS02.

SAVE EMPLOYEE ON SAS100, SAS110, SAS120 DS=16/SAS75 DS=3.

SAVE EMPLOYEE ON SAS130, SAS140/DSN=UPDVAR.
```

## Example of Usage

This example shows using SAVE in a Multi-User environment.

```
WORKING-STORAGE SECTION.
COMMBLOCK OF EMPLOYEE.
      :
      :
PROCEDURE DIVISION.
      :
      :
   (Put master password in COMMBLOCK)
   START S2K USE (1).
   (Check return code in S2KRTC)
            :
            :
   OPEN EMPLOYEE.
   (Check return code and damage flag
   in COMMBLOCK)
            :
            :
   (Move  values to SAS320, SAS310)

   SAVE EMPLOYEE ON SAS320/SAS310.
   (Updates to EMPLOYEE by different users)
            :
            :
   KEEP EMPLOYEE.
   (Updates to EMPLOYEE by different users)
            :
            :
   KEEP EMPLOYEE.
   (Move value to SAS330)
            :
            :
   SAVE EMPLOYEE ON SAS330/SAS310.
            :
            :
   (Updates to EMPLOYEE by different users)
```

## Considerations

1. After you issue a SAVE command, check the return code. It is placed in the second COMMBLOCK field for the database to which the command refers.

2. The program must open the database with the master password or the DBA password before it issues a SAVE command for that database. Otherwise, SYSTEM 2000 Error Code 024 is issued, and the program terminates. Also, if the DBA password is used and the SAVE command is not available to it, SYSTEM 2000 Error Code 024 occurs.

   If the database is not open, the SAVE command is ignored, and return code 92, 93, or 94 results.

3. All databases that are updated should be saved periodically, especially after a volume of updates or a large loading job.

4. The SAVE command suspends update logging unless it is specified. Issue a KEEP command before a SAVE if update logging was activated with a previous SAVE or RESTORE and there have been updates since the previous KEEP. Otherwise, these updates are not recorded on the Keepfile.

   SAVE does not save the Update Log. Even when update logging is activated in the SAVE command, you must issue a KEEP to copy the previous updates (if there are any on the Update Log) to the Keepfile. Activating update logging in the SAVE command reinitializes the Update Log.

5. For the Savefile, you can specify up to ten volumes. For the Keepfile, however, you can specify only one volume.

6. If you issue SAVE in queue mode for the database that is being updated, it is rejected with return code 28. However, SAVE is allowed for a different database.

7. If the SAVE command specifies no database name, or an invalid database name, the PLEX processor gives an error message for the command.

8. If you continue a SAVE command on more than one line, the first line must contain the database name and the keyword ON. Each subsequent line can contain one or more *volume* specifications, and all but the last specification must be followed by a comma. The DS or DSN specification for either the Savefile or Keepfile can appear on a separate line. The Keepfile and the slash for activating update logging can also appear on separate lines.

9. A SAVE command ends a pending insert sequence if it is for the same database.

10. If rollback is enabled, the SAVE command causes a database synchpoint and, therefore, a user synchpoint for Coordinated Recovery. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for a discussion about Coordinated Recovery.

# RESTORE

The RESTORE command copies the specified database from the Savefile (tape or disk) onto disk. You can use this command to restore an archived database or to recover a damaged database. Also, you can activate update logging for the database.

## Format

---

**RESTORE** *database* **FROM** *Savefile* [/ [*Keepfile*]]

---

*database* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

*Savefile* |*volume-list* [**DS** = *sequence-number*]
|**DSN** = *data-set-name*

*volume-list* *volume* [,*volume-list*]
(Up to ten volumes can be specified.)

*volume* is a variable containing a six-character, standard volume identification label, left-justified and blank-filled.

*sequence-number* is a variable containing the data set sequence number for the identified file, such that $1 \leq n \leq 255$. If

**DS** = *sequence-number*

is not included, the default is DS = 1. This specification applies only to the first volume of a multi-volume data set. Recording continues at the beginning (DS = 1) of any additional specified volumes. For COBOL, specify COMP usage.

*data-set-name* is a variable containing a 44-character data set name conforming to IBM syntax, left-justified and blank-filled.

/ activates update logging. If you omit the slash and update logging was activated by a previous SAVE command, it remains active.

*Keepfile* |*volume* [**DS** = *sequence-number*]
|**DSN** = *data-set-name*
(Only one volume can be specified.)

In the RESTORE command, you specify a database name. SYSTEM 2000 software copies the database files from the Savefile to disk. The Savefile is a tape or QSAM disk file that holds the archival database. On the Savefile, the individual files of the database constitute a single file (in a different format).

If the optional slash is missing, the status of update logging remains unchanged. That is, if update logging for that database was already activated by a previous SAVE, there is no need to use this option in RESTORE, unless you want to change the Keepfile. If update logging was not activated previously, you can activate it with the slash.

The RESTORE command (along with the Keepfile) can recover a damaged database. More information about update logging can be found in *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

## Dynamically Allocating the Savefile for RESTORE

SYSTEM 2000 software dynamically allocates the Savefile if it is not already allocated in the JCL or CLIST. The Keepfile is dynamically allocated when you issue a KEEP or APPLY command. If the software dynamically allocates the Savefile or Keepfile, it dynamically deallocates it after the SAVE, RESTORE, KEEP, or APPLY command completes.

**Note**: If the data set is already cataloged, the data set information given in the RESTORE command (for example, volume or DS) is ignored.

SYSTEM 2000 software performs the following steps before opening the Savefile:

1. Determines whether the DDname is specified in the JCL or CLIST, and, if so, uses that file.

2. Gives user exit EXIT09 a chance to dynamically allocate the file.

3. Allocates the file using the default data set name or the DSN specified in the RESTORE command. The data set must exist and must be cataloged so that SYSTEM 2000 software can allocate the Savefile.

When you activate update logging, SYSTEM 2000 software dynamically allocates database File 7, provided it is not already allocated in the JCL or CLIST.

## Examples of Syntax

```
RESTORE EMPLOYEE FROM SAS280.

RESTORE EMPLOYEE FROM SAS280/SAS2940.

RESTORE EMPLOYEE FROM SAS40, SAS50, SAS60/SAS02.

RESTORE EMPLOYEE FROM SAS100, SAS110, SAS120 DS=16/SAS75 DS=3.

RESTORE EMPLOYEE FROM SAS130, SAS140/DSN=UPDVAR.
```

## Example of Usage

This example recovers a database from damage assuming that a clean database was saved and that the updates were kept on the appropriate Keepfile.

```
COMMBLOCK OF EMPLOYEE.
        .
        .
        .
START S2K . . .
(Check return code in S2KRTC)
        .
        .
        .
(Move value to SAS280)
        .
        .
        .
RESTORE EMPLOYEE FROM SAS280.
APPLY ALL.
```

## Considerations

1. After you issue a RESTORE command, check the return code. It is placed in the second COMMBLOCK field for the database to which the command refers.

2. The program must place the master password or the DBA password in the COMMBLOCK for the database before it issues a RESTORE command for that database. Otherwise, SYSTEM 2000 Error Code 024 is issued, and the program terminates. That error code also occurs if the DBA password is used and the RESTORE command is not available to it.

3. You can recover a damaged database by restoring a clean copy from the Savefile and then applying the update commands from the Keepfile.

4. RESTORE does not suspend update logging. If update logging was activated by the previous SAVE command, it will still be active.

5. For *Savefile*, you can specify up to ten volumes. However, for the Keepfile, you can specify only one volume.

6. If you issue RESTORE in queue mode for a database that does not exist (a successful restore), return code 00 is issued. However, if the database exists, return code 48 occurs.

7. After a RESTORE command is executed, the stacks and Locate Files for the database are empty.

8. SYSTEM 2000 software guarantees an undamaged database when the database is restored from a copy of the undamaged database on the Savefile.

9. If the RESTORE command specifies no database name or an invalid database name, the PLEX processor issues an error message for the command.

10. When you continue a RESTORE command on one or more lines, the first line must contain the database name and, optionally, the keyword FROM. Each subsequent line can contain one or more *volume* specifications, and all but the last specification must be followed by a comma. The DS or DSN specification for either *Savefile* or *Keepfile* can appear on a separate line.

# KEEP

The KEEP command transfers processed updates from the Update Log (database File 7) to the Keepfile.

The software records updates in internal format on the Update Log. (See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for more information about the Update Log.) The Update Log for a database can be used after update logging has been activated in a SAVE or RESTORE command. The Update Log and Rollback Log are both required if rollback is enabled.

In a single-user or Multi-User environment, when a program performs a sequence of update operations, the updates are recorded in internal form on an intermediate file, called the Update Log. This is database File 7, which is a preformatted BDAM file. Periodically, the program gives the KEEP command to transfer the recorded updates from the Update Log to the Keepfile. The Keepfile is not dedicated for the duration of the program; it is used only when requested. When the Keepfile is on a tape drive (single-user jobs only), the tape is rewound and demounted after the copying operation is completed. The next KEEP command remounts the tape and positions it to the end of the previously recorded updates. The Keepfile cannot be tape in a Multi-User environment.

You can issue KEEP for a damaged database. It is normally used to save the most recent set of updates prior to restoring and recovery from database damage.

## Format

---

**KEEP** *database-identifier*

---

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

## Dynamically Allocating the Keepfile

The DDname for the Keepfile is the first seven characters of the database name plus a suffix of K. (The DDname KEEPFILE used in earlier releases is no longer valid.) SYSTEM 2000 software takes the following steps when opening the Keepfile:

1. Determines whether the Keepfile DDname is given in the JCL or CLIST, and, if so, uses that allocation.

2. Gives user exit EXIT10 a chance to dynamically allocate the Keepfile.

3. Allocates a new file for the first KEEP command, on disk or tape as specified in the SAVE or RESTORE command. Disk is the default.

4. Allocates an existing Keepfile with the default data set name or the DSN specified in the SAVE or RESTORE command. If the command is APPLY or a subsequent KEEP (after the first KEEP), the file must exist and must be cataloged in order for SYSTEM 2000 software to allocate the Keepfile.

If the KEEP command is the first KEEP and an existing data set is not found, SYSTEM 2000 software creates a Keepfile with the following attributes:

```
DSN=prefix.databaseK    (or the DSN in the SAVE or RESTORE command)
VOL=SER=volume          (or the default for SYSDA)
UNIT=SYSDA
RECFM=F
LRECL=same as database File 7
BLKSIZE=same as database File 7
SPACE=enough space to do the KEEP in primary space
  and secondary space of the same size to allow subsequent KEEPs.
```

## Specifying the Keepfile with JCL

You can allocate the Keepfile, or you can let SYSTEM 2000 software allocate it dynamically.

If you allocate the Keepfile with JCL or a CLIST, you can code all the information on the DDname *xxxxxxx*K and simply issue the 'KEEP:' command. Any information specified in the SAVE command syntax overrides that information in the DD statement. The only exception is keeping to an existing cataloged data set. Information from the catalog takes precedence over everything else.

## Example of Syntax

```
KEEP EMPLOYEE.          (copies recorded updates from the Update Log to the Keepfile)
```

## Example of Usage

Use of KEEP in a Multi-User environment.

```
WORKING-STORAGE SECTION.
      :
      :
     (Program statements)
      :
      :
COMMBLOCK OF EMPLOYEE.
      :
      :
     03 RETURN-KODE.
      :
      :

      :
      :
     (SSR-descriptions)

PROCEDURE DIVISION.
      :
      :
     START S2K USE (1).                  (specifies Multi-User environment)
     (Check return code in S2KRTC)
           :
           :
     RESTORE EMPLOYEE . . ./.            (activates update logging)
     (Check RETURN-KODE)
           :
           :
     (Update commands)
           :
           :
     KEEP EMPLOYEE.                      (requests the transfer of accumulated
                                         processed updates from the Update
                                         Log to the Keepfile)

     (Check RETURN-KODE)
           :
           :
```

## Considerations

1. After you issue a KEEP command, check the return code, which is placed in the second COMMBLOCK field for the database to which the command refers.

2. A program can use a secondary password or a DBA password to open a database before it gives a KEEP command for that database. The master password is not required.

3. If a program is in queue mode when it issues the KEEP command, the command is ignored, and return code 28 results.

4. If the program did not designate update logging in a SAVE or RESTORE command for a certain database, or if it gave a SUSPEND command without a subsequent SAVE or RESTORE that activates update logging, the software ignores a KEEP command referring to that database, and return code 35 results.

5. If a KEEP command specifies no database name or an invalid database name, the PLEX processor gives an error message for the command.

6. If the correct device for the Keepfile is not available, or if some other error occurs relating to this device, the KEEP command is ignored, and return code 39 results.

7. A KEEP command ends a pending insert sequence if the inserts are for the same database.

8. If the Update Log is empty when a KEEP command is given, return code 64 results. If the Keepfile is also empty and is on tape, the tape is mounted, and a label and logical end-of-file mark are written on it. The tape is then demounted.

9. Following a KEEP of the Update Log on a damaged database, the update cycle number is the cycle number of the last update that was kept. This is always equal to or greater than the cycle number prior to the KEEP. That is, the KEEP processing tries to keep as many complete update cycles as possible, regardless of the last cycle that was reflected in the Master Record.

10. If rollback is enabled, the KEEP command causes a database synchpoint and, therefore, a user synchpoint for Coordinated Recovery. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for a discussion of Coordinated Recovery.

# APPLY

The APPLY command recovers a database using an archival database and the updates recorded in internal form in a Keepfile (tape or disk).

- Updates recorded in internal form to the Keepfile can be processed without re-execution of the PLEX update program.
- You can specify some or all of the cycles of recorded updates for the recovery process.
- When rollback is enabled, the software applies cycles automatically with the Coordinated Recovery process.

## Recording Updates

Updates can be recorded in internal form in an Update Log, which is available after activating update logging in a SAVE or RESTORE command. The Update Log, (database File 7, a preformatted BDAM file) holds recorded updates until they are transferred to the Keepfile with a KEEP command. A Keepfile with an APPLY command enables recovery from errors during updating. (See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for more information about the Update Log.)

## Update Cycles

Updates to a database are performed in cycles. Each update cycle represents the database as it stands after an update operation is performed. Cycles are numbered consecutively; the number is increased one or more times depending on the execution mode as a sequence of update commands is executed.

Database damage occurring in any one cycle can be corrected in two steps. First, the database is released and then restored to an update cycle previous to the erroneous one. Second, cycles of updates are applied to the archival database up through a specified cycle number.

## Using the Archival Database and the Keepfile

In the first step, the database is released (with the RELEASE command) and then restored (with the RESTORE command). The master password or the DBA password must be present in the COMMBLOCK when either the RELEASE or the RESTORE command is given. This command causes the archival database (Savefile) to be copied to disk. Since the archival copy does not contain recent updates, its cycle number is lower than that of the database before the RELEASE command was given.

The cycles of updates in the Keepfile correspond to the update operations performed on the database after the archival copy of the database (Savefile) was saved. These cycles of updates in the Keepfile are now ready to be applied, without syntax scanning or update value checking.

## Executing the APPLY Command

An APPLY command can specify that all the cycles in the Keepfile are to be applied to the archival copy of the database or that only a range of cycles be applied, starting with the first cycle in the Keepfile. For example, if the Keepfile holds update cycles 20 through 35 for a database, the command can indicate that only cycles 20 through 29 are to be applied.

After the APPLY command is executed, the database is current as of the last update cycle that was applied. Updated database pages in memory are cleared to disk. The program can now perform new updates on the database, and the software records those updates in the Keepfile when you issue KEEP.

## APPLY with Coordinated Recovery

When the Rollback Log is enabled, the Coordinated Recovery feature is in force, and the Update Log must be allocated. During the recovery process to remove an uncommitted LUW, any committed LUW update cycles that were not removed since the last reset of the Rollback Log are automatically applied to the affected databases.

## Format

```
APPLY database-identifier  |THRU cycle-number
                           |ALL
```

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

*cycle-number* is the number of the last update cycle to be applied to the archival database: a constant or a variable with usage. For COBOL, specify COMP usage.

## Examples of Syntax

**APPLY EMPLOYEE ALL.**

*(specifies that all the update cycles in the Keepfile be applied to the restored, archival copy of the* EMPLOYEE *database)*

**APPLY INVENTORY THRU 69.**

*(specifies that all the update cycles through* 69 *in the Keepfile be applied to the archival copy of the* INVENTORY *database)*

**APPLY PARTS THRU LASTCYCLE.**

*(specifies that all the update cycles through the value for variable* LASTCYCLE *be applied to the archival copy of the PARTS database)*

## Examples of Usage

### Example 1

This example shows a sequence of PLEX commands and program statements showing recovery from an unrecoverable error with APPLY.

**Program execution**

(EMPLOYEE *database cycle number is now* 16.)

(*Program displays cycle number whenever it increases.*)

(*Updates that increase cycle number to* 20)

KEEP EMPLOYEE. (*transfers updates from the Update Log to the Keepfile*)

(*Updates that increase cycle number to* 28)

(*Updates that increase cycle number to* 31)

(*Unrecoverable error occurs*)

**Recovery sequence** (*resubmits cycles* 21 *through* 31 *rather than applying them from the Keepfile*)

RELEASE EMPLOYEE. (*releases the database from disk; master password required*)

RESTORE EMPLOYEE. . . (*restores the archival database (Savefile); master password required*)

APPLY EMPLOYEE ALL. (*applies all update cycles on the Keepfile, which in this case are all the cycles up through* 20 *since no* KEEP *commands were given after cycle* 20)

(*Resubmit updates beginning with cycle* 21.)

### Example 2

This example gives a sequence of PLEX commands and program statements showing recovery from unacceptable updates with APPLY.

**Program execution**

```
     (EMPLOYEE database cycle number
     is now 40.)

     (Program displays cycle number
     whenever it increases.)

     (Updates that increase cycle number
     to 55)

     KEEP EMPLOYEE.                  (transfers updates from Update
                                     Log to Keepfile)

     (Updates that increase cycle
     60 but are later found to
     be unacceptable)

     KEEP EMPLOYEE.
           :
           :
```

**Recovery sequence**

```
     RELEASE EMPLOYEE.               (releases the database from disk;
                                     master password required;
                                     necessary if the database has
                                     not yet been released from
                                     disk)

     RESTORE EMPLOYEE . . .          (restores the archival
                                     database (Savefile); master
                                     password required)

     APPLY EMPLOYEE THRU 55.         (applies update cycles through
                                     55, the last acceptable cycle,
                                     as shown in the program's
                                     display of cycle numbers
                                     during execution; master
                                     password required)

     (Resubmit corrected updates.)
```

## Considerations

1. After you issue an APPLY command, check the return code, which is placed in the second COMMBLOCK field for the database to which the command refers.

2. If the program does not have exclusive use of the database when it gives an APPLY command, the command is ignored, and return code 91 results. The program can obtain exclusive use by giving a RESTORE command or by issuing an OPEN command with the EXCLUSIVE option.

3. The program must place the master password or the DBA password in the COMMBLOCK for the database before it issues the RESTORE and APPLY commands for that database. Otherwise, SYSTEM 2000 Error Code 024 is issued, and the program terminates. The same error code occurs if the DBA password is used and the RESTORE and APPLY commands are not available to it.

4. Only one Keepfile can be used with an APPLY command for each restored database; this Keepfile is the one that was specified in the most recent SAVE or RESTORE command for that database. SYSTEM 2000 software marks the beginning of a Keepfile with certain information, including the name and cycle number of the associated archival database. If the name and cycle number of the restored database do not match this Keepfile information, the APPLY command is ignored, and return code 37 results. Other disk errors relating to the Keepfile bring the same result.

5. An APPLY command can be executed only if the cycle number in the Keepfile label agrees with the current cycle number in the COMMBLOCK for the restored database. Consequently, the program must not perform update operations between a RESTORE and an APPLY command for the same database. If it does, the APPLY command is ignored and return code 37 results. The same result is obtained when the program specifies a previously applied cycle number in an APPLY command.

6. If the program is in queue mode, the APPLY command is ignored and return code 28 results.

7. If the database is damaged, the APPLY command is ignored and return code 36 results.

8. If update logging is not activated for the database, return code 35 occurs. If the database cannot be opened, return code 93 occurs. Otherwise, APPLY opens the database with return code 00.

9. The APPLY command clears all the stacks for the database to which it refers.

10. If the APPLY command specifies no database name or an invalid database name, the PLEX processor gives an error message for the command.

11. An APPLY command ends a pending insert sequence if the inserts are for the same database.

12. For Coordinated Recovery, update cycles associated with valid LUWs (that is, LUWs that were not removed or are currently being removed) are automatically applied to the affected databases.

# SUSPEND

The SUSPEND command stops the recording of processed updates on the Update Log, that is, it deactivates update logging.

Updates can be recorded in internal form in an Update Log. The Update Log, (database File 7, a preformatted BDAM file) holds recorded updates until they are transferred to the Keepfile with a KEEP command. A Keepfile with an APPLY command enables recovery from errors during updating. (See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for more information about the Update Log.)

A program can stop the recording of processed updates at any time by giving a SUSPEND command for that database. The SUSPEND command does not affect the processed updates already recorded in the Keepfile.

The SUSPEND command has a twofold result. Not only does the command stop the recording of updates on the Update Log, but it also makes the Update Log unavailable to the program. However, the Keepfile remains unaffected. Therefore, if you want to keep any updates, you must issue a KEEP command before suspending update logging.

When another SAVE or RESTORE command for the same database activates update logging, the recording of processed updates starts anew with no relation to prior contents of the Keepfile. There is no way to consecutively apply two sets of Update Log data that have an intervening update logging suspension. After the SAVE or RESTORE command, the program can apply updates from the Update Log. Subsequent updates are recorded on the Update Log following the applied cycles.

## Format

**SUSPEND** *database-identifier*

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

## Example of Syntax

```
SUSPEND EMPLOYEE.          (stops the recording of processed
                           updates in the Keepfile for the
                           EMPLOYEE database. The command
                           also makes the Update Log disk
                           file unavailable to the program.)
```

## Example of Usage

Use of SUSPEND in a Multi-User environment.

```
WORKING-STORAGE SECTION.
      :
      :
COMMBLOCK OF EMPLOYEE.
      :
      :
     03 RETURN-KODE.
      :
      :
     (SSR-descriptions)

PROCEDURE DIVISION.
      :
      :
     START S2K USE (1).                    (specifies the Multi-User
                                            environment)
     (Check return code in S2KRTC)
      :
      :
     SAVE EMPLOYEE . . ./.                 (activates update logging)
     (Check RETURN-KODE)
      :
      :
     (Update commands)
      :
      :
     KEEP EMPLOYEE.
     (Check RETURN-KODE)
      :
      :
     (Update commands)
      :
      :
     KEEP EMPLOYEE.                        (ensures that the updates
                                            recorded on the Update Log
                                            are transferred to the
                                            Keepfile)
     (Check RETURN-KODE)
      :
      :
     SUSPEND EMPLOYEE.                     (suspends update logging:
                                            stops the recording of
                                            processed updates and
                                            makes the Update Log disk
                                            file unavailable to any
                                            user)
      :
      :
     (Check RETURN-KODE)
```

## Considerations

1. After you issue a SUSPEND command, check the return code. It is placed in the second COMMBLOCK field for the database to which the command refers.

2. The program must place the master password or the DBA password in the COMMBLOCK for the database before it issues the SUSPEND command for that database. Otherwise, SYSTEM 2000 Error Code 024 is issued, and the program terminates. Also that error code occurs if the DBA password is used and the SUSPEND command is not available to it.

3. The SUSPEND command causes the loss of the remaining recorded updates in the Update Log (database File 7). These updates can be preserved in the Keepfile by issuing a KEEP command before the SUSPEND command.

4. If you issue a SUSPEND command during queue mode for a database being updated, the command is ignored, and return code 28 results. However, a SUSPEND command is allowed in queue mode if it is for some database other than the one being updated.

5. After you issue a SUSPEND command, the updates recorded in the Keepfile are unavailable until the action discussed here is taken. You can give a RESTORE command for the archival database (Savefile), explicitly activating update logging with the same Keepfile. Then give an APPLY command, specifying the update cycles in the Keepfile that are to be applied to the archival database. New updates can then be executed, and they will be recorded in the Keepfile at a point immediately after the last recorded update that was just applied.

6. If a SUSPEND command specifies no database name or an invalid database name, the PLEX processor gives an error message for the command.

7. A SUSPEND command ends a pending insert sequence if it is for the same database.

8. If update logging is suspended when rollback is enabled, SYSTEM 2000 software activates an Update Log for internal use. The software continues to use an Update Log internally for Coordinated Recovery. Any update cycles recorded while the Update Log is being used internally are not available for your use.

# RELEASE

The RELEASE command makes the disk copy of the database unavailable for use by SYSTEM 2000 software.

When you do not need the disk copy of a database, you can give the RELEASE command to effectively delete it from disk. The files for the database are still allocated, but they are unavailable for use by SYSTEM 2000 software. The stacks and Locate Files for the database are canceled. However, the archival copy of the database (Savefile) is not affected.

Before the program issues the RELEASE command, it should obtain exclusive use of the database and place the master password or the DBA password in the COMMBLOCK for the database. If database recovery is to be performed by using the RESTORE command and the Update Log, the RELEASE command must be issued beforehand.

## Format

---

**RELEASE** *database-identifier*

---

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

## Example of Usage

A recovery sequence using the Update Log.

| | |
|---|---|
| **RELEASE EMPLOYEE.** | *(removes the copy of the disk database to prepare for restoring and recovery)* |
| **RESTORE EMPLOYEE. . .** | *(makes a fresh copy of the archival database on disk)* |
| **APPLY EMPLOYEE. . .** | *(applies the updates in the Keepfile through the specified cycle to the restored disk database files)* |

## Considerations

1. Check the return code after you give the RELEASE command It is placed in the second COMMBLOCK field for the database to which the command refers.

2. The program must place the master password or the DBA password in the COMMBLOCK for the database before it issues the RELEASE command for that database. Otherwise, SYSTEM 2000 Error Code 024 is issued, and the program terminates. Also that error code occurs if the DBA password is used and the RELEASE command is not available to it.

3. Before a program gives the RELEASE command in a Multi-User environment, it must obtain exclusive use of the database by issuing either a RESTORE command or an OPEN command with the EXCLUSIVE option. If the program does not have exclusive use of the database, the command is ignored, and return code 91 results.

4. If you issue a RELEASE command in queue mode for the database being updated, the command is ignored, and return code 28 results. However, a RELEASE command is allowed during a queue mode session if it is for some database other than the one being updated.

5. If a RELEASE command specifies no database name or an invalid database name, the PLEX processor assumes that the RELEASE command is a programming language command, and therefore, the PLEX processor does not generate code for the command and does not issue an error message. The error is caught by the compiler.

6. A RELEASE command ends a pending insert sequence if it is for the same database.

# ENABLE ROLLBACK AND DISABLE ROLLBACK

The ENABLE ROLLBACK command enables the Rollback Log for a specified database and indicates that the database should participate in any Coordinated Recovery processing that might be required. The DISABLE ROLLBACK command causes the database not to participate in any Coordinated Recovery processing and disables the Rollback Log.

Coordinated Recovery is a process by which SYSTEM 2000 software ensures the integrity of one or more databases during update sessions. In order for a specific database to be recovered automatically, both its Rollback Log and its Update Log must be enabled. These two logs contain all the information necessary to reinstate a database to a specific synchpoint. They contain update information for all users who are updating the database.

The ENABLE ROLLBACK command by itself does not cause Coordinated Recovery to occur. It allows the use of the Rollback Log for a specific database so that SYSTEM 2000 software can record before-images of all database table pages that are modified by any user. If a database needs to be reinstated, SYSTEM 2000 software uses these before-images and the updates recorded on the Update Log to ensure that all committed updates (through a synchpoint) are contained in the database. Uncommitted (or damaging) updates are skipped over in the process. Thus all users' updates are committed except the specific logical unit of work (LUW) for the one user job that caused the database(s) to be reinstated.

The effect of a ENABLE or DISABLE ROLLBACK lasts for all subsequent jobs accessing the database until the opposite command is given for the database. These commands require the master password and exclusive use of the database.

For a more complete discussion of Coordinated Recovery, the Rollback Log, and the Update Log, see *SYSTEM 2000 CONTROL Language, Version 12, First Edition*. See also the detailed syntax and description of the COMMIT command, the ROLLBACK command, and the RESET ROLLBACK command in this manual.

## Format

```
|ENABLE ROLLBACK FOR database-identifier
|DISABLE
```

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

## Considerations

1. Check the return code after you give an ENABLE ROLLBACK or a DISABLE ROLLBACK command. The return code is placed in the second COMMBLOCK field for the database to which the command refers.

2. The program must place the master password or the DBA password in the COMMBLOCK for the database before it issues an ENABLE or DISABLE ROLLBACK command for that database. Otherwise, SYSTEM 2000 Error Code 024 is issued, and the program terminates. Also that error code occurs if the DBA password is used and these commands are not available to it.

3. If you give an ENABLE ROLLBACK or DISABLE ROLLBACK command for a database that is already damaged, the command is ignored, and return code 16 results.

4. An ENABLE ROLLBACK or DISABLE ROLLBACK command can cause the opening of the specified database for exclusive use, as well as the enabling or disabling of rollback. If the program has already opened the database for exclusive use, the ENABLE ROLLBACK or DISABLE ROLLBACK command is executed without error. But if the program has already opened the database for non-exclusive use, the ENABLE ROLLBACK or DISABLE ROLLBACK command is ignored, and return code 16 results. In this situation, the program would have to close the database before giving the ENABLE ROLLBACK or DISABLE ROLLBACK command.

5. After you give an ENABLE ROLLBACK or DISABLE ROLLBACK command, the eighth COMMBLOCK field for the specified database indicates whether rollback was enabled prior to the command. If the field equals 1, rollback was enabled for the database before the command was given; if the field equals zero, rollback was not previously enabled.

6. The Rollback Log can be a multi-volume file.

7. If the rollback processor was not included in the system configuration and a program gives an ENABLE ROLLBACK command, a SYSTEM 2000 Error Code results, and the program is terminated.

8. Although the rollback recovery process undoes database damage caused by an abnormally terminated frame of commands, the Update Log must be used to recover from physical damage to the database.

9. If the Rollback Log is enabled, the exclusive open caused by the ENABLE/DISABLE ROLLBACK command causes a database synchpoint and a reset of the Rollback Log for the database. The exclusive open also causes a user synchpoint and any updates performed by the user since the last user synchpoint for that user (logical unit of work) are committed (permanently recorded) in the affected databases. For details, see the COMMIT, ROLLBACK, and RESET ROLLBACK commands. See also the Coordinated Recovery discussion in *SYSTEM 2000 CONTROL Language, Version 12, First Edition.*

10. If rollback is enabled, the MULTIPLE HOLDS option is automatically set to allow multiple holds. If rollback is later disabled, the MULTIPLE HOLDS option reverts to its previous setting.

# FRAME AND END FRAME

The FRAME and END FRAME commands indicate the beginning and end of a frame of update commands. In addition, FRAME causes rollback processing if a specified database is damaged, and it can open and begin a hold on one or more databases. These commands offer the following features:

- You can name up to 16 databases in one FRAME command.
- A FRAME command enables a program to obtain a global hold in immediate mode.
- A FRAME command enables a program to hold more than one global hold simultaneously if it specifies the UPDATE option for each database to be held.
- For the IMMEDIATE option, global holds obtained in the previous frame are maintained in the next frame.
- For the CONDITIONAL option, return code 93 is issued if any of the requested databases is unavailable.

## Format

```
FRAME      |[/IMMEDIATE/]     database-list
           |[/CONDITIONAL/]

END FRAME
```

*database-list* *database-specifications* **[,** *database-list***]**

*database-specifications* *database-identifier* **[***operation***]**

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

*operation* **|/UPDATE** *(default)*
**|/READ**

The FRAME command places a global hold on one or more databases. It also designates a synchpoint for Coordinated Recovery. That is, it commits all previous updates to the database(s) for a user job. Thus, if rollback is enabled and if the last frame fails to complete, the FRAME command ensures automatic recovery of all updated databases in the user's job back to the last synchpoint. The END FRAME command also causes a synchpoint.

Since using the FRAME commands causes a global hold on the affected databases, you should consider the tradeoffs offered by the use of local holds and the COMMIT command. For a Multi-User environment where several users need to update the same database(s) concurrently, local holds on selected records are not as restrictive as frames with global holds. See the COMMIT and ROLLBACK commands and the topic on multiple local holds in this manual. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for details about Coordinated Recovery.

## Frames, Update Cycles, and LUWs

If rollback is enabled for specific databases, you can give the FRAME command to specify the beginning of a frame of update commands involving those databases. The FRAME command establishes a user synchpoint. That is, updates in the frame constitute a logical unit of work (LUW) for that user. These updates can correspond to one or more update cycles; an update cycle is one update command or insert sequence, one QUEUE/TERMINATE session, or one LOAD/TERMINATE session.

Thus, the database cycle number can be incremented one or more times within a frame.

A logical unit of work (LUW) is considered uncommitted until a user synchpoint occurs, which commits the LUW. Once an LUW is committed, the associated updates are permanently recorded in the affected database(s) and are not removed if recovery occurs for a database.

Each FRAME command commits the previous frame of updates. The END FRAME also commits the previous frame. If the FRAME command is not used and rollback is enabled, an LUW is bounded by user synchpoints (for example, issuing a COMMIT command,) and no global holds are enforced. The use of COMMIT and the MULTIPLE HOLDS option allows more than one local hold on specific records in one or more databases and is less restrictive than global holds with frames in a Multi-User environment.

## Frames and Coordinated Recovery

If a frame fails to complete, Coordinated Recovery ensures that all previously committed updates for all databases (having rollback enabled) will be preserved. Only the uncommitted updates for the frame that failed are skipped. Thus, through recovery of each affected database, the cycle number for each database is restored to the latest cycle that was processed just before the FRAME command, that is, to the beginning of the last synchpoint.

Coordinated Recovery requires an Update Log and a Rollback Log for each database involved. For each update processed within a frame, certain database pages are selected for modification. These pages are first copied to the Rollback Log before the update operation is actually executed in the database. The Rollback Log holds the before-images of database pages, that is, the pages as they appear without changes caused by update commands in the current frame. Use of the Rollback Log requires the Update Log, which contains a record of all updates performed. If a frame fails, Coordinated Recovery uses the before-images from the Rollback Log and applies the updates from the Update Log to recover the databases to the cycle at the beginning of the frame. This means that all committed updates from all users in a Multi-User environment are recovered and only the updates from the current user frame that failed are skipped.

The first update issued after a FRAME command establishes an LUW for that user. This LUW is enclosed in a global hold on the database(s) involved, preventing updates from other users. The updates in the frame are uncommitted until another FRAME or END FRAME occurs. (COMMIT is not honored within a frame.) If no updates occur in a frame, an LUW is not established. The ROLLBACK command can be given within a frame to cancel all uncommitted updates. (See **ROLLBACK** on page 11-45.)

When Coordinated Recovery occurs, the recorded updates associated with the frame that failed to complete (that is, the uncommitted LUW) are removed from the Update Log. The KEEP command is not allowed within a frame. Therefore, the Keepfile contains none of the uncommitted updates and is not affected by Coordinated Recovery.

## Opening a Database and Coordinated Recovery

Not only does a FRAME command indicate the beginning of a frame and a synchpoint, but it also lists databases for which commands will be given during the frame. If the program has not yet opened one or more of the databases listed in a FRAME command, these databases are opened by the command. In addition, a FRAME command can cause recovery to occur for one or more of the specified databases. If rollback was previously enabled for a database named in a FRAME command, and if the damage flag for that database is set to signify damage, recovery for the database begins when the FRAME command is executed and the database is opened. (An OPEN command initializes recovery processing for such a database.) The execution of the program is suspended until recovery processing is completed.

After the completion of recovery processing, the execution of the program resumes. The program should check the database cycle number in the eleventh COMMBLOCK field. Retrieval and update commands can then be executed.

If a FRAME command specifies no database name or an invalid database name, the PLEX processor gives an error message for the command.

## UPDATE and READ Options

For each database listed in a FRAME command, UPDATE (the default) or READ can also be specified. The UPDATE option requests a global hold on the database and further indicates that the frame includes update commands for that database. Even though rollback was not enabled for a database, a global hold is still obtained with the UPDATE option, but the recovery process for that database will never be used. The global hold obtained prevents concurrent user updating of the database. That is, multiple local holds are not permitted for a database with a global hold. The READ option indicates that the frame includes only retrieval commands for the database, and no global hold is requested for that database.

## UPDATE Option and the Damage Flag

For each database for which rollback is enabled and a FRAME command specifies UPDATE, the database is marked as being damaged when the command is given, although no damage has in fact been caused at this point. If the frame ends abnormally (by a CLOSE command or by program termination during the frame), the database is still marked as damaged when a subsequent FRAME or OPEN command is given. Either command given for this damaged database causes recovery processing to occur if rollback is enabled. A FRAME command can cause recovery processing for a damaged database regardless of whether it specifies UPDATE or READ for that database.

## FRAME and END FRAME Processing

A frame ends normally when you issue an END FRAME command or another FRAME command. END FRAME (or another FRAME) performs the following functions:

1. terminates the current frame
2. ends an insert sequence, if there is one
3. commits the updates performed in the frame, that is, ends the logical unit of work (LUW) and clears the Update Log pages only
4. ends all global holds secured through the previous FRAME command
5. establishes a new user synchpoint.

The FRAME command clears all stacks and Locate Files used by the program, but the END FRAME command does not. Therefore, any retrievals needed for updates should be performed after a FRAME command is given. Stack entries and Locate File addresses obtained before the END FRAME is executed remain available afterward. Database pages are not cleared, but Update Log pages are.

## IMMEDIATE and CONDITIONAL Options

In a Multi-User environment, a FRAME command with the IMMEDIATE option retains global holds obtained in the previous frame. Also, for the IMMEDIATE option, all global holds previously obtained for the databases specified with UPDATE are continued in the new frame, regardless of whether other users have issued commands requesting either local or global holds on the same databases. However, the CONDITIONAL option ends all global holds previously obtained and indicates that return code 94 should be issued if a global hold cannot be obtained on all databases specified with UPDATE in the command. If a frame command does not include IMMEDIATE or CONDITIONAL, it cancels all global holds previously obtained by the program. In this situation, if the command specifies UPDATE (requesting a global hold) for any of the databases it lists, the execution of the program is suspended until all requested global holds are obtained.

## Acceptable Commands in a Frame

Within a frame, you can give retrieval commands for all databases named in the FRAME command, as well as for any databases that were already open to the program before the frame began. You can give the following retrieval commands within a frame:

```
DROP HOLD    GETD      LOAD
FOR          GET1      LOCATE
FRAME        GETP      ORDER BY
GET          LINK      ROLLBACK
GETA
```

However, you can issue update commands within a frame only for those databases specified with UPDATE (the default) in the FRAME command. The following update commands are acceptable within a frame:

```
INSERT          REMOVE
MODIFY          REMOVE TREE
```

In addition to specific retrieval and update commands, the following commands are also acceptable within a frame:

```
CANCEL QUEUE                          COPYS
CLEAR UPDATE LOG                      END FRAME
CLEAR UPDATE LOG AUTOMATICALLY        QUEUE
COPYL                                 TERMINATE
```

**Note**: When rollback is enabled for a database, the CLEAR commands are accepted within a frame, but no clearing is done. The CLEAR commands only cause the mode of clearing to be set in case rollback is subsequently disabled in the job session.

Return code 112 results from any command, except CLOSE, that is not specified in this list. For example, an OPEN, retrieval, or update command results in return code 112 if it would cause the opening of another database besides those already open before the frame began and those listed in the FRAME command. If rollback is enabled for a database that was framed, a CLOSE command specifying any database ends the frame abnormally. This abnormal end causes the framed database to be damaged since the database is still marked as being damaged when the frame ends.

## Storing the Return Codes

The return code for the END FRAME command is placed in S2KRTC (a variable in the S2KDUM control block) rather than in a COMMBLOCK since it is not associated with any particular database. S2KRTC holds the return code for any other PLEX command as well, including FRAME.

Since you can name more than one database in a FRAME command, the result of the command for one database can be different from that for another database named in the same command. If a non-zero return code results from FRAME processing of any database, S2KRTC contains that non-zero return code. Thus, if S2KRTC equals zero after you give a FRAME command, the command was successfully executed for each database, and the program does not need to check the return code in each of the COMMBLOCKs.

However, if the return code in S2KRTC is non-zero, at least one of the COMMBLOCKs for the databases named in the FRAME command also contains that non-zero return code. The return code in S2KRTC is from the COMMBLOCK of the first listed database that received a non-zero return code. Each of the following topics pertaining to a non-zero return code from a FRAME command mentions the COMMBLOCK(s) in which the return code is placed.

## Return Codes and the FOR Command

When a FRAME command is executed for several databases, FOR processing is invoked for the first listed database for which a non-zero return code results, if the database and the return code were both specified in a FOR command. If the first database for which a non-zero return code results was not specified in a FOR command, FOR processing is not performed as a result of the FRAME command.

Consider two situations in which a program gives FOR commands specifying databases A and B for all non-zero return codes. However, the program does not give a FOR command for either database C or database D. Suppose that a FRAME command specifies all four databases and results in non-zero return codes for databases A, B, and C, but a return code of zero for database D.

- Suppose the program gives the command

  ```
  FRAME D, A, B, C.
  ```

  FOR processing is invoked for database A, since it is the first listed database for which a non-zero return code occurred. No FOR processing is invoked for database B.

- However, suppose that the program gives the command

  ```
  FRAME D, C, A, B.
  ```

  FOR processing is not invoked since C is the first listed database for which a non-zero return code occurred; no FOR command was given for database C. No FOR processing is invoked for either database A or database B.

## FRAME, END FRAME, and the Execution Modes

If you give an END FRAME command during queue mode or load mode, the command is ignored and return code 101 results. A FRAME command issued in queue mode or load mode obtains the same result, and the return code is placed in the COMMBLOCK of the first database named in the command. The FRAME and END FRAME commands must be issued outside of queue and load mode, that is, while the program is in immediate mode.

## Effects of a Rejected FRAME Command

When a FRAME command is ignored (non-zero return code) while rollback is enabled, the frame is not established and the user job defaults to any updates issued establishing an LUW between user synchpoints. When you issue a FRAME command after a frame of commands, that previous frame ends normally even if the FRAME command results in a non-zero return code. If you give an END FRAME command after the most recent FRAME command is rejected, END FRAME is also ignored and return code 105 results. The same result occurs when END FRAME is issued and no FRAME command was given previously.

## Effects of a Rejected END FRAME Command

If a non-zero return code results (in S2KRTC) from an END FRAME command, the command was ignored. If an END FRAME command is ignored and rollback is enabled for a database that was framed, a CLOSE command or termination of SYSTEM 2000 processing will cause the database to be damaged. Reissue the END FRAME in order to end the frame normally.

Suppose, for example, that an insert sequence is ended by an END FRAME command, but for some reason the insertions are not successful. The END FRAME command is ignored, and a non-zero return code results. However, if the END FRAME command is issued again, it will be executed and the frame will end normally.

## Effects of Enabling and Disabling Rollback

When you give a FRAME or OPEN command, the eighth COMMBLOCK field for each specified database is set to show whether rollback was enabled for that database. If the field equals 1, rollback is enabled; if it equals zero, rollback is not enabled. However, this field does not indicate whether the command actually caused recovery processing to occur for the database.

Generally, a FRAME command indicates the databases that will be used during the frame and the kinds of operations (retrieval or update) that will be performed on them. In addition, the command signifies a request for a global hold on each database specified with UPDATE (the default). These indications by the FRAME command are significant even when rollback is not enabled for a database listed in the command.

Thus, regardless of whether rollback is enabled, if a database listed in a FRAME command is not specified with UPDATE (the default), an update command given for that database is ignored and return code 103 results. The return code is sent to the COMMBLOCK of the database to which the command refers.

Whether rollback is enabled or disabled for a database, return code 103 results if a retrieval command is given during a frame and it specifies HOLD for a database that was not specified with UPDATE (the default) in the FRAME command. The return code is sent to the COMMBLOCK of the database to which the retrieval command refers. However, for a database that is specified with UPDATE in the FRAME command, the HOLD option in a retrieval command has no effect within the frame.

If rollback is enabled and you give a CLOSE command for any database during a frame, the frame terminates abnormally. Consequently, all databases that were framed with UPDATE intent and with rollback enabled are damaged (the damage flag remains set at 1). Recovery processing occurs when the next FRAME or OPEN command is given for that database.

The damage flag for a database is set to 1 (signifying damage) as a result of a FRAME command only if rollback is enabled and if READ is not specified for that database. If rollback is not enabled, a FRAME command specifying UPDATE (the default) for a database does not cause the damage flag for that database to be set to 1.

If rollback is enabled, the last frame that is begun with a FRAME command specifying a database with UPDATE (the default) should end with the END FRAME command. If it does not, the database is damaged. The damage flag is still equal to 1.

If you issue a FRAME command for a database for which rollback is not enabled, the command is executed, and return code 85 is sent to the COMMBLOCK of the database. A global hold is still placed on all databases named in the command that are specified with UPDATE, regardless of whether rollback was enabled for any of them. However, no before-image recording is done in the Rollback Log for any database for which rollback is not enabled.

## Considerations for Opening Databases with FRAME

If a FRAME command would cause more than 63 databases to be open to a program at one time, the command is ignored, and return code 92 results. This return code is sent to the COMMBLOCK of the first database whose opening causes the program to exceed its limit of 63 open databases. Such a result can be obtained when, for example, a program already has 62 databases open and gives a FRAME command before it closes any of them. In this situation, all 62 remain open during the frame. However, suppose that the FRAME command lists two databases that are not opened by the program. In this situation, the command is ignored, and return code 92 is sent to the COMMBLOCK of the second database. Opening the second database would have caused the program to exceed its quota of 63 databases by one.

A FRAME command is ignored and results in return code 93 if a listed database cannot be opened because

- table limits were exceeded
- available memory was exhausted
- DD statements were omitted in the JCL for the job.

The same result occurs when the JCL for the job does not include a DD statement for the Rollback Log. Return code 93 is sent to the COMMBLOCK of the first listed database for which the error occurs.

## Holds and Exclusive Use of Databases

A FRAME command requests a global hold on each database that is specified with UPDATE (the default), regardless of whether rollback is enabled for the databases. A global hold and all local holds that the program obtained before giving the FRAME command are canceled, except in one situation: a subsequent FRAME command specifying IMMEDIATE continues global holds obtained through the previous FRAME command, provided that an intervening END FRAME command was not given.

The HOLD option for a retrieval command has no effect within a frame for a database specified with UPDATE in the preceding FRAME command. But if some database has not been so specified, a retrieval command referring to that database and specifying the HOLD option is ignored, and a non-zero return code results. This consideration for the HOLD option is true regardless of whether rollback is enabled for the database.

A FRAME command specifying IMMEDIATE is ignored, and return code 100 results if the command requests a global hold that was not obtained by a previous FRAME command. This result is obtained in the following situations:

- if it is the first FRAME command executed
- if it follows an END FRAME without an intervening FRAME command
- if it lists a database (specifying UPDATE or READ) that was not included in the previous FRAME command
- if it specifies UPDATE for a database for which READ was specified in the previous FRAME command.

The return code is sent to the COMMBLOCK of the first listed database for which the command is in error.

However, if a FRAME command specifies IMMEDIATE and also specifies READ with a database for which UPDATE was specified in the previous FRAME command, the command is executed, provided it contains no errors.

If a FRAME command specifies CONDITIONAL and lists a database that is unavailable because it is already being held by another user or is under exclusive use by another user, the command is ignored, and return code 94 results. The return code is sent to the COMMBLOCK of the first listed database that is unavailable.

A program can have only one database open for exclusive use at any one time. If a program already has a database open for exclusive use and issues a FRAME command listing a database other than that one, the command is ignored, and return code 96 results. This return code is sent to the COMMBLOCK of the first listed database that is not under exclusive use.

## Clearing Database Pages

The FRAME and END FRAME commands do not clear updated database pages. If rollback is enabled for a database and UPDATE is specified for that database in a FRAME command, the software does not clear database pages for the database, even if a CLEAR command of any kind is given by the same program or by some other user who has the database open. CLEAR commands result in return code 00 when they are issued within a frame.

In general, CLEAR commands have no effect for databases that have rollback enabled since database pages are cleared when the Rollback Log is reset. If the length of each frame is the default (that is, an LUW between synchpoints), the clearing effect is the same as if a CLEAR AUTO command had been issued. If a CLEAR AUTO type command is issued before a frame begins and a CLEAR command is given within the frame, no clearing occurs during the frame and automatic clearing does not resume after an END FRAME command is given. Similarly, if a CLEAR command is given before a frame begins and a CLEAR AUTO type command is issued within the frame, no clearing occurs during the frame, but automatic clearing begins if the frame ends with an END FRAME command.

## FRAME Command Continuation

If you continue a FRAME command to more than one line, the first line of the command must contain a database name and the READ or UPDATE (default) specification, if there is one, for that database. Each subsequent line must contain one or more database names together with the UPDATE or READ specifications for the listed databases. For COBOL, the final line of the command must end in a semicolon or a period, but each preceding line must end with a comma.

## Examples of Syntax

```
FRAME EMPLOYEE, PARTS, STOCK.          (requests three databases,
                                       EMPLOYEE, PARTS, and STOCK,
                                       for UPDATE, the default; does
                                       not specify either IMMEDIATE or
                                       CONDITIONAL)

FRAME/IMMEDIATE/ OFFICE,
     PROJECTS/READ.                    (specifies IMMEDIATE; requests
                                       the OFFICE database for UPDATE
                                       and the PROJECTS database for
                                       READ; updates not allowed)

FRAME/CONDITIONAL/ PARTS/READ,
     MACHINES.                         (specifies CONDITIONAL; requests the
                                       PARTS database for READ and the
                                       MACHINES database for UPDATE)

FRAME/CONDITIONAL/ OFFICE/UPDATE,
     EMPLOYEE.                         (specifies CONDITIONAL; explicitly
                                       requests UPDATE for the OFFICE
                                       database, implicitly requests the
                                       EMPLOYEE database for UPDATE, the
                                       default)

FRAME PROJECTS/UPDATE,
     STOCK/READ.                       (explicitly requests UPDATE for
                                       the PROJECTS database and
                                       READ for the STOCK database;
                                       does not specify either
                                       IMMEDIATE or CONDITIONAL)
```

## Examples of Usage

Sequence 1: Frames of commands in immediate mode.

```
FRAME OFFICE, PROJECTS.                    (begins a frame and requests UPDATE
                                           (the default) and global holds
                                           for the OFFICE and PROJECTS
                                           databases. Execution of the program
                                           is suspended until both global
                                           holds are obtained.)
      .
      .
      .
(Commands in frame, including
updates for the OFFICE and
PROJECTS databases)

END FRAME.                                 (ends the frame)

FRAME/CONDITIONAL/EMPLOYEE,OFFICE/READ.    (begins a frame with CONDITIONAL,
                                           requesting the EMPLOYEE database
                                           for UPDATE and the OFFICE
                                           database for READ. Because
                                           CONDITIONAL is specified if
                                           the requested hold cannot be
                                           obtained when this command is
                                           given, the command is ignored
                                           and return code 94 results.
                                           Execution of the program would
                                           continue, and the length of the
                                           frame would default to one
                                           update cycle.)
      .
      .
      .
(Commands in frame, including
commands for several update
cycles of the EMPLOYEE database;
only retrievals for the OFFICE
database)
      .
      .
      .
FRAME/IMMEDIATE/EMPLOYEE.                  (ends the previous frame and
                                           begins a new one specifying
                                           IMMEDIATE, so that the hold
                                           on the EMPLOYEE database
                                           continues)
      .
      .
      .
```

Sequence 2: Frames in queue mode.

```
FRAME EMPLOYEE, PARTS.                  (begins a frame while in
                                        immediate mode)
                                              :
                                              :
QUEUE.                                  (enters queue mode)
                                              :
                                              :
(updates for the EMPLOYEE
and PARTS databases, which
were specified for UPDATE
in FRAME command)
      :
      :
TERMINATE.                              (ends QUEUE/TERMINATE session)
                                              :
                                              :
QUEUE.                                  (reenters queue mode)
      :
      :
(more updates for the
EMPLOYEE and PARTS
databases)
      :
      :
TERMINATE.                              (ends QUEUE/TERMINATE session)
```

Sequence 3: A FRAME command that causes rollback processing.

| | |
|---|---|
| **FRAME OFFICE, PROJECTS.** | *(begins the first frame with* **UPDATE** *for the* **OFFICE** *and* **PROJECTS** *databases)* |
| . . . | |
| *(commands in frame, including updates to the* **OFFICE** *and* **PROJECTS** *databases)* | |
| . . . | |
| **FRAME/IMMEDIATE/ OFFICE.** | *(ends the first frame and begins a second one, retaining the global hold on the* **OFFICE** *database)* |
| . . . | |
| *(updates to the* **OFFICE** *database)* | |
| . . . | |
| *(program terminates without ending the frame)* | *(because no* **END FRAME** *was issued,* **database is damaged)** |
| **FRAME OFFICE.** | *(opens the* **OFFICE** *database. This causes Coordinated Recovery to occur automatically since the database is damaged. All updates performed in the second (latest) frame are removed and the database is recovered to its condition at the end of the first frame. When recovery is complete, a new frame (synchpoint) for* **OFFICE** *is begun.)* |

# COMMIT

The COMMIT command permanently records all updates performed since the issuing user's last synchpoint. That is, the updates are committed, and Coordinated Recovery cannot remove them from the affected databases. The COMMIT command

- is part of the Coordinated Recovery feature. Rollback must be enabled for the database.
- gives you an optimum way of making updates permanent, rather than using FRAME/END FRAME commands with their global holds.
- allows multiple users updating multiple databases concurrently to commit their updates at logical times during each user program.

## Format

```
COMMIT
```

For each user program, a logical unit of work (LUW) begins with the first update to a database. When the LUW is established, the software writes an LUW header record to the Update Log of the updated database (and any other database updated during that LUW). The header record becomes part of that unique LUW.

An LUW is uncommitted until a COMMIT command is issued or until another synchpoint occurs for that user. Then the LUW is committed.

When processing COMMIT, SYSTEM 2000 software writes an LUW trailer record on the Update Log; it indicates that the LUW is committed. A trailer record is written for each updated database that had rollback enabled. Then the Update Log buffers for the database(s) are cleared to disk, and that user's LUW is committed. If the LUW involved multiple databases, each LUW trailer record contains a list of all affected databases in that LUW.

After an LUW is committed, SYSTEM 2000 software drops all local holds that the program might have established. If another user program was suspended waiting for a record held by the user whose local holds were just dropped, that program is acquitted and is eligible for execution.

A complete discussion of the Coordinated Recovery feature (including synchpoints, LUWs, and so on) is given in *SYSTEM 2000 CONTROL Language, Version 12, First Edition*. Also, see the RESET ROLLBACK command for details about when synchpoints can occur.

## Example

```
WORKING-STORAGE SECTION.                  (program statements)
      :
      :
COMMBLOCK OF EMPLOYEE.
      :
      :
     03 EMP-RC.                           (SSR-descriptions)
           :
           :
COMMBLOCK OF PERSONNEL.
      :
      :
     03 PER-RC.
           :
           :
COMMBLOCK OF LIBRARY.
      :
      :
     03 LIB-RC.
      :
      :
PROCEDURE DIVISION.
      :
      :
     START S2K.                           (user synchpoint)
      :
      :
     OPEN EMPLOYEE PERSONNEL LIBRARY.     (assumes that rollback is enabled
                                          and that update logging is active)
      :
      :
     (EMPLOYEE and PERSONNEL
     databases are updated and
     LUW is established.)
      :
      :
     COMMIT.                              (permanently records the updates
                                          to EMPLOYEE and PERSONNEL and
                                          establishes a new synchpoint;
                                          commits the LUW)
      :
      :
     (updates to LIBRARY; new LUW
     is established.)
      :
      :
     CLOSE LIBRARY.                       (causes a user synchpoint that
                                          commits the LUW established for
                                          LIBRARY)
      :
      :
```

## Considerations

1. Because several databases might be updated in an LUW, the COMMIT command is not directly associated with one specific database. Therefore, the return code for COMMIT is placed in S2KRTC (a variable in the S2KDUM control block) rather than in a specific COMMBLOCK. If the S2KRTC return code is zero after a COMMIT, the uncommitted LUW for that user was successfully committed for all databases involved. However, if this return code is non-zero, the commit process failed for at least one of the databases involved.

2. The software drops all local holds for the program when COMMIT is issued.

3. For the COMMIT command to be effective, rollback must be enabled (see the ENABLE/DISABLE ROLLBACK command) and update logging must be activated for the affected databases.

4. For any database that has rollback enabled, COMMIT is ignored if the database was not updated.

5. COMMIT is not acceptable within a user frame. If you issue this command inside a user frame, the software places return code 112 in the S2KRTC field of the S2KDUM control block. Only a FRAME or END FRAME command can successfully end a user frame.

6. COMMIT is not acceptable in queue mode. If you issue this command in queue mode, the software places return code 28 in the S2KRTC field of the S2KDUM control block.

7. The COMMIT command causes a user synchpoint for Coordinated Recovery if rollback is enabled for an updated database. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for a discussion of Coordinated Recovery.

# ROLLBACK

The ROLLBACK command removes all updates performed since the user's last synchpoint without damaging the database. If several databases (with rollback enabled) were updated, they are all reinstated.

## Format

```
ROLLBACK
```

If rollback is enabled for a database, the ROLLBACK command invokes the automatic Coordinated Recovery feature, then establishes a new synchpoint for the user. The recovery process removes all updates to all databases in the uncommitted logical unit of work (LUW). Updates in an LUW are uncommitted, that is, not permanently recorded, until another synchpoint occurs in the program.

The recovery process for ROLLBACK is the same as the process that automatically removes an LUW for a transaction failure. Briefly, the software reinstates all updated database pages to their before-images, which were recorded on the Rollback Log. Then the Update Log for each database is positioned to the update cycle contained in each database Master Record. (The Master Record is always the first before-image database page recorded on the Rollback Log after the log is reset.) The software applies the updates recorded on the Update Logs to their respective databases.

Only databases associated with a specific user LUW are affected; no other user or LUW is affected by the recovery process. After the databases are recovered, all local and global holds for that user are dropped, and the program continues its execution.

You can use ROLLBACK to remove (cancel) any uncommitted LUW. This includes LUWs that began with either the FRAME command or the QUEUE command. However, ROLLBACK should be used with great care and consideration. The amount of recovery and processing overhead time could be quite substantial, depending on the number of before-images recorded on the Rollback Log since its last reset and the number of updates to be applied.

**Note**: For queue mode, the ROLLBACK command is not like the CANCEL QUEUE command, which just evicts update information recorded on scratch files. ROLLBACK should not be used as freely as CANCEL QUEUE.

See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for a complete discussion of the Coordinated Recovery feature, including synchpoints, LUWs, and so on. Also, see the RESET ROLLBACK command for additional details about when synchpoints can occur.

## Example of Usage

```
WORKING-STORAGE SECTION.                    (program statements)
       .
       .
COMMBLOCK OF EMPLOYEE.
       .
       .
   03 EMP-RC.
       .
       .
COMMBLOCK OF PERSONNEL.
       .
       .
     03 PER-RC.
       .
       .
COMMBLOCK OF LIBRARY.
       .
       .
     03 LIB-RC.

(SSR-descriptions)
       .
       .
PROCEDURE DIVISION.
       .
       .
     START S2K.                             (establishes a user synchpoint)
           .
           .
         OPEN EMPLOYEE PERSONNEL LIBRARY.
           .
           .
         (updates to all three databases)
           .
           .

         (The program logic determines
         that the updates just applied
         were in error.)
           .
           .
         ROLLBACK.                          (removes the updates performed
                                            since the last user synchpoint and
                                            establishes a new user synchpoint)
       .
       .
     (Processing continues.)
       .
       .
```

## Considerations

1. Because more than one database might be involved in an LUW, the ROLLBACK command is not directly associated with any specific database. Therefore, the return code for ROLLBACK is placed in S2KRTC (a variable in the S2KDUM control block) rather than in a specific COMMBLOCK.

   If the S2KRTC return code is zero after ROLLBACK is issued, the recovery was successful for each database involved. However, if this return code is non-zero, the recovery process failed for at least one of the databases.

2. ROLLBACK is ignored for any database that has rollback enabled but has not been updated. That is, the database is not part of the LUW.

3. ROLLBACK affects only those databases that have rollback enabled and update logging activated.

4. If you issue ROLLBACK within a user frame, the software removes all uncommitted updates in the current LUW, which began at the start of the frame. The software clears the stacks and Locate Files. Also, all global holds are dropped.

5. If you issue ROLLBACK while in queue mode, the software rolls the database back and evicts all update information that was written on scratch files. Also, all global holds for that user are dropped. The stacks and Locate Files are not affected.

6. ROLLBACK causes a user synchpoint for Coordinated Recovery if rollback is enabled for any database. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for a discussion of Coordinated Recovery.

# RESET ROLLBACK

The RESET ROLLBACK command specifies how much recovery takes place if either a transaction or a system failure occurs. This command

- permits user control of the amount of recovery.
- allows thresholds for resetting the Rollback Log, based either on a percentage of the log being filled or on a specific number of synchpoints being recorded on the log.

## Format

```
RESET ROLLBACK FOR database-id
   AFTER [x PERCENT] [ ,]  [y SYNCHPOINTS]
```

*database-id* is the name of either a database or a COMMBLOCK.

*x* is the percentage of the Rollback Log's primary space allocation that can be filled before the log is reset. It is either a numeric constant or a variable, equal to or greater than zero and equal to or less than 100 (that is, $0 \leq x \leq 100$). For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage with a picture suitable for the value. The default value is 50.

*y* is the number of user synchpoints containing a committed LUW that can be processed before the Rollback Log is reset. It is either a numeric constant or a variable, equal to or greater than 1 and equal to or less than 999999 (that is, $1 \leq y \leq 999999$). For COBOL, the variable can have DISPLAY, COMP-3, or COMP usage with a picture suitable for the value. The default value is 999999.

The RESET ROLLBACK command requires the master password or the DBA password. If the DBA password is used and RESET ROLLBACK is not available to it, SYSTEM 2000 Error Code 024 is issued. **Note**: The command also requires exclusive use of the database.

You can issue RESET ROLLBACK regardless of whether rollback is enabled or disabled. The software stores the PERCENT and SYNCHPOINT values in the Master Record of the specified database. They become effective only when rollback is enabled.

If PERCENT is zero, the software resets the Rollback Log at the start of each logical unit of work (LUW). If PERCENT is 100, the software resets the Rollback Log when a new LUW starts after the log's primary allocation is filled. If the SYNCHPOINT threshold is reached, the next user synchpoint that contains an LUW resets the Rollback Log.

Abbreviations:
ROLLBACK = RB
PERCENT = PCT
SYNCHPOINT = SPT

After either a PERCENT or SYNCHPOINT threshold has been reached, the software performs the following steps at the start of the next LUW:

1. establishes a database synchpoint. All currently uncommitted LUWs for that database continue until they are committed. New LUWs are not allowed to start until after the Rollback Log is reset.

2. writes any database pages currently in memory to the disk and turns off the database damage flag.

3. resets the Rollback Log to the beginning and writes the Master Record to the Rollback Log.

4. terminates the database synchpoint. The LUW and user synchpoint that triggered the reset continue.

The following commands cause a user synchpoint, as do any commands or operations that cause a database synchpoint:

| | |
|---|---|
| **COMMIT** | **TERMINATE** |
| **ROLLBACK** | **CLOSE** |
| **FRAME** | **START S2K** |
| **END FRAME** | **STOP S2K** |
| **QUEUE** | **DROP HOLD** |

A global hold causes both a database synchpoint and a user synchpoint to occur. The following commands cause global holds:

- all SCF update commands
- the FRAME command in both SCF and PLEX
- queue mode holds in PLEX
- SAVE and KEEP commands.

In addition to causing both a database synchpoint and a user synchpoint, the following commands and operations also reset the Rollback Log:

- an exclusive open of a database
- a physical close of a database
- a KEEP command
- a SAVE command
- a PLEX LOAD command
- a TERMINATE command for a PLEX LOAD command
- both the beginning and the end of processing for
  - a CREATE INDEX command
  - a REMOVE INDEX command
  - a REORGANIZE command
  - a RELOAD command
  - a DEFINE/MAP session with restructuring.

For complete details about why these commands and operations cause user synchpoints, database synchpoints, and a reset of the Rollback Log, see the Coordinated Recovery feature in *SYSTEM 2000 CONTROL Language, Version 12, First Edition*.

## Examples of Usage

### Example 1

In this example, one user program causes a reset of the Rollback Log because the PERCENT threshold is exceeded.

```
PROCEDURE DIVISION.
      :
      :
     START S2K.                        (establishes a user synchpoint)
           :
           :
     OPEN EMPLOYEE.                    (opens the EMPLOYEE database.  The
                                        Rollback Log is enabled and is
                                        49 percent full. The PERCENT
                                        threshold equals 50.)
           :
           :
     (updates to EMPLOYEE)             (The updates to EMPLOYEE fill
                                        the Rollback Log to 60
                                        percent of its primary
                                        space allocation.)
           :
           :
     COMMIT.                           (commits the LUW consisting of
                                        the updates since the previous
                                        synchpoint; establishes a new
                                        synchpoint)
           :
           :
     (update to EMPLOYEE)              (The first update at this synchpoint
                                        starts an LUW.  This LUW triggers a
                                        reset of the Rollback Log because
                                        the PERCENT threshold was exceeded
                                        in the previous LUW.  The Rollback
                                        Log is reset, the Master Record is
                                        written to the log, then the
                                        update is performed causing the
                                        necessary before-images to be
                                        recorded.  The LUW header and
                                        update cycle are written to the
                                        Update Log.)
           :
           :
```

**Example 2**

This example shows two users jobs that reset the Rollback Log because of the SYNCHPOINT threshold. Load mode in user 1's job resets the Rollback Log. Then user 1 sets the SYNCHPOINT threshold to 2. Remember that a user synchpoint must contain an LUW before it counts against the synchpoint threshold.

user 1

```
PROCEDURE DIVISION.

START S2K.  (user synchpoint)
      :
      :
OPEN EMPLOYEE.
      :
      :
LOAD.       (user synchpoint;
            resets the Rollback
            Log)
      :
      :
TERMINATE.  (user synchpoint;
            resets the Rollback
            Log)
      :
      :
RESET ROLLBACK FOR EMPLOYEE
AFTER 2 SYNCHPOINTS.
      :
      :
(updates to EMPLOYEE).
      :
      :
COMMIT.     (user synchpoint;
            commits LUW; first
            synchpoint toward
            threshold)
      :
      :
(update to EMPLOYEE)
             (starts LUW and
             synchpoint 3; log
             must be reset before
             the update can be
             applied.  User 1 is
             suspended until LUW 2
             by user 2 completes.)
      :
      :
(Reset of the Rollback Log
occurs after the LUW for
user 2 is committed.  User 1
is reactivated, and his LUW
continues.)
```

user 2

```
PROCEDURE DIVISION.
      :
      :
START S2K.  (user synchpoint)
      :
      :
OPEN EMPLOYEE.
      :
      :
(updates to EMPLOYEE)
             (starts LUW and
             synchpoint 2.
             Threshold is
             reached.)
      :
      :
(more updates to EMPLOYEE)
      :
      :
(more updates to EMPLOYEE)
      :
      :

CLOSE EMPLOYEE.
             (user synchpoint;
             commits LUW)
```

## Considerations

1. After you give the RESET ROLLBACK command, check the return code. It is in the second COMMBLOCK field for the specified database.

2. The database specified in a RESET ROLLBACK command must be opened with exclusive use. Otherwise, the software ignores the command and issues return code 91.

3. The RESET ROLLBACK command can only specify a database that was opened with the master password or the DBA password in its COMMBLOCK. Otherwise, the program abends with SYSTEM 2000 Error Code 024. Also, if the DBA password is not authorized to use this command, the same error code occurs.

Chapter 12

# FOR, $FOR, and $CFOR

*INTRODUCTION 12-1*

*CONVERTING FROM THE FOR COMMAND TO THE $FOR AND $CFOR DIRECTIVES 12-2*
*Example 12-4*

*FOR PROCESSING 12-5*
*Format 12-6*
*Examples of Syntax 12-7*
*Nesting the FOR Command 12-8*
*GO TO Option 12-8*
*NAME Option 12-8*
*Considerations 12-9*
*Example of Usage 12-10*

*THE $FOR DIRECTIVE 12-11*
*Format 12-12*
*Examples 12-13*
*Nesting $FOR Directives 12-15*
*GO TO Option 12-15*
*NAME Option 12-15*
*IGNORE Option 12-15*
*Considerations 12-16*

*THE $CFOR DIRECTIVE 12-17*
*Format 12-17*
*Examples 12-18*
*Considerations 12-19*

This chapter is specific to COBOL, FORTRAN, and Assembler PLEX. For PL/I information, see the PL/I appendix. Also, for FORTRAN, special considerations exist for the NAME option; See the FORTRAN appendix.

## INTRODUCTION

Two additional precompiler directives were introduced in Release 12.1: $FOR and $CFOR.

COBOL II does not allow you to pass the address of paragraph names as parameters. The CICS command-level interface does not allow direct branching back into the user's program. Because of these limitations, the $FOR and $CFOR directives have been introduced to replace the FOR commands in your PLEX programs.

The FOR command causes execution time calls into the SYSTEM 2000 interface. These calls instruct the interface to do alternate processing based on the return code from PLEX

commands. FOR processing is flexible in that program logic at execution time can dynamically alter the return codes being checked and the action to be taken.

The $FOR precompiler directive causes code to be generated after the PLEX commands that deal with the database specified in the $FOR statement. Unlike FOR commands, the $FOR directives do not cause the precompiler to generate calls into the interface.

With the FOR command, you can PERFORM a paragraph that contains FOR commands and modify the way you deal with return codes at execution time. With $FOR directives this is not the case. $FOR directives merely tell the precompiler to generate code after subsequent PLEX commands.

Therefore, the $FOR statement must be placed before any PLEX statement in the PROCEDURE DIVISION. In fact, since the $FOR statement itself does not cause a CALL statement, it can be coded in the WORKING-STORAGE SECTION anywhere after the COMMBLOCK definition for the database the $FOR command is to act upon. If you use a $COPY statement for your COMMBLOCK definition, you can include a global $FOR statement for that database with the $COPY member.

In terms of syntax, the conversion from using FOR commands to using $FOR and $CFOR directives is straightforward. All FOR commands can be converted to $FOR directives by putting a dollar sign ($) in front of the FOR command. While the $FOR directive has features the FOR command lacks, all FOR syntax is valid $FOR syntax.

The $CFOR directive is similar to the $FOR directive conceptually but very different from it syntactically. The $CFOR directive is used in conjunction with a PLEX command and is only in effect for that one command. It can be used in combination with FOR or $FOR but with different results:

- $CFOR conditions are checked after FOR conditions
- $CFOR conditions are checked before $FOR conditions.

$CFOR is not database-related. Therefore, you can code it with non-database commands such as START S2K, STOP S2K, and ROLLBACK.

# CONVERTING FROM THE FOR COMMAND TO THE $FOR AND $CFOR DIRECTIVES

Because you are changing from a return code check at run-time to IF statements generated within the PLEX program, physical placement of the $FOR statement is important.

The following situations provide guidelines to consider when converting FOR commands to $FOR and $CFOR directives in a program.

### Situation 1

A single FOR command to cover all PLEX commands for a given database. The FOR statement is located at the end of the program and is performed from the initialization paragraph.

Solution: Physically move the FOR statement to the beginning of the PROCEDURE DIVISION and change it to a $FOR.

### Situation 2

You use $COPY *member* throughout to be consistent in all your PLEX programs. The FOR command is in a $COPY *member*.

Solution: Modify the $COPY *member* to remove the FOR command. If you $COPY the database COMMBLOCK, place the $FOR directive in the $COPY book after the COMMBLOCK definition.

(NOTE: $FOR can be in the WORKING-STORAGE SECTION, while the FOR command must be in the PROCEDURE DIVISION.)

### Situation 3

You use multiple FOR commands to do specialized processing around certain PLEX commands.

Solution: Change the FOR commands to $FOR directives. For commands that require processing other than is specified by $FOR, use the $CFOR directive to do the specialized checking.

### Situation 4

You use multiple FOR commands, multiple CANCELs, and rely on program logic to determine which FOR command is active.

Solution: This is difficult for the $FOR directive since $FOR is not affected by program logic. Here are several possible solutions:

(a) Assuming your environment does not support the FOR .... GO TO option, change to a subroutine call with the FOR .... NAME option.

(b) Code a generic $FOR statement for each database at the beginning of the PROCEDURE DIVISION. Use the $CFOR for specialized processing on individual PLEX commands.

(c) Similar to (b): Use $FOR OFF to temporarily disable $FOR directives and code your own return code checking, then enable $FOR with $FOR ON. The $FOR OFF/$FOR ON technique is useful when return codes from multiple databases need to be checked.

## Example

Following are two PLEX programs. The first one uses multiple FOR commands to modify FOR processing. The second program assumes that the FOR commands use the GO TO option, and shows one way of converting to $FOR/$CFOR processing.

Existing PLEX program:

```
(Main program)
      .
      .
      .
 PERFORM MAINFOR (invokes normal FOR processing for a database)
 (PLEX commands)
 PERFORM CANFOR  (cancels FOR processing for the database)
 PERFORM SPCLFOR (invokes special FOR processing for the database)
 MODIFY ENTRY    (PLEX command with special FOR)
 PERFORM CANFOR  (cancels FOR processing for the database)
 PERFORM MAINFOR (returns to normal FOR processing for the database)
 (PLEX commands)
```

Converted PLEX program:

```
(Main program)
      .
      .
      .
 ($FOR directives to set up normal $FOR processing for a database)
 (PLEX commands)
 $FOR OFF                        (disables $FOR processing)
 MODIFY ENTRY $CFOR (rc,label) (PLEX command with special $CFOR)
 $FOR ON                         (enable $FOR processing)
 (PLEX commands)
```

# FOR PROCESSING

The FOR command is an aid for isolating return code checking from the program logic. Also, it automatically initializes a user-supplied error recovery routine when specified return codes occur. The FOR command offers the following features:

- You can establish the return code processing routines in several levels. You create or delete these levels with the NEW, REVERT, and CANCEL options.
- The error processing routine can be a paragraph name or a member in JOBLIB, STEPLIB, or the system link library that is dynamically loaded.

You can establish return code processing routines to handle exceptional conditions so that your program does not need to check for these conditions after every PLEX command. You can establish different routines for each condition and each database. The software automatically gives control to the appropriate routine when the specified return code occurs.

Here is a list of PLEX commands whose return codes can be covered by a FOR command. Each command relates to a particular database.

**APPLY**
**CLOSE**
**DROP HOLD** *database*
**ENABLE ROLLBACK/DISABLE ROLLBACK**
**ENABLE VALUES PADDING/DISABLE VALUES PADDING**
**FRAME** (*See consideration 7.*)
**FULL PASSES/NO FULL PASSES**
**GET**
**GETA**
**GETD**
**GET1**
**GETP**
**INSERT**
**KEEP**
**LOAD**
**LOCATE**
**MODIFY**
**MOVE TREE**
**OPEN**
**ORDER BY**
**RELEASE**
**REMOVE**
**REMOVE TREE**
**RESET ROLLBACK**
**RESTORE**
**SAVE**
**SUSPEND**
**TERMINATE**

Here is a list of commands whose return codes are not covered by a FOR command.

| | |
|---|---|
| **COPYL** | **ENABLE ROUTINE/DISABLE ROUTINE** |
| **COPYS** | **END FRAME** |
| **COMMIT** | **ROLLBACK** |
| **DROP HOLD ALL** | **START S2K** |
| | **STOP S2K** |

In addition, these commands are not covered by a FOR command. They always set a return code of 00 in S2KRTC.

| | |
|---|---|
| **CANCEL QUEUE** | **CLEAR UPDATE LOG AUTOMATICALLY** |
| **CLEAR** | **FOR** |
| **CLEAR AUTOMATICALLY** | **LINK** |
| **CLEAR UPDATE LOG** | **QUEUE** |

## Format

```
FOR database-identifier  |RETURN-CODE code-list  |GO TO paragraph
                                                 |NAME variable
                                                 |IGNORE
                         |REVERT
                         |CANCEL
```

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

*code-list*

```
|[     code      ]|
|[code THRU code]| [,code-list]
```

*code* is a valid PLEX return code. The return code on the right of THRU (or —) must be greater than that on the left.

*paragraph* is the name of an error-processing paragraph in the same program that has the associated FOR command.

*variable* is a variable containing the left-justified, blank-filled name or alias of a module in JOBLIB, STEPLIB, or the system link library. The variable must be eight characters long.

Abbreviations:
RETURN-CODE = RC
THRU = —

## Examples of Syntax

### Example 1

In this example for the EMPLOYEE database, if return code 04 is issued, control will be transferred to the label FINISH.

```
FOR EMPLOYEE RC 4 GOTO FINISH.
```

### Example 2

In this example, the module named by the value in MEMBER is loaded when the FOR is issued. The software calls it whenever a return code other than 00 or 04 occurs. It is deleted if the program issues another FOR command specifying REVERT or CANCEL for EMPLOYEE.

```
FOR EMPLOYEE RC 1 THRU 3, 5 THRU 127 NAME MEMBER.
```

### Example 3

This example establishes a new level of FOR processing for the EMPLOYEE database.

```
FOR EMPLOYEE NEW.
```

### Example 4

This example does not try to find a routine for return code 50, which was included in a previous FOR command. The program resumes the normal control flow if return code 50 occurs.

```
FOR EMPLOYEE RETURN-CODE 50 IGNORE.
```

### Example 5

For the EMPLOYEE database, forget all the FOR commands after the last NEW. That is, delete the current level so that the previous level becomes current.

```
FOR EMPLOYEE REVERT.
```

### Example 6

This example cancels all FOR commands for the EMPLOYEE database.

```
FOR EMPLOYEE CANCEL.
```

## Nesting the FOR Command

FOR commands are dynamically established in nested levels. Every time FOR *database* NEW is executed, a new level is created. This level becomes the current level for the database. When FOR *database* REVERT is executed, the current level is deleted so that all commands in this level (since the last NEW) are lost. The previous level becomes the current level.

When SYSTEM 2000 software issues a return code, it searches the commands of the current level for a paragraph name or module associated with that return code. If it finds the paragraph name or module, control is transferred to that paragraph or module. If it finds an IGNORE for that return code, the search is stopped. Otherwise, the previous level is searched. Thus, the search continues until the software finds a paragraph or module or an IGNORE for that return code, or until all levels are exhausted.

If a routine is not found, the normal flow of control is resumed in the program. The FOR *database* CANCEL command cancels the effect of all FOR commands at all levels.

## GO TO Option

When a FOR command includes the GO TO option, control is transferred to the specified paragraph (routine) for error processing, without returning to the point in the program where the error occurred. The paragraph must be present in the same program that issues the associated FOR command, along with the command causing a specified return code for the specified database.

If a FOR command appears in a calling program for a particular database and the called program contains PLEX commands for that same database, the called program must establish its own FOR level for possible return codes and its own error-processing paragraph (routine). To establish its own independent level of FOR processing, the called program must issue a FOR command specifying NEW, one or more FOR commands specifying return codes and paragraph names, and a FOR command specifying IGNORE for all other return codes not already covered. The called program should end with a FOR command specifying REVERT to deactivate its level of FOR processing, so that the level for the calling program once again becomes the current level.

The GO TO option with the FOR command does not work with COBOL II or the CICS command-level interface.

## NAME Option

Except in FORTRAN, when a FOR command includes the NAME option, SYSTEM 2000 software uses the specified module for error processing. This module is dynamically loaded from the JOBLIB, STEPLIB, or system link library the first time it is invoked. It is deleted only when the program issues another FOR command (with REVERT or CANCEL) that deactivates the associated level of FOR processing for the database. Each time the module is invoked, it is passed four parameters: S2KDUM, the database COMMBLOCK, the SSR specified in the PLEX command causing the return code, and a table of addresses pointing to the remaining parameters for the PLEX command. (For the NAME option in FORTRAN, see the appropriate appendix.)

## Considerations

1. If a cascaded retrieval occurs for a series of LINK commands, the resulting return codes in the table S2KLKH do not cause automatic transfer to return code routines named in a FOR command. Instead, when the cascade is completed, the return code resulting from the retrieval command that caused the cascade is the one used to call a specified return code routine. Therefore, the return codes in S2KLKH must be checked separately by the program, without the use of FOR commands.

2. The syntax USE *paragraph* can be substituted for GO TO *paragraph* in a FOR command. The two choices of syntax are processed identically.

3. If a FOR command specifies no database name or an invalid database name, the PLEX processor gives an error message for the command.

4. If a FOR command specifies a return code number outside the range 0-127, the PLEX processor gives an error message for the command.

5. If the return code specified to the right of THRU is not greater than the return code specified on the left, the PLEX processor gives an error message for the command.

6. A FOR command does not end an insert sequence.

7. When a FRAME command is executed for several databases, FOR processing is invoked for the first listed database for which a non-zero return code results if the database and the return code were both specified in a FOR command. If the first database for which a non-zero return code results was not specified in a FOR command, no FOR processing is performed as a result of the FRAME command.

   Consider two situations in which a program gives FOR commands specifying databases A and B for all non-zero return codes. However, the program does not give such a FOR command for either database C or database D. Suppose that a FRAME command specifies all four databases and results in non-zero return codes for databases A, B, and C, but a return code of zero for database D.

   - In the first situation, suppose that the program gives the command

     ```
     FRAME D, A, B, C.
     ```

     FOR processing is invoked for database A, since it is the first listed database for which a non-zero return code occurred. No FOR processing is invoked for database B.

   - In the second situation, suppose that the program gives the command

     ```
     FRAME D, C, A, B.
     ```

     FOR processing is not invoked, since C is the first listed database for which a non-zero return code occurred. No FOR command was given for database C. No FOR processing is invoked for either database A or database B.

## Example of Usage

Suppose the following FOR commands are executed in the stated order:

```
        FOR  DB  RC  1  GO TO  A.
        FOR  DB  RC  2  GO TO  B.
        FOR  DB  RC  3  GO TO  C.
        .
        .
        .
W:
        FOR  DB  NEW.
        FOR  DB  RC  1  GO TO  D.
        FOR  DB  RC  2  IGNORE.
        .
        .
        .
X:
        FOR  DB  REVERT.
        .
        .
        .
Y:
        FOR  DB  CANCEL.
        .
        .
        .
Z:
```

The effects of these commands are summarized in the following table:

| The Point in the Program Being Executed | Resulting Return Code | Paragraph to Which Control is Transferred |
|---|---|---|
| W | 1 | A |
| W | 2 | B |
| W | 3 | C |
| X | 1 | D |
| X | 2 | none |
| X | 3 | C |
| Y | 1 | A |
| Y | 2 | B |
| Y | 3 | C |
| Z | 1 | none |
| Z | 2 | none |
| Z | 3 | none |

**Note:** The outermost level need not start with FOR *database* NEW. However, a REVERT can cancel the commands in this level if it is given when the outermost level is the current level.

# THE $FOR DIRECTIVE

The $FOR directive was designed to replace the GO TO processing formerly available in the regular 'FOR' SYSTEM 2000 command. The newer versions of the CICS interface do not support the FOR...GO TO command. Additionally, COBOL II does not allow a label address as a parameter on a CALL.

The $FOR command is a SYSTEM 2000 precompiler directive that automatically generates IF...THEN...ELSE logic following a SYSTEM 2000 command. The generated code checks the return code from the command.

$FOR directives cause the precompiler to build blocks of information. These blocks are retained for the duration of the precompilation of the program, or until the $FOR CANCEL or $FOR REVERT directives are issued. The precompiler later uses these blocks to generate code inline in the user's program. Following PLEX commands that refer to the same database as the $FOR directives, this code is generated to check return codes from SYSTEM 2000. Unlike the FOR command, $FOR directives do not generate calls into the SYSTEM 2000 interface. $FOR is a compile-time directive instead of an execution-time command.

$FOR processing allows up to 10 levels of return code checking to be in effect at any given time. These levels can be controlled through the use of the REVERT, CANCEL, and NEW options.

Return code processing can be directed to an internal label or to a subprogram. A static call is made to the subprogram.

$FOR directive is used in much the same way as the FOR command, with the following exceptions:

- GO TO logic is allowed in all languages with $FOR directives.
- There are no special considerations for CICS code with $FOR directives.
- Three parameters are supplied to the module referred to in the NAME option: S2KDUM, COMMBLOCK and the SSR.
- The Boolean operators EQ and NE are allowed in conjunction with the return code-list. EQ is the default.
- $FOR processing can be temporarily suspended and then reinstated with the $FOR OFF/$FOR ON directives.
- The $FOR directive causes the precompiler to generate code in the user's program for the same commands whose return codes are covered by the FOR command.
- $FOR directives can appear anywhere in the PLEX program following the COMMBLOCK definition.

## Format

```
                          |ON
                          |OFF
$FOR database-identifier  |RETURN-CODE code-list  |GO TO paragraph
                                                  |NAME program
                                                  |IGNORE
                          |NEW
                          |REVERT
                          |CANCEL
```

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

*code-list*

```
|[ [opr]     code     ]|
|[ [opr]  code THRU code]| [,code-list]
```

You can include a Boolean operator to specify an EQ or NE condition. Although you can have several EQ conditions, each $FOR statement can have only one NE condition. If you do not specify a condition, EQ is assumed.

*opr* is a Boolean operator EQ, =, NE, or !=. The default is EQ. You can specify only one NE (or !=) for each $FOR directive.

*code* is a valid PLEX return code. The return code on the right of THRU (or —) must be greater than that on the left.

*paragraph* is the name of an error-processing paragraph in the same program as the associated FOR command.

*program* is the name of the program to be called.

The format of the $FOR directive is generally the same as the FOR command, except that $FOR includes the ON and OFF options and the Boolean operators EQ and NE.

Syntactic units cannot be split across lines.

---

Abbreviations:

RETURN-CODE can be abbreviated RC
THRU can be —
EQ can be =
NE can be !=

## Examples

### Example 1

This is a simple check of a single return code.

```
$FOR EMPLOYEE RC 4 GOTO FINISH.
GET1 BOOKS WH TITLEX EQ.
```

The precompiler generates the following code after the PLEX command:

```
IF S2KRTC = 4
GO TO FINISH
ELSE
MOVE ZEROS TO S2KDU5.
```

### Example 2

A second $FOR directive will be added to the conditions of the first $FOR. The last $FOR directive is expanded first.

```
$FOR LIBRARY RC 1 THRU 3, 5 THRU 127 NAME MEMBER.
GET1 BOOKS WH TITLEX EQ.
```

The precompiler generates the following code after the PLEX command:

```
IF (S2KRTC > 1 OR = 1) AND (S2KRTC < 3 OR = 3) OR
(S2KRTC > 5 OR = 5) AND (S2KRTC < 127 OR = 127)
CALL 'MEMBER' USING S2KDUM  STACKS-0 BOOKS-0
ELSE
IF S2KRTC = 4
GO TO FINISH
ELSE
MOVE ZEROS TO S2KDU5.
```

### Example 3

NEW has no effect until the next $FOR directive is encountered.

```
$FOR LIBRARY NEW.
GET1 BOOKS WH TITLEX EQ.
```

The precompiler generates the following code after the PLEX command:

```
IF (S2KRTC > 1 OR = 1) AND (S2KRTC < 3 OR = 3) OR
(S2KRTC > 5 OR = 5) AND (S2KRTC < 127 OR = 127)
CALL 'MEMBER' USING S2KDUM  STACKS-0 BOOKS-0
ELSE
IF S2KRTC = 4
GO TO FINISH
ELSE
MOVE ZEROS TO S2KDU5.
```

**Example 4**

NEW has no effect until the next $FOR directive. The latest $FOR directive is expanded first.

```
$FOR LIBRARY RETURN-CODE 50 IGNORE.
GET1 BOOKS WH TITLEX EQ.
```

The precompiler generates the following code after the PLEX command:

```
IF S2KRTC NOT = 50
IF (S2KRTC > 1 OR = 1) AND (S2KRTC < 3 OR = 3) OR
(S2KRTC > 5 OR = 5) AND (S2KRTC < 127 OR = 127)
CALL 'MEMBER' USING S2KDUM  STACKS-0 BOOKS-0
ELSE
IF S2KRTC = 4
GO TO FINISH
ELSE
MOVE ZEROS TO S2KDU5.
```

**Example 5**

REVERT deletes any $FOR directives encountered since the last $FOR...NEW directive.

```
$FOR LIBRARY REVERT.
GET1 BOOKS WH TITLEX EQ.
```

The precompiler generates the following code after the PLEX command:

```
IF (S2KRTC > 1 OR = 1) AND (S2KRTC < 3 OR = 3) OR
(S2KRTC > 5 OR = 5) AND (S2KRTC < 127 OR = 127)
CALL 'MEMBER' USING S2KDUM  STACKS-0 BOOKS-0
ELSE
IF S2KRTC = 4
GO TO FINISH
ELSE
MOVE ZEROS TO S2KDU5.
```

**Example 6**

This example cancels all $FOR commands for the LIBRARY database.

```
$FOR LIBRARY CANCEL.
GET1 BOOKS WH TITLEX EQ.
```

The precompiler does not generate any condition checking following the PLEX command.

## Nesting $FOR Directives

$FOR directives are established in nested levels called blocks. Every time $FOR *database* NEW is executed, a new block is created. This block becomes the current block for the database. When $FOR *database* REVERT is processed, the current block is deleted so that all directives in this block (since the last NEW) are lost. The previous block becomes the current block.

If a PLEX command affects a database for which there are $FOR directives in effect, the precompiler generates conditional statements following the PLEX statement. These statements are in accordance with the $FOR directives which are processed as last in, first out. This means that the last $FOR block that was built is the first used to generate conditional statements following a PLEX statement. The $FOR *database* CANCEL directive cancels the effect of all $FOR directives in all blocks that are related to that database.

## GO TO Option

When a $FOR directive includes the GO TO option, the paragraph specified in the GO TO option must be present in the same program that issues any PLEX commands for the specified database.

If a $FOR directive appears in a calling program for a particular database and the called program contains PLEX commands for the same database, the called program must have its own $FOR directives for possible return codes and its own error-processing paragraph (routine).

## NAME Option

The SYSTEM 2000 precompilers generate static call statements for the module specified in the NAME option. If the compiler used for the PLEX program supports dynamic loading, the called module can reside in the JOBLIB, the STEPLIB, or the system link library. Otherwise, the called module must be linked with the PLEX program. Each time the module is called, it is passed three parameters: S2KDUM, the database COMMBLOCK, and the SSR specified in the PLEX command. If there is no SSR specified in the PLEX command, the database COMMBLOCK will be passed twice, as the second and third parameters.

## IGNORE Option

Within a $FOR block, return codes specified with $FOR with the IGNORE option are always processed first, regardless of the order in which they are presented. For example, if you specify the sequence $FOR ... IGNORE, $FOR ... GO TO, $FOR ... IGNORE, the return codes specified in the two IGNOREs are checked before the return codes with the GO TO option.

## Considerations

1. With COBOL and ASSEMBLER, the USE option of the $FOR directive causes GO TO logic to be generated, but with PL/1 and FORTRAN, it causes CALL statements to be generated.

2. If a $FOR directive specifies no database name or an invalid database name, the PLEX processor gives an error message for the directive.

3. If a $FOR directive specifies a return code number outside the range 0-255, the PLEX processor gives an error message for the directive.

4. If the return code specified to the right of THRU is not greater than the return code specified on the left, the PLEX processor gives an error message for the directive.

5. The GO TO option of the $FOR directive can be used with COBOL II and the CICS command-level interface.

6. The $FOR directive can be used with PLEX commands that do not reference a subschema record. In this case, if the NAME option is used, the COMMBLOCK is passed as both the second and third parameters.

7. Order of return code checking:

   - FOR conditions (checking done in the interface)
   - $CFOR conditions
   - $FOR conditions.

# THE $CFOR DIRECTIVE

$CFOR directives work in much the same way as $FOR directives, but they are in effect only for the PLEX command to which they are appended. The information used to generate the code for the $CFOR directive is retained only during the processing of one PLEX command.

The $CFOR directives specify an internal label to be branched to when the interface returns to the PLEX program and the $CFOR conditions are met. The $CFOR directives can be specified on any SYSTEM 2000 command with the exception of the FOR and LINK commands.

## Format

---

*PLEX command* **$CFOR** (*condition*) [,(*condition*),...]

---

*PLEX command* is any applicable PLEX command such as CLOSE *database*

*condition* is one or more conditions of the form:
([*opr*] *rc*, *label*)
([*opr*] *rc* **THRU** *rc*, *label*)

*opr* is a Boolean operator EQ, =, NE, or !=. The default is EQ. You can specify only one NE (or !=) for each $CFOR directive.

*rc* is the return code or range of return codes to check. If you are specifying a range, the left value must be less than right value.

*label* is the name of a paragraph in the same program.

If multiple conditions are coded, they are included in the same IF statement as long as the label they are branching to does not change. Changing the label causes the precompiler to generate an ELSE IF clause.

Syntactic units cannot be split across lines.

---

Abbreviations:
THRU can be —

## Examples

### Example 1

```
START S2K(0) USE(BAM) $CFOR (NE 0,BADERR).
```

The precompiler generates the following code after the PLEX command:

```
IF S2KRTC NOT = 0
GO TO BADERR
ELSE
MOVE ZEROS TO S2KDU5.
```

### Example 2

This example uses $CFOR in conjunction with $FOR:

```
$FOR PERSONNEL RC NE 0 GO TO BADERR.

CLOSE PERSONNEL $CFOR (26,NOT-OPEN),(EQ 27,QUEUE-CANCELLED).
```

The precompiler generates the following code after the PLEX command:

```
IF S2KRTC = 26
GO TO NOT-OPEN
ELSE
IF S2KRTC = 27
GO TO QUEUE-CANCELLED
ELSE
IF S2KRTC NOT = 0
GO TO BADERR
ELSE
MOVE ZEROS TO S2KDU5.
```

**Example 3**

This example uses $CFOR in conjunction with $FOR, with $ FOR temporarily disabled:

```
$FOR PERSONNEL RC NE 0 GO TO BADERR.
$FOR OFF.
CLOSE PERSONNEL $CFOR (26,NOT-OPEN),(EQ 27,QUEUE-CANCELLED).
$FOR ON.
```

The precompiler generates the following code after the PLEX command:

```
IF S2KRTC = 26
GO TO NOT-OPEN
ELSE
IF S2KRTC = 27
GO TO QUEUE-CANCELLED
ELSE
MOVE ZEROS TO S2KDU5.
```

The previous example shows that while $FOR OFF is in effect, no $FOR directives are used to generate code, although $CFOR directives can still be used.

## Considerations

1. If a $CFOR directive specifies a return code number outside the range 0-255, the PLEX processor gives an error message for the directive.

2. If a SPANS condition is encountered, the return code specified on the right of the - must be greater than the return code specified on the left. Otherwise, PLEX processor gives an error message for the directive.

3. Order of return code checking:

   - FOR conditions (checking done in the interface)
   - $CFOR conditions
   - $FOR conditions.

Chapter 13

# ENABLE/DISABLE VALUES PADDING

*INTRODUCTION 13-1*
*Format 13-4*
*Considerations 13-5*

## INTRODUCTION

The VALUES PADDING command determines the placing of new key values in the Distinct Values Table. Only the master password holder or the DBA password can issue this command.

- The DISABLE option turns off padding. The software will fill the last index page of each key item completely before creating the next new page at the end of the index.
- The ENABLE option requests partial filling of the last index page of each key item when inserting there. A new last page is created to leave space for future additions in each page.

The Distinct Values Table (DVT) contains an index for each key item. This index contains an entry for each distinct value of that key item, and the values are sorted in ascending order. Each value in the index is associated with a pointer to relate the value to the Hierarchical Table (HT) addresses of the data records that the value belongs to. For example, in the EMPLOYEE database (see the foldout at the end of this manual), the index for the item EMPLOYEE NUMBER would have the following entries, among others, in the order shown here:

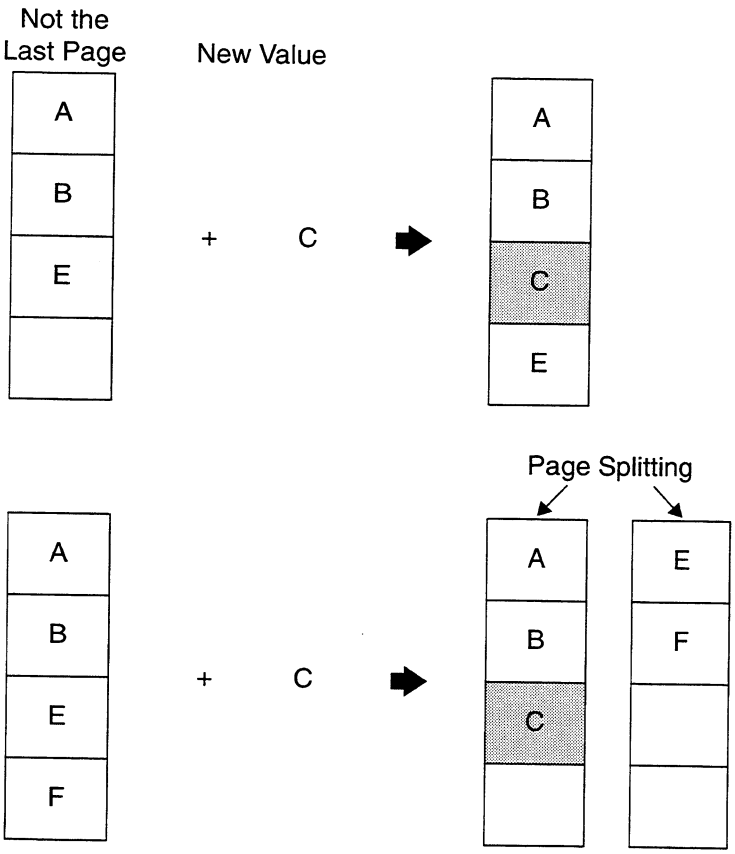
| Value | HT address |
|---|---|
| 1043 | 1 |
| 1120 | 12 |
| 1265 | 24 |

The same value can appear in more than one data record. For example, in the EMPLOYEE database, the value MALE or FEMALE appears in the level 0 records. When a value occurs repeatedly for the same item in the database, the index for that item does not include the HT addresses of all records containing that value. Instead, the index points to an entry in another table, called the Multiple Occurrence Table (MOT), which stores the HT addresses. As more occurrences of an item's value are added to a database, the MOT entry for that value expands.

In the Distinct Values Table, the index for an item contains one or more pages, and it grows as new values are added. Indexes of different items are always kept separate. That is, the values of different items are never mixed in a page.

When a value is to be added to a page that is not full, the higher existing values are moved down on the page, and the value is entered after the next lower value. If the page is full, it is split into two partially filled pages, and the new value is added to the appropriate partially filled page, as shown in Illus. 13.1.

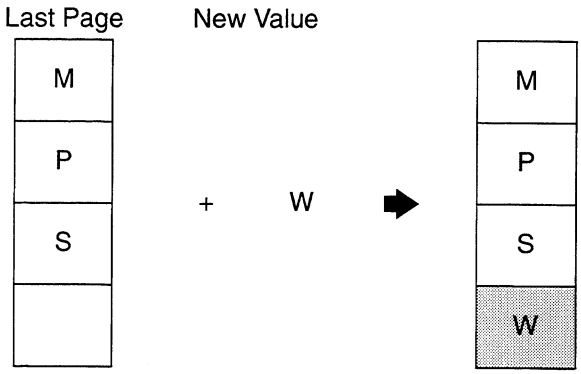
**Illus. 13.1** Adding a Value in the Distinct Values Table



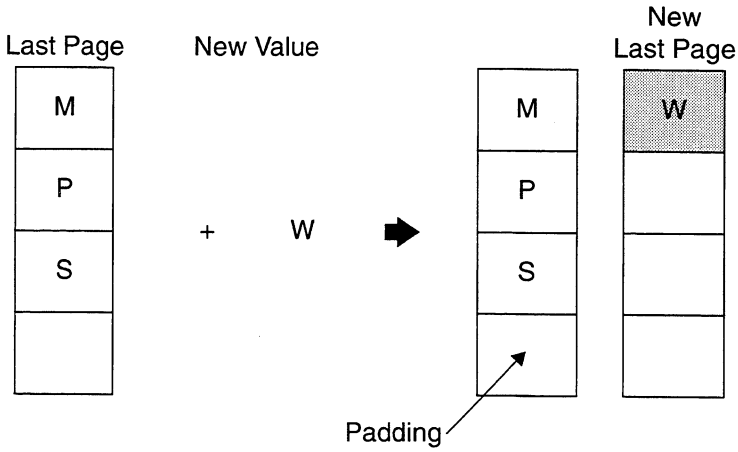

This method of inserting a new value is always used (regardless of the VALUES PADDING option chosen) for all pages except the last in the index for that item. When the new value is greater than all existing values for the key item, the new value must be added at the end of the last page for the index. In this particular situation the flag set by a VALUES PADDING command determines where the value is to be inserted. For DISABLE VALUES PADDING, the value is added to the existing last page if there is room, and to a newly created page otherwise. However, for ENABLE VALUES PADDING, the value is added to the existing last page only if enough padding will be left after adding the new value; otherwise, it is placed in a newly created page, which becomes the last page. This is shown in Illus. 13.2 on page 13-3.

**Illus. 13.2** Distinct Values Table Padding

**A.** flag=DISABLE




**B.** flag=ENABLE



What is enough padding? SYSTEM 2000 software calculates Distinct Values Table padding as an inverse function of the padding defined for a Multiple Occurrence Table entry for that item. Note that in the foldout, some of the items are declared WITH **FEW** FUTURE OCCURRENCES, WITH **SOME** FUTURE OCCURRENCES, or WITH **MANY** FUTURE OCCURRENCES, and the others default to WITH **NO** FUTURE OCCURRENCES. These declarations determine the padding in the entries of the Multiple Occurrence Table as follows:

| Declaration | Multiple Occurrence Table Padding |
|---|---|
| NO | 0% |
| FEW | 100% |
| SOME | 400% |
| MANY | 1400% |

SYSTEM 2000 software assumes that if an item has many occurrences of the same value, it has few distinct values for that item; therefore, it assigns padding for the Distinct Values Table page for that item as follows:

| Declaration | Distinct Values Table Padding |
|---|---|
| NO | 25% (3/4 filled) |
| FEW | 12.5% (7/8 filled) |
| SOME | 6.25% (15/16 filled) |
| MANY | 0 (full) |

Thus, the ENABLE VALUES PADDING command sets the flag for the specified database. After this flag is set, the creation of the last index page for each key item in that database guarantees padding equal to or greater than the amount shown in the table above, calculated as an inverse function of the Multiple Occurrence Table padding. The flag can be turned off to disallow padding, by giving the DISABLE VALUES PADDING command.

## Format

```
|DISABLE  VALUES PADDING FOR  database-identifier        (default)
|ENABLE
```

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

The Distinct Values Table padding option is stored in the database. It is disabled by default when the database is created, and it remains disabled until a user issues an ENABLE VALUES PADDING command.

## Considerations

1. The VALUES PADDING command is executed only if the corresponding database was opened with the master password or the DBA password in its COMMBLOCK. Otherwise, the program abends with SYSTEM 2000 Error Code 024. Also, if the DBA password is used and the master password did not make the ENABLE/DISABLE VALUES PADDING commands available to the DBA password, SYSTEM 2000 Error Code 024 is issued.

2. Note that ENABLE VALUES PADDING has no effect (no padding for distinct values) on the indexes for the schema items declared WITH MANY FUTURE OCCURRENCES.

3. Depending on the nature of the data, the VALUES PADDING command can increase loading time, updating time, or storage space. However, this is extremely dependent on the data definition, data values, and the order in which data are loaded initially and are added later.

4. You can issue a VALUES PADDING command in queue mode, and it is processed immediately. Any new Distinct Values Table pages created because of update commands given between QUEUE and TERMINATE are padded according to the current flag setting at TERMINATE time, in other words, when the updates are processed.

5. The Self-Contained Facility QUEST language has an equivalent command. Therefore, the padding flag set in a PLEX program can be altered by a Self-Contained Facility job and vice versa. The flag is retained in the database.

6. If a VALUES PADDING command specifies no database name or an invalid database name, the PLEX processor issues an error message for the command.

7. A VALUES PADDING command ends an insert sequence if it specifies the database of the insertions.

8. Regarding the multi-level index, only level 0 pages of the Distinct Values Table are padded. Otherwise, for example, padding of level 1 pages in the multi-level index could force the creation of level 2 pages prematurely.

Chapter 14

# FULL PASSES

*INTRODUCTION 14-1*
*Format 14-1*
*Considerations 14-1*

## INTRODUCTION

The FULL PASSES command allows or prohibits a complete pass through the Data Table of the specified database. Only the master password holder or DBA password holder can issue this command.

The PLEX where-clause can be executed in three different ways, depending on the nature of the items involved in the conditions: index processing, non-index processing, and partial-index processing. The software uses the Index for index and partial-index processing. However, non-index processing requires a complete linear search of the Data Table for values that are not indexed. This full pass through the Data Table can be time consuming. If the NO FULL PASSES command is issued for a database, where-clauses requiring a complete pass through the Data Table of that database are not executed, and return code 62 is issued. The FULL PASSES and NO FULL PASSES commands affect all jobs using that database until another user gives the opposite command.

### Format

```
[NO]  FULL PASSES FOR database-identifier
```

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

If neither a FULL PASSES nor a NO FULL PASSES command is issued, full passes are allowed by default.

### Considerations

1. The FULL PASSES and NO FULL PASSES commands require the master password or the DBA password. If the FULL PASSES commands are not available to the DBA password, SYSTEM 2000 Error Code 024 is issued.

2. There is no guarantee that the processing will be fast if full passes are not allowed. Two partial passes can cost more than one full pass.

3. If a FULL PASSES command specifies no database name or an invalid database name, the PLEX processor issues an error message for the command.

4. A FULL PASSES command does not end an insert sequence.

Chapter 15

# $TIME

*Format 15-1*
*Considerations 15-3*

The $TIME command obtains statistics on CPU time and I/O activities during the execution of PLEX commands.

## Format

```
$TIME   |ON
        |OFF
```

When you issue the $TIME ON command, the following information is displayed for each SYSTEM 2000 module (overlay) segment called until the $TIME OFF command is given:

**EXITING OVERLY** *prim, sec*

**FILE 1** *n* **DVT=** *n* **EFT** = *n*

**MOT** *n* **HT=** *n* **DT** = *n*

**TBLIO** *table I/Os* SCRIO= *scratch I/Os* TIO = *total I/Os*

**TIMING FOR OVERLAY** *prim, sec CPU time*

*prim* is a primary module number.

*n* is the number of I/O requests for individual database tables.

*sec* is a secondary module number.

*CPU time* is the CPU time in hundredths of seconds.

*table-I/Os* is the number of I/O requests to the database tables.

*scratch-I/Os* is the number of I/O requests to the scratch files or pads.

*total-I/Os* is the total number of I/O requests (not including sort/merge I/O).

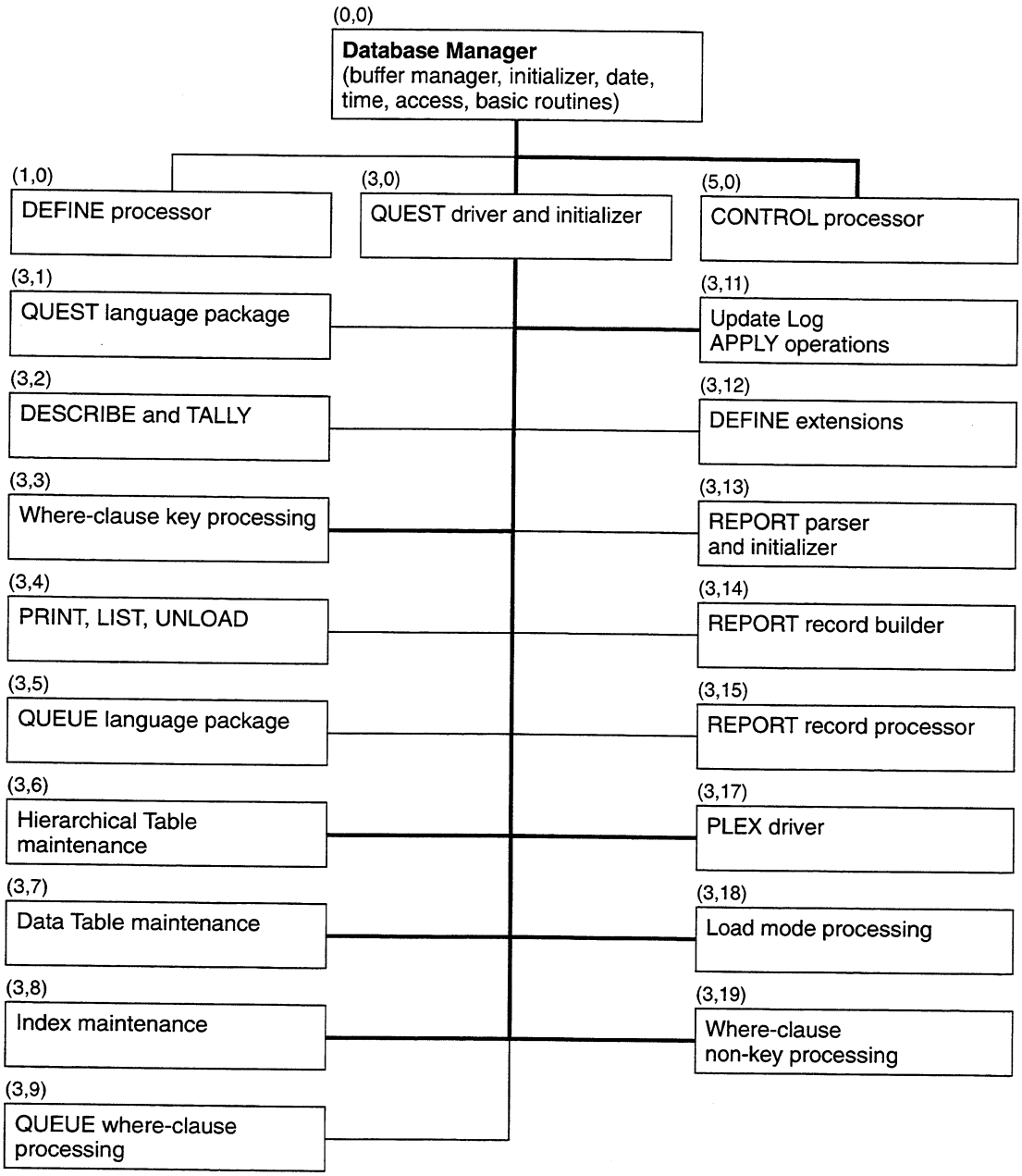
Whenever a new PLEX command is given, the opcode is displayed as follows:

**OPCODE** = *opcode*

Illus. 15.2 on page 15-4 lists the opcodes of all the PLEX commands.

The module structure of SYSTEM 2000 software is shown in Illus. 15.1. The Database Manager is the root module (0,0). The three modules (5,0), (1,0), and (3,0) are the primary modules that can be called by the root. The rest are the secondary modules of (3,0). You can combine modules into segments through use of the PRELNK macro (see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*).

**Illus. 15.1** SYSTEM 2000 Modules



## Considerations

1. If a command includes either the /HOLD option or the /EXCLUSIVE option, 500 is subtracted from the opcode for the command, and the result is displayed in the field for *opcode*. For example, the opcode for the OPEN command is 60, but 440- is displayed for OPEN/EXCL. Also, if UPDATE LOG is specified in a CLEAR command, 500 is subtracted from the opcodes for the CLEAR commands.

2. The OPCODE line is displayed before entering module (3,17). EXITING OVERLY, DB I/O, TBLIO, and TIMING lines are displayed for the previous module when the next overlay is entered. Therefore, entering (3,17) displays statistics for the previous use of (3,17). Thus, the OPCODE line is one set of statistics ahead of where you might logically expect it to be. This is illustrated with examples in Illus. 15.3 on page 15-6.

3. To get a cumulative statistic for a queue or load session, it is best to turn timing off before QUEUE, and to turn it on and off just before and after TERMINATE as shown here.

```
        .
        .
        .
$TIME OFF.
        .
        .
        .
QUEUE.
        .
        .
        .
$TIME ON.

TERMINATE.

$TIME OFF.
        .
        .
        .
```

4. Some installations might use a flat system rather than a modular structure for SYSTEM 2000 software, so that no module segments need be loaded during execution. In this situation, output from the $TIME ON command will still show the timing for each module segment at the time it would have been loaded.

**Illus. 15.2** Opcodes for PLEX Commands

| PLEX Command | Opcode Decimal | Hex |
|---|---|---|
| GET1 *SSR* FIRST | 00 | 00 |
| GET1 *SSR* LAST | 01 | 01 |
| GET1 *SSR* S2KCOUNT | 03 | 03 |
| GET1 *SSR* NEXT | 04 | 04 |
| GET1 *SSR* WHERE *where-clause* | 05 | 05 |
| GET1 *SSR* DYNAMICALLY | 07 | 07 |
| GET1 *SSR* PRESENT | 08 | 08 |
| LOCATE *SSR* WHERE *where-clause* | 10 | 0A |
| GETP *SSR* WHERE *single-key-condition* | 11 | 0B |
| LOCATE *SSR* DYNAMICALLY | 12 | 0C |
| GET *SSR* FIRST | 20 | 14 |
| GET *SSR* LAST | 21 | 15 |
| GET *SSR* PREVIOUS | 22 | 16 |
| GET *SSR* S2KCOUNT | 23 | 17 |
| GET *SSR* NEXT | 24 | 18 |
| GET *SSR* PRESENT | 25 | 19 |
| GETD *SSR* FIRST | 30 | 1E |
| GETD *SSR* LAST | 31 | 1F |
| GETD *SSR* S2KCOUNT | 33 | 21 |
| GETD *SSR* NEXT | 34 | 22 |
| GETD *SSR* PRESENT | 36 | 24 |
| GETA *SSR* | 40 | 28 |
| Reserved for future use | 46 | 2E |
| Reserved for future use | 47 | 2F |
| Reserved for future use | 48 | 30 |
| INSERT *SSR* | 50 | 32 |
| INSERT *SSR* AFTER | 50 | 32 |
| INSERT *SSR* BEFORE | 51 | 33 |
| MOVE TREE *SSR* | 55 | 37 |
| MOVE TREE *SSR* AFTER | 55 | 37 |
| MOVE TREE *SSR* BEFORE | 56 | 38 |
| OPEN *database* | 60 | 3C |
| LINK | 63 | 3F |
| OPENR *database* | 64 | 40 |
| LOAD *database* | 65 | 41 |
| FOR | 67 | 43 |
| REMOVE *SSR* | 70 | 46 |
| MODIFY *SSR* | 80 | 50 |
| REMOVE TREE *SSR* | 90 | 5A |
| START S2K (*n*) | 91 | 5B |
| STOP S2K | 92 | 5C |
| ORDER BY *ordering-clause* | 93 | 5D |
| CLEAR* | 94 | 5E |
| CLEAR AUTOMATICALLY** | 95 | 5F |

*continued on next page*

**Illus. 15.2** *continued*

| PLEX Command | Decimal | Hex |
|---|---|---|
| CANCEL QUEUE | 96 | 60 |
| CLOSE *database* | 97 | 61 |
| QUEUE | 98 | 62 |
| TERMINATE | 99 | 63 |
| APPLY | 201 | C9 |
| KEEP | 202 | CA |
| Reserved for future use | 203 | CB |
| Reserved for future use | 204 | CC |
| RELEASE | 205 | CD |
| SUSPEND | 206 | CE |
| FULL PASSES/NO FULL PASSES | 207 | CF |
| ENABLE/DISABLE ROLLBACK | 208 | D0 |
| FRAME | 209 | D1 |
| END FRAME | 210 | D2 |
| DROP HOLD | 211 | D3 |
| RESET ROLLBACK | 212 | D4 |
| SAVE | 213 | D5 |
| RESTORE | 214 | D6 |
| COPYS | 215 | D7 |
| COPYL | 216 | D8 |
| COMMIT | 217 | D9 |
| ROLLBACK | 218 | DA |
| ENABLE/DISABLE VALUES PADDING | 220 | DC |
| ENABLE/DISABLE EXITS | 240 | F0 |
| YEARCUTOFF | 260 | 104 |

```
*   CLEAR UPDATE LOG = 94 - 500 = 406-
**  CLEAR UPDATE LOG AUTOMATICALLY = 95 - 500 = 405-
```

**Illus. 15.3** Sample PLEX Timing Output

```
OPCODE= 005                (GET1 . . . WHERE . . . from user program)
EXITING OVERLY=3,17
(Database I/Os)
TBLIO= . . .
TIMING FOR OVERLAY 3,17. . .
```

---

```
EXITING OVERLY= 3,17
(Database I/Os)
TBLIO= . . .                                (enters 3,3)
TIMING FOR OVERLAY 3,17. . .
EXITING OVERLY= 3,03                        (enters 3,17)
(Database I/Os)                                        (total time for
TBLIO= . . .                                (ends)      GET1...WHERE)
TIMING FOR OVERLAY 3,3. . .
OPCODE= 080                (MODIFY from user program)
EXITING OVERLY= 3,17
(Database I/Os)
TBLIO= . . .                                 (enters 3,17)
TIMING FOR OVERLAY 3,17. . .                 (begins)
```

---

```
EXITING OVERLY= 3,17
(Database I/Os)
TBLIO= . . .                                 (enters 3,7)
TIMING FOR OVERLAY 3,17. . .
EXITING OVERLY= 3,07
(Database I/Os)
TBLIO= . . .                                 (enters 3,8)
TIMING FOR OVERLAY 3,7. . .
EXITING OVERLY= 3,08                               (total time for
(Database I/Os)                                     MODIFY)
TBLIO= . . .                                 (enters 3,17)
TIMING FOR OVERLAY 3,8. . .                  (ends)
OPCODE= 034                 (GETD . . . NEXT from user program)
EXITING OVERLY= 3,17
(Database I/Os)
TBLIO= . . .                                 (enters 3,17)
TIMING FOR OVERLAY 3,17. . .                 (begins)
```

---

```
OPCODE= 050                 (INSERT from user program)
EXITING OVERLY= 3,17
(Database I/Os)                                          (total time for
TBLIO= . . .                                              GETD)
TIMING FOR OVERLAY 3,17. . .
```

---

*continued on next page*

**Illus. 15.3** *continued*

```
OPCODE= 080                          (MODIFY from user program)
EXITING OVERLY= 3,17
(Database I/Os)                      (INSERT)
TBLIO= . . .
TIMING FOR OVERLAY 3,17
```

---

```
EXITING OVERLY= 3,17
(Database I/Os)
TBLIO= . . .
TIMING FOR OVERLAY 3,17. . .
EXITING OVERLY= 3,06                                  (forced
(Database I/Os)                                       INSERT
TBLIO= . . .                                          time)
TIMING FOR OVERLAY 3,6. . .
EXITING OVERLY= 3,07
(Database I/Os)
TBLIO=. . .
TIMING FOR OVERLAY 3,7. . .
EXITING OVERLY= 3,08
(Database I/Os)
TBLIO= . . .
TIMING FOR OVERLAY 3,8. . .
```

---

```
EXITING OVERLY= 3,17
(Database I/Os)
TBLIO= . . .
TIMING FOR OVERLAY 3,17. . .
EXITING OVERLY= 3,07
(Database I/Os)
TBLIO= . . .
TIMING FOR OVERLAY 3,17. . .
OPCODE= 080                      (MODIFY from user program)
EXITING OVERLY= 3,17
(Database I/Os)
TBLIO= . . .
TIMING FOR OVERLAY 3,17. . .
```

---

Chapter 16

# ENABLE ROUTINE and DISABLE ROUTINE

*INTRODUCTION 16-1*
*Format 16-1*
*Considerations 16-2*

## INTRODUCTION

Using the optional user exits capability, systems personnel at a given site can implement exits from SYSTEM 2000 software that are permitted at predefined points. Each implemented exit allows the execution of code developed by systems personnel. The ENABLE ROUTINE command specifies the exits that the program will use. The DISABLE ROUTINE command specifies the enabled exits that the program will no longer use. These commands affect only the program that issues them.

Since the exits are implemented with code developed at each site, the requirements of that code determine the information that must be provided in an ENABLE ROUTINE or DISABLE ROUTINE command. For more information about the nature and implementation of user exits, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*

### Format

```
|DISABLE ROUTINE field-name FOR EXITS exit-list
|ENABLE
```

*field-name* is the variable name of an 80-byte field in which information was placed for use by the exit(s) listed. This field can be an array of ten fields of eight bytes each.

*exit-list* |*range*
|*exit-number* [ , *exit-list*]

Synonym: EXITS = EXIT

*range* *lower-bound* **THROUGH** *upper-bound*

*lower-bound* *exit-number* such that *lower-bound* is greater than zero and less than *upper-bound*.

*upper-bound* *exit-number* such that *upper-bound* is greater than *lower-bound* and less than or equal to 63, depending on how EXIT01 is coded.

*exit-number* is a constant representing the number of an exit to be enabled or disabled.

## Considerations

1. Check the return code after you issue an ENABLE ROUTINE or DISABLE ROUTINE command. This return code is placed in S2KRTC, a variable in the S2KDUM control block.

2. If the exit interface routine S2EXIT was not loaded, the software ignores an ENABLE ROUTINE or DISABLE ROUTINE command, and return code 901 results.

3. If the exit EXIT01 is not enabled, an ENABLE ROUTINE or DISABLE ROUTINE command is ignored, and return code 902 results.

---

Abbreviation: THROUGH = THRU

Chapter 17

# Termination

*CLOSE 17-1*
*Format 17-1*
*Example of Usage 17-2*
*Considerations 17-3*

*STOP S2K 17-4*
*Format 17-4*
*Example of Usage 17-5*
*Considerations 17-5*

This chapter describes the commands that are needed to terminate a PLEX program. The first section describes the CLOSE command, which marks the end of using a database in a program. The second section describes the STOP S2K command, which marks the end of a program's use of SYSTEM 2000 software.

## CLOSE

The CLOSE command makes a database unavailable to the program.

When your program completes its processing of a database, give a CLOSE command to make the database unavailable to it. CLOSE ensures that an updated database will not be damaged. It clears updated database pages to disk before the database becomes unavailable. Also, the CLOSE command cancels the stacks and Locate Files for a database.

Up to 63 databases can be available to a program at one time. If your program already has 63 databases open, you must give a CLOSE command for one of those databases before you can open an additional database.

### Format

---

**CLOSE** *database-identifier*

---

*database-identifier* is the name of a SYSTEM 2000 database or the name of a COMMBLOCK.

## Example of Usage

This example illustrates using the CLOSE command.

```
OPEN A.
(Check the return code and damage flag.)
      :
      :
OPEN B.
(Check the return code and damage flag.)
      :
      :
OPEN C.
(Check the return code and damage flag.)
      :
      :

(Updates in immediate mode for
database B)

CLOSE B.                          (ends an insert sequence for
                                  database B; clears updated
                                  pages to disk; makes database
                                  B unavailable to the program)

QUEUE.
      :
      :
(Updates in queue mode for        (performs the pending updates
database C)                       for database C)

TERMINATE.

CLOSE C.                          (given after TERMINATE; clears
                                  updated pages to disk; makes
                                  database C unavailable to
                                  the program)
LOAD A.
      :
      :
(INSERT commands in load mode     (performs the pending insertions
for database A)                   in database A)

TERMINATE.

CLOSE A.                          (given after TERMINATE; clears
                                  updated pages to disk; makes
                                  database A unavailable to
                                  the program)
```

## Considerations

1. Check the return code after you give the CLOSE command. The return code is placed in the second COMMBLOCK field for the database to which the command refers.

2. Before you issue STOP S2K, give the CLOSE command for each database updated by the program. In a Multi-User environment, give the STOP S2K command after you give CLOSE commands for all databases available to the program. In this situation, STOP S2K causes the user-active tables for the program to be cleared and causes a user termination record to be placed in the Multi-User Accounting Log.

3. In immediate mode, the CLOSE command ends an insert sequence for the database to which it refers. Updated database pages are then cleared to disk. However, if you give the CLOSE command in queue mode or load mode, the software cancels updates pending during the queue or load session. Furthermore, the stack in use is cleared, and return code 27 results.

4. The software ignores a CLOSE command given for a database that is not open, and return code 26 results.

5. If a CLOSE command specifies no database name or an invalid database name, the PLEX processor assumes it to be a programming language statement and, therefore, generates no code for the command and issues no error message. The error is caught by the compiler.

6. A CLOSE command abnormally terminates any rollback frame in effect. A user frame established by a FRAME command must be terminated by an END FRAME command before CLOSE is issued.

7. The CLOSE command causes a user synchpoint for Coordinated Recovery if rollback is enabled. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for a discussion of Coordinated Recovery.

# STOP S2K

The STOP S2K command signals the end of PLEX processing in your PLEX program..

For Multi-User and single-user jobs in either a batch mode or teleprocessing (TP) environment, the STOP S2K command causes the following actions to occur:

- All databases opened to the job are closed.
- All active synchpoints are closed.
- A STOP S2K command immediately after an insert sequence ends the insert sequence.
- A STOP S2K command issued in queue mode behaves like CANCEL QUEUE. That is, updates pending since the QUEUE command are canceled.
- A STOP S2K command issued in load mode cancels insertions to be made in the Index. Insertions already made in the Hierarchical Table and Data Table are backed out when the database is next opened.

The standard should be to terminate PLEX program processing by giving a STOP S2K command. A PLEX program should give the STOP S2K command after it gives the rest of its PLEX commands, including CLOSE commands for all databases updated by the program.

When the STOP S2K command is given, the Multi-User user-active tables are cleared for the PLEX user. In addition, the software frees all Locate Files and scratch files assigned to that PLEX program.

## Format

```
STOP S2K
```

## Example of Usage

This example illustrates using STOP S2K in a Multi-User environment.

```
      .
      .
      .
OPEN A.
      .
      .
      .
OPEN B.
      .
      .
      .
OPEN C.
      .
      .
      .
(Updates to databases A, B, C)
      .
      .
      .
CLOSE C.
      .
      .
      .
CLOSE A.
      .
      .
      .
CLOSE B.
      .
      .
      .
STOP S2K.                          (given after the last
                                   CLOSE command)
```

## Considerations

1. For a Multi-User environment, the STOP S2K command clears the user-active tables and forces a user termination record to be written on the Multi-User Accounting Log.

2. For the TP user, any SYSTEM 2000 PLEX command (except START S2K) issued after a STOP S2K command is rejected with return code 44. START S2K issued after STOP S2K produces unpredictable results. Return code 44 is not issued to batch jobs.

3. If STOP S2K is not issued in a PLEX program, the databases are closed, and the user becomes inactive at the following point in time:

   - for a batch job, when SYSTEM 2000 software intercepts job termination
   - for a TP job, when the SYSTEM 2000 software interface intercepts the TP transaction to return to the TP monitor.

4. If STOP S2K is inadvertently executed due to bad branching or faulty program logic, processing completes normally as described for STOP S2K. You will have no indication that the PLEX program did not run to logical completion.

5. The STOP S2K command causes a user synchpoint for Coordinated Recovery if rollback is enabled. See *SYSTEM 2000 CONTROL Language, Version 12, First Edition* for a discussion of Coordinated Recovery.

Chapter 18

# YEARCUTOFF

*INTRODUCTION 18-1*
*Format 18-1*
*Examples of Syntax 18-2*
*Considerations 18-2*

## INTRODUCTION

For dates that have two-digit years (MM/DD/YY, for example), YEARCUTOFF allows you to indicate to SYSTEM 2000 in which century the dates belong. The currently accepted range for dates in SYSTEM 2000 is October 15, 1582, to December 31, 2499.

You can use YEARCUTOFF to specify a value *nnnn* to be used as the first year of a 100-year span. Without YEARCUTOFF the default is 1900 (until January 1, 2000, when the default becomes 2000).

If the default value (1900) is in effect, the 100-year span begins with 1900 and ends with 1999. That is, any dates input with a two-digit year value are assigned a prefix of 19. For example, the value 92 refers to the year 1992.

If you want to use dates that occur in two centuries, specify a YEARCUTOFF command to accommodate both centuries. For example, if you specify YEARCUTOFF=1950, any two-digit value between 50 and 99 inclusive refers to the first half of the 100-year span, which is in the 1900s. Any two-digit value between 00 and 49 inclusive refers to the second half of the 100-year span, which is in the 2000s.

### Format

```
YEARCUTOFF |=  [(]|nnnn      [)].
           |IS [(]|variable  [)]
           |EQ [(]|RESET     [)]
```

*nnnn* is a four-digit year between 1500 and 2400.

*variable* is a variable containing a full-word binary integer value between 1500 and 2400.

## Examples of Syntax

**Example 1**

```
YEARCUTOFF =  1900.
```

This is the same as the default until January 1, 2000, when the default century becomes 2000.

**Example 2**

```
YEARCUTOFF EQ (2143).
```

This causes any two-digit years in the range 43 through 99 to be assigned to 2143 through 2199, and two-digit years in the range 00 through 42 to be assigned to 2200 through 2242.

**Example 3**

```
YEARCUTOFF IS YRCT.
```

This uses the contents of the word YRCT as a binary integer value to modify the current setting of YEARCUTOFF.

**Example 4**

```
YEARCUTOFF IS (RESET).
```

This resets YEARCUTOFF to the value given in the YEARCUTOFF parameter at startup time, if given, or to the current default if the YEARCUTOFF parameter was not given.

## Considerations

1. YEARCUTOFF has no effect on four-digit years.

2. The system default for the century is 1900 (until January 1, 2000, when it will change to 2000), which is used if dates are entered as two-digit years. If the parameter YEARCUTOFF is used, it takes precedence over the system default. If a PLEX command is issued to change YEARCUTOFF, the year specified takes precedence over both the parametric entry and the system default.

   The hierarchy of precedence, beginning at the top, is as follows:

   - Date given as four-digit year
   - YEARCUTOFF PLEX command
   - YEARCUTOFF execution parameter
   - System default (current century).

3. The return code should be checked after the YEARCUTOFF command is issued. Since YEARCUTOFF does not refer to a particular database, its return code is not placed in the COMMBLOCK. Instead, the return code is placed in S2KRTC, a

variable in the S2KDUM control block.

There are two possible return codes for YEARCUTOFF:

- 0 indicates a successfully executed command
- 130 indicates that the date was unacceptable.

4. In a Multi-User environment, the YEARCUTOFF command affects only the user who specified it. The YEARCUTOFF parameter affects all users of Multi-User.

5. This command affects dates only if the FORM option for a SUBSCHEMA RECORD has a date format with a two-digit year.

6. Except for where-clause processing, this command affects only updates and not retrievals. If two-digit years are used in where clauses, they are converted to four-digit years, as explained above. With this in mind, consider the following:

   In a PLEX program that retrieves dates with a FORM option that includes a two-digit year, the year 1732 is retrieved as 32. Suppose the current value for YEARCUTOFF is 2000, and the PLEX program retrieves a record that contains information about GEORGE WASHINGTON to correct the spelling of his name, which had been entered as WASHINGTNO. The record also contains his year of birth, 1732, which is retrieved as 32 due to the FORM option in the SUBSCHEMA RECORD. When the field that contains WASHINGTNO is changed to be WASHINGTON and the record is then modified in the database, the year of birth is changed to 2032.

   To avoid this situation, you can:

   - Use only FORM options that have four-digit years.
   - Use inclusion lists with your MODIFY commands.

Chapter 19

# COBOL PLEX Processing

*COBOL PLEX PROCESSOR DIRECTIVES 19-2*
- *$BLOCKS Directive 19-2*
- *$CICS Directive 19-7*
- *$COPY Directive 19-7*
- *END PROCEDURE Directive 19-8*
- *$NAME Directive 19-8*
- *$S2KDUM Directive 19-9*

*COBOL PLEX PROCESSOR EXECUTION OPTIONS 19-10*
- *LANG Parameter 19-10*
- *CICS Execution Option 19-10*
- *COPY and NOCOPY Execution Option 19-11*
- *DECK and NODECK Execution Option 19-11*
- *ECHO and NOECHO Execution Option 19-11*
- *ERRORS and NOERRORS Execution Option 19-12*
- *ERRSYM Execution Option 19-12*
- *EXIT Execution Option 19-12*
  - *Modifying, replacing, and adding records 19-13*
  - *Diagnostic messages 19-14*
- *FORM and NOFORM Execution Option 19-16*
- *GEN and NOGEN Execution Option 19-16*
- *ITEM Execution Option 19-17*
- *LINECNT Execution Option 19-17*
- *LOAD and NOLOAD Execution Option 19-17*
- *MEMORY Execution Option 19-17*
- *QUOTE Execution Option 19-18*
- *SEQID Execution Option 19-18*
- *SORMGIN Execution Option 19-19*
- *SOURCE and NOSOURCE Execution Option 19-19*
- *SSR Execution Option 19-20*
- *SYSSEP Execution Option 19-20*
- *TERM and NOTERM Execution Option 19-20*
- *TRANS Execution Option 19-21*

This chapter describes the syntax and results of each COBOL PLEX processor directive and execution option. See the FORTRAN, PL/I, and Assembler appendixes for descriptions of processor directives and execution options for those languages.

# COBOL PLEX PROCESSOR DIRECTIVES

You code the COBOL PLEX processor directives like PLEX commands in the source program. They are signals to the COBOL PLEX processor.

## $BLOCKS Directive

### Format

```
$BLOCKS .
```

Each COBOL PLEX program must include the $BLOCKS directive in the WORKING-STORAGE SECTION. In response to $BLOCKS, the PLEX processor generates descriptions of control blocks for retrieval or update commands containing item lists or requiring special information to be passed to SYSTEM 2000 software during execution. In addition, if the $S2KDUM directive does not appear in the LINKAGE SECTION, the PLEX processor generates the description of S2KDUM with initial values in response to $BLOCKS. (See Illus. 19.3 on page 19-6 and Illus. 19.2 on page 19-4 for a description of S2KDUM.)

The PLEX processor also generates PLXCOMM after S2KDUM in the program if you specified the $CICS ON directive or the CICS execution option.

$BLOCKS before COMMBLOCK and subschema definition causes PLEX variables to be generated before S2KDUM.

$BLOCKS after COMMBLOCK and subschema definition causes PLEX processor variables to be generated after S2KDUM (and after optional PLXCOMM).

END SCHEMAS serves the same purpose and can be used in place of $BLOCKS. An error message is generated if you have both. With END SCHEMAS, PLEX variables are always generated before S2KDUM. For more information about CICS programs, see *SYSTEM 2000 Interface to CICS (Command Level), Version 12, Second Edition.*

**Illus. 19.1** Sample S2KDUM Generated by the PLEX Processor for $BLOCKS

```
*      $BLOCKS.
 01    S2KDUM.
       02 S2KDU1     PIC S9(9) USAGE COMP VALUE +0.
       02 S2KRTC     PIC S9(9) USAGE COMP VALUE +0.
       02 S2KCOUNT   PIC S9(9) USAGE COMP VALUE +0.
       02 S2KDU2     PIC S9(9) USAGE COMP VALUE +0.
       02 S2KDU3     PIC S9(9) USAGE COMP VALUE +0.
       02 FILLER     PIC S9(9) USAGE COMP VALUE +2.
       02 S2KDU4     PIC S9(9) USAGE COMP VALUE +0.
       02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
       02 FILLER     PIC X(8) VALUE 'TEST'.
       02 FILLER     PIC X(4) VALUE '0290'.
       02 FILLER     PIC X(8) VALUE '79270   '.
       02 FILLER     PIC X(8) Value '0040098 '.
       02 FILLER     PIC X(4) VALUE '    '.
       02 S2KDU5     PIC S9(9) USAGE COMP VALUE +0.
       02 S2KDU6     PIC S9(9) USAGE COMP VALUE -1.
       02 S2KDU7     PIC X(4) VALUE '    '.
       02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
       02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
       02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
       02 S2KDU8     PIC S9(9) USAGE COMP VALUE +0.
       02 S2KDU9     PIC S9(9) USAGE COMP VALUE +0.
       02 S2KSRT     PIC S9(9) USAGE COMP VALUE +0.
       02 S2KDU11    PIC S9(9) USAGE COMP VALUE +0.
       02 S2KDU12    PIC S9(9) USAGE COMP VALUE +0.
       02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
       02 S2KLKH PIC S9(9) USAGE COMP OCCURS 10 TIMES.
```

**Illus. 19.2** Description of S2KDUM Control Block Items

| Offset HEX | Offset DEC | Name | Length/ Alignment | Type/ Value | Field Description |
|---|---|---|---|---|---|
| 0 | 0 | S2KDU1 | 4 | Binary | SYSTEM 2000 operation code; set by the expanded PLEX code at execution time. |
| 4 | 4 | S2KRTC | 4 | Binary | SYSTEM 2000 return code; set by a PLEX command execution. |
| 8 | 8 | S2KCOUNT | 4 | Binary | Value of S2KCOUNT; set by the PLEX program. |
| C | 12 | S2KDU2 | 4 | Binary | Stack or Locate File number; set by expanded code at run time. |
| 10 | 16 | S2KDU3 | 4 | Binary | Maximum number for Locate File; set at run time by START S2K expanded code. Also, the number of the target stack or Locate File referred to in a COPYS or COPYL command; set at run time. |
| 14 | 20 | FILLER | 4 | Binary<br>1<br>2<br>3<br>4<br>5 | PLEX program type; set by PLEX processor.<br>FORTRAN<br>COBOL<br>PL/I; optimizing or checkout.<br>PL/I (F)<br>ASSEMBLER |
| 18 | 24 | S2KDU4 | 4 | Binary<br>0<br>1 | SYSTEM 2000 Multi-User indicator set at run time by the START S2K expanded code.<br>single-user<br>Multi-User |
| 1C | 28 | FILLER | 4 | Alpha-numeric | If S2KDU4 is neither fullword binary zero or one, S2KDU4 concatenated with this field contains the name of a special module to be used (left-justified and blank-filled). |
| 20 | 32 | FILLER | 8 | Alpha-numeric | Name of program, left-justified and blank-filled; set by the PLEX processor as specified in the $NAME directive. |
| 28 | 40 | FILLER | 4 | Alpha-numeric | Release of the SYSTEM 2000 PLEX processor used; set by the PLEX processor. If changes are made to the Program Service Processor, this field identifies the PLEX processor release that is used. |
| 2C | 44 | FILLER | 8 | Alpha-numeric | Date of PLEX processor run, YYDDD (left-justified and blank-filled); set by the PLEX processor. |
| 34 | 52 | FILLER | 8 | Alpha-numeric | Time of PLEX processor run; HHMMSSS (left-justified and blank-filled); set by the PLEX processor. |
| 3C | 60 | FILLER | 4 | Alpha-numeric | Release number of SYSTEM 2000 software; set at execution time by the software. |

*continued on next page*

**Illus. 19.2** *continued*

| Offset HEX | Offset DEC | Name | Length/ Alignment | Type/ Value | Field Description |
|---|---|---|---|---|---|
| 40 | 64 | S2KDU5 | 4 | Binary | Reserved for internal use. |
| 44 | 68 | S2KDU6 | 4 | Binary | Reserved for internal use. |
| 48 | 72 | S2KDU7 | 4 | Alpha | Reserved for internal use. |
| 4C | 76 | FILLER | 4 | Binary | Address of run-time interface (S2KPLR). |
| 50 | 80 | FILLER | 4 | Binary | Address of SYSTEM 2000 software (S2KPLI, MUPLINT). |
| 54 | 84 | FILLER | 4 | Binary | Address of working storage; set by run-time interface. This address can be used by installation run-time exit routines S2KEX1 or S2KEX2. |
| 58 | 88 | S2KDU8 | 4 | Binary<br>0<br>1 | Snap request indicator; set by expanded code.<br>Off<br>On: SYSTEM 2000 software will provide snapshot data. |
| 5 | 92 | S2KDU9 | 4 | Binary<br><br>0<br>1 | Timing request indicator; set by expanded code.<br>Off<br>On |
| 60 | 96 | S2KSRT | 4 | Binary | No-sort request indicator; set by the PLEX program.<br>0 = Sort normally.<br>1 = Do not sort. |
| 64 | 100 | | 4 | Binary | Reserved for internal use. |
| 68 | 104 | | 4 | Binary | Reserved for internal use. |
| 6C | 108 | | 1 | Bit<br>.... ...0<br>.... ...1 | Switches used at run time.<br>Link history was not modified.<br>Link history was modified. |
| 6D | 109 | | 3 | | Reserved for internal use. |
| 70 | 112 | S2KLKH | 40 | Binary | Ten full words representing LINK0 through LINK9 return codes; set by linked retrievals; initialized by the PLEX processor to ten words of binary minus one. |

**Illus. 19.3** Sample PLXCOMM Generated by the PLEX Processor for $BLOCKS

```
       01  PLXCOMM.
           02 FILLER PIC X(4) VALUE  'PLEX'.
           02 FILLER PIC S9(9) USAGE COMP VALUE +0.
           02 FILLER PIC S9(9) USAGE COMP VALUE +0.
           02 FILLER PIC S9(9) USAGE COMP VALUE +0.
           02 FILLER PIC S9(9) USAGE COMP VALUE +0.
           02 FILLER PIC S9(9) USAGE COMP VALUE +0.
           02 FILLER PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC S9(9) USAGE COMP VALUE +0.
         02 FILLER     PIC X(392).
         02 PLXWK1     PIC X(8).
```

Total length of PLXCOMM is 496 bytes. It is only generated for CICS.

## $CICS Directive

### Format

```
$CICS  |ON .
       |OFF
```

To use the CICS interface, you must either specify the CICS execution option in the job's JCL or issue the $CICS ON directive.

When the PLEX processor encounters the $CICS ON directive, it generates command-level code suitable for CICS processing. If LANG=COB2, a call is made directly to PLXFRMT. Otherwise, a call is made to PLXBLD. Use of the CALL statement gives better CPU performance versus the CICS LINK command.

Beginning with Release 12.0, the $CICS OFF command is still accepted, but it has no effect on generated code.

## $COPY Directive

### Format

```
$COPY member .
```

*member* is the name of a member of the SYSLIB copy library.

For COBOL, the $COPY directive instructs the PLEX processor to copy source statements from a member of a prepared library into the PLEX program in the DATA DIVISION or PROCEDURE DIVISION. $COPY specifies the member of the SYSLIB that provides the statements, and the placement of $COPY indicates where the copied statements go.

The COPY execution option must be specified in the JCL for a program that issues the $COPY directive. Only entire commands, COMMBLOCKs, or subschema records can be copied. For example, a where-clause or a portion of a subschema record cannot be copied by itself. The PLEX processor supports up to ten levels of $COPY nesting. If a $COPY directive is included in a member that is copied, the PLEX processor obtains the additional commands from the indicated member. If $COPY files are concatenated, the first must have the largest block size.

A special zap is available that requires all subschema record declarations to be copied from the $COPY library. (Contact your SYSTEM 2000 Software Representative.)

## END PROCEDURE Directive

### Format

```
END PROCEDURE .
```

**Note**: The END PROCEDURE directive has no initial dollar sign.

If you use the FORM execution option, you must issue END PROCEDURE in order to separate two programs that are submitted together but are to be individually processed by the PLEX processor. This directive causes the PLEX processor to reinitialize its internal settings. If you do not issue END PROCEDURE in this situation, the PLEX processor issues error messages when any of the following occurs:

- the $BLOCKS directive appears twice
- two COMMBLOCKs or subschema records have the same name
- a command refers to an unqualified subschema item name that is found in more than one subschema record.

On the other hand, if you specify the NOFORM (default) execution option, you must omit END PROCEDURE between two programs because the COBOL IDENTIFICATION DIVISION statement of the second program serves the same function for the PLEX processor.

If you submit only one program to the PLEX processor, you can omit END PROCEDURE if you use the FORM execution option.

## $NAME Directive

### Format

```
$NAME name .
```

*name* is an eight-character name, left-justified and blank-filled.

The $NAME directive instructs the PLEX processor to initialize a FILLER field in the S2KDUM control block to the specified name. Also, this name is printed at the top of each page of output from the PLEX processor. If you do not include $NAME in the program, the program name is extracted from the PROGRAM-ID statement.

## $S2KDUM Directive

### Format

```
$S2KDUM .
```

When a program receives the S2KDUM control block as a parameter, you must issue the $S2KDUM directive in the LINKAGE SECTION of the program. In response to the directive, the PLEX processor generates the description of S2KDUM, the control block used by SYSTEM 2000 software for the execution of each PLEX command. No initial values are specified for the items in S2KDUM.

If you omit $S2KDUM from the program or if it appears in the COBOL WORKING-STORAGE SECTION, S2KDUM is generated and initialized in response to the $BLOCKS directive; see **$BLOCKS Directive** on page 19-2 for a discussion of $BLOCKS. See Illus. 19.3 on page 19-6 and Illus. 19.2 on page 19-4 for an explanation of the items in S2KDUM.

In response to the $S2KDUM directive in the LINKAGE SECTION, the PLEX processor generates the description of S2KDUM, beginning with an 01-level group item, without initial values for the elementary items. If this is a CICS program, PLXCOMM is generated after S2KDUM. Immediately after the $S2KDUM directive, you must include descriptions of COMMBLOCKs and subschema records without intervening item descriptions. For more information about CICS programs, see *SYSTEM 2000 Interface to CICS (Command Level), Version 12, Second Edition*.

# COBOL PLEX PROCESSOR EXECUTION OPTIONS

You set the COBOL PLEX processor execution options with the PARM parameter on the EXEC statement. For example, if the name of the job step for the PLEX processor is PRECOM, the parameter list for the job step appears as follows:

```
//PRECOM  EXEC..., PARM ='parameter-list'
```

Illus. 20.5 on page 20-26 shows a complete example of JCL for a single-user PLEX program.

## LANG Parameter

**Format**

```
LANG = COB2
```

The COB2 option is used to identify COBOL 11 program source, and it serves two purposes. The first purpose is to generate IF statements in response to a $FOR directive. Because the structure of IF and ELSE is unique in a COBOL II environment, the COB2 option is used to generate statements in the proper form.

The second purpose of the COB2 option is to identify the use of COBOL II-only functionality in the newer versions of CICS. With the COB2 option, a dynamic call directly to PLXFRMT is generated in response to SYSTEM 2000 command syntax. Without COB2, a static call to PLXPBLD is generated. Eliminating the call to PLXPBLD decreases processing time.

## CICS Execution Option

**Format**

```
CICS
```

When you are doing compiles for CICS Command-Level PLEX programs, you can use either the CICS execution option or the $CICS ON and $CICS OFF processor directives. Either method generates the code that is necessary for communication with the CICS Command-Level Interface.

If you are using COBOL II, see the LANG parameter, above.

## COPY and NOCOPY Execution Option

**Format**

```
|NOCOPY        (default)
|COPY
```

If a PLEX program uses the $COPY directive, you must specify the COPY execution option. If you do not specify this execution option (or if you specify the NOCOPY execution option), the PLEX processor issues an error message when it encounters a $COPY directive.

The COPY execution option does not force the use of $COPY. A special zap is available that requires all subschema record declarations to be copied from the $COPY library. (Contact your SYSTEM 2000 Software Representative.)

## DECK and NODECK Execution Option

**Format**

```
|NODECK        (default)
|DECK
```

The DECK option instructs the PLEX processor to punch the expanded source on SYSPUNCH.

## ECHO and NOECHO Execution Option

**Format**

```
|ECHO          (default)
|NOECHO
```

ECHO specifies that source PLEX commands and PLEX processor directives should be included as comments in the input (SYSPUNCH and SYSGO) to the COBOL compiler. NOECHO specifies that source commands and directives should be omitted in the input to the COBOL compiler.

## ERRORS and NOERRORS Execution Option

### Format

```
|ERRORS     (default)
|NOERRORS
```

The ERRORS execution option requests the diagnostic summary to be printed on SYSPRINT.
| NOERRORS indicates the error messages are shown where the error was made, but errors
| are not summarized at the end.

## ERRSYM Execution Option

### Format

```
ERRSYM = character        (The default is the period.)
```

*character* is a single alphanumeric or special character.

ERRSYM designates the character that fills blank columns preceding the cursor C, which is printed above certain PLEX processor error messages, such as message 406: DATA BASE IDENTIFIER NOT FOUND.

## EXIT Execution Option

### Format

```
EXIT = load-module
```

*load-module* is the name of a load module (not link-edited with the PLEX processor).

The EXIT execution option names a user-written exit routine for the PLEX processor. The routine edits the source input (including statements input by a $COPY directive) before processing by the PLEX processor. The routine must reside in JOBLIB, STEPLIB, or the system link library.

As the PLEX processor receives each input record, it calls the specified exit routine and passes the following three parameters:

- the address of the current source record as it stands before processing by the PLEX processor. Also, you use this address if you want to add records.
- the address of a fullword value indicating the current level of $COPY nesting. Binary zero means that the current record was not generated by a $COPY directive (see **$COPY Directive** on page 19-7). Binary 1 means that a $COPY directive in the original source code caused the record to be read from a member of the copy library. Higher numbers indicate the current value of nested $COPY logic, up to a maximum of 10.
- the address of a fullword value containing an action code that is to be set by the exit routine. The action code indicates the disposition of the current record processed by the exit routine. It is initialized to zero by the PLEX processor before each call to the exit routine. The exit routine then sets the action code before passing control back to the PLEX processor, as appropriate:

  **0** means the PLEX processor should accept the current record as passed back (and possibly modified) by the exit routine.

  **1** means the PLEX processor should delete the current record.

  **2** means the PLEX processor should accept the current record as passed back (and possibly modified) by the exit routine and should allow another record to be added on the next call to the exit routine.

  **>2** is the address of a diagnostic message to be written in the output from the PLEX processor after the current record.

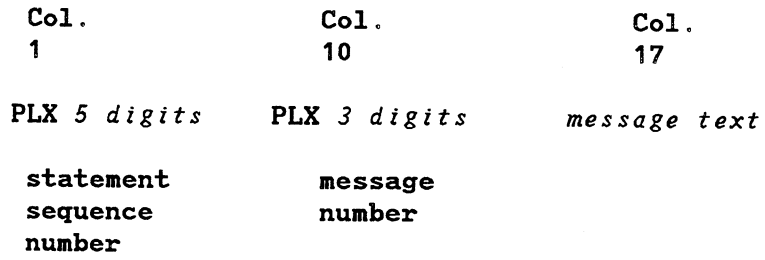
**Modifying, replacing, and adding records**

The exit routine can modify or replace the current record in the source input by receiving it in a call from the PLEX processor, making the appropriate substitution in the field containing the received record, and then sending it back to the PLEX processor. If the action code from the exit routine is not 1 (delete record), the PLEX processor accepts what it receives as the current record.

If the action code is 2, the PLEX processor accepts and processes the current record from the exit routine, then immediately makes another call to the exit routine without taking another record from the source input. The exit routine then supplies a record to be added to the PLEX program, sets the action code, and returns control to the PLEX processor. If the exit routine sets the action code to 2 again this time, the PLEX processor accepts the new record and again calls the exit routine. In this way, one or more records can be added to the PLEX program through multiple calls to the exit routine.

To discontinue this sequence of adding a group of records, the exit routine must set the action code to 0 before it passes the last new record. Action code 0 instructs the PLEX processor to accept the last new record and to read another record from the source input before calling the exit routine again.

**Diagnostic messages** The format for the standard PLEX processor diagnostics is also used for printing diagnostic messages issued by an exit routine.




The PLEX processor automatically sets the statement sequence number. The exit routine determines the message number and message text and passes both to the PLEX processor for printing. The message text must be in two forms: the long form, occupying 16 words, followed by the short form, occupying 5 words. The PLEX processor prints the long form of the message when the NOTERM (default) execution option is in effect and prints the short form when you specified the TERM execution option. (See **TERM and NOTERM Execution Option** on page 19-20.)

In addition, the exit routine must determine both the severity level of the diagnostic and the column in which the cursor for the diagnostic message should appear. (See **PLEX Processor Messages** on page 22-14.)

The field containing the diagnostic message specifications must be arranged as follows:

| Word(s) | Contents |
|---|---|
| 1st | is a fullword value of the column number in which the error in the current record was detected. It determines the positioning of the PLEX processor error cursor, which is printed on a line between the current record and the diagnostic message text. If the column number is less than 4, the cursor is not printed. |
| 2nd | is a fullword value of the severity level of the error. |
| 3rd | is a message number of up to four digits in EBCDIC character representation, left-justified and blank-filled. If the message number is four characters long, no space appears between the message number and the message text when the diagnostic message is printed. |
| 4th-19th | is the long-form message text (64 characters), which is used if NOTERM (default) execution option is in effect. |
| 20th-24th | is the short-form message text (20 characters), which is used if the TERM execution option is in effect. |

**Notes**

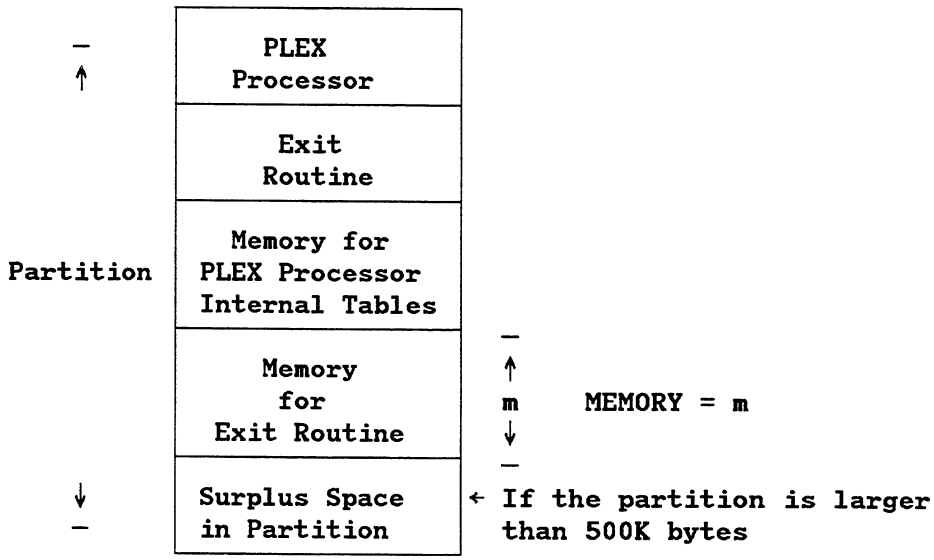
- As an alternative to using a customized exit routine through the EXIT execution option, a standard exit routine named S2KEXIT and developed by systems personnel at the site can be link-edited with the PLEX processor for editing source input from all jobs using the PLEX processor. This exit must have the PROGRAM-ID S2KEXIT (or the CSECT name S2KEXIT if written in Assembler) and must be included as object input during a relinking of the PLEX processor.

  The PLEX processor automatically checks for the name S2KEXIT to determine whether an exit routine was link-edited with the PLEX processor. If S2KEXIT was link-edited with the PLEX processor, do not specify the EXIT execution option without approval from the systems personnel at the site because it overrides use of the standard routine.

- An exit routine can issue up to five diagnostic messages. If more than five messages are issued, PLEX processor diagnostic message 100 is issued.

- An exit routine cannot issue a message for a comment line or a continuation line in the source to the PLEX processor. An attempt to do so brings unpredictable results.

- If the exit routine obtains additional memory using the GETMAIN macro, the amount to be obtained must be specified previously through the MEMORY execution option (see **MEMORY Execution Option** on page 19-17). Through a GETMAIN, the PLEX processor obtains all available space in its partition (up to 500K bytes). Then it reserves the specified memory space for the exit routine and calculates the amount of memory remaining for its internal tables. The resulting arrangement is shown here.



Sufficient memory must be available in the partition for the PLEX processor internal tables for COMMBLOCKs, SSRs, and subschema items described in the PLEX program. The amount of memory required for each of these internal tables is as follows:

COMMBLOCK Table

24 bytes for the table control block plus
72 bytes per entry (one entry per COMMBLOCK in the program)

SSR Table

24 bytes for the table control block plus
52 bytes per entry (one entry per SSR in the program)

Item Table

24 bytes for the table control block plus
80 bytes per entry (one entry per subschema item in the program)

## FORM and NOFORM Execution Option

**Format**

```
|NOFORM      (default)
|FORM
```

If you specify the FORM execution option, the PLEX processor recognizes PLEX commands and directives only if they begin either in column 7 (the default) or in the column specified for *b* in the SORMGIN execution option (see **SORMGIN Execution Option** on page 19-19). If you specify the NOFORM execution option (the default), PLEX commands can be free-form and can begin anywhere after column 6. **Format of COBOL PLEX Commands** on page 3-1 discusses the format of PLEX commands under both execution options, as well as restrictions on keywords that can be used in the program.

FORM is primarily intended to provide upward compatibility with prior SYSTEM 2000 software releases. The use of FORM for new programming efforts is not encouraged.

## GEN and NOGEN Execution Option

**Format**

```
|NOGEN     (default)
|GEN
```

The GEN option specifies the printing of the expanded source listing from the PLEX processor on SYSPRINT. NOGEN suppresses printing the expanded source, except for COMMBLOCKs.

## ITEM Execution Option

### Format

```
ITEM = nnnnnn      (The default is 5.)
```

*nnnnnn* is the average number of subschema items per subschema record.

The ITEM option directs the memory allocation for processing subschema record descriptions. The ITEM and SSR options together are used for optimizing memory usage during the execution of the PLEX processor. See **Memory Utilization during the PLEX Processor Step** on page 22-16 for more information about using this execution option.

## LINECNT Execution Option

### Format

```
LINECNT = nn      (The default is 55.)
```

*nn* is the number of lines per page.

The LINECNT option sets the number of lines per page for SYPRINT output.

## LOAD and NOLOAD Execution Option

### Format

```
|LOAD      (default)
|NOLOAD
```

The LOAD option (default) specifies that the expanded source code for the PLEX program is to be sent to SYSGO for the compilation job step. NOLOAD specifies that no further processing of the PLEX program is to be performed.

## MEMORY Execution Option

### Format

```
MEMORY = bytes      (The default is 4096.)
```

*bytes* is the number of bytes of main memory, up to four digits.

The MEMORY option specifies the amount of main memory that will remain available in the region during the PLEX processor job step. If you also specify an EXIT routine that dynamically obtains additional main memory, you must supply a value for MEMORY.

## QUOTE Execution Option

### Format

```
QUOTE = symbol     (The default is the single quote-mark.)
```

*symbol* is the symbol used in the PLEX program as the delimiter around literals in COMMBLOCK and subschema record descriptions.

The QUOTE option pertains only to COBOL PLEX programs.

## SEQID Execution Option

### Format

```
SEQID  =  |TRAILING   (default)
          |LEADING
```

SEQID specifies that the statement sequence identifier generated by the PLEX processor should be placed to the right (TRAILING) or to the left (LEADING) of the program's source statements. A sequence identifier can be generated for each statement in the source code to the PLEX processor, but not for statements the PLEX processor generates.

In general, if you specify TRAILING (default), the PLEX processor generates an eight-character sequence identifier for each source statement. This identifier consists of the letters PLX and a five-digit number. If you specify LEADING, the PLEX processor generates a five-digit sequence number starting in column 1 of each source line.

The placement of sequence identifiers also depends on the *b* and *d* specifications with the SORMGIN execution option (for COBOL, see **SORMGIN Execution Option** on page 19-19).

For SEQID=LEADING, if the *b* specification (continuation column) is less than 7, not enough columns can be reserved at the beginning of each source line for the five-digit sequence number. In this situation, SEQID=LEADING is ignored. If *b* is greater than or equal to 7, the PLEX processor checks for sequencing or other information already in the source code before column 7 of each line. If the columns preceding column 7 are not blank, SEQID=LEADING is ignored, and no additional sequence numbering is generated by the PLEX processor.

For SEQID=TRAILING, if the *d* specification (sequence number starting position) is greater than 73, not enough columns can be reserved at the end of each source line for the eight-character sequence identifier. In this situation, SEQID=TRAILING is ignored, and no sequence numbers are generated. If *d* is less than or equal to 73, the PLEX processor checks for sequencing or other information already in the source code starting with the specified column. If the field starting with the specified column is not blank, SEQID=TRAILING is ignored, and no additional sequencing is generated.

## SORMGIN Execution Option

### Format

```
SORMGIN = (a , b , c , d)
```

*a* is the column for COBOL sequence numbers.

*b* is the continuation column for code generated by the PLEX processor. It is also the starting column for the scan by the PLEX processor and the column for a comment flag (*).

*c* is the column in which the PLEX processor should place the beginning of the source PLEX command after turning it into a comment (used only if the FORM execution option is specified).

*d* is the sequence number starting position.

SORMGIN designates the margin positions for the source statements. The defaults for COBOL are as follows:

*a* = **1**

*b* = **7**

*c* = **8**

*d* = **73**

If you use the SORMGIN execution option, you must specify *a* through *d* explicitly. The range of values is $1 \leq a \leq b \leq c \leq d \leq 80$. If the values are not in these ranges, the PLEX processor issues an error message.

See **SEQID Execution Option** on page 19-18 for a discussion of the effects of SORMGIN specifications on the SEQID execution option.

## SOURCE and NOSOURCE Execution Option

### Format

```
|SOURCE      (default)
|NOSOURCE
```

The SOURCE option causes the source input, including source COBOL statements, to be listed on SYSPRINT. NOSOURCE causes only PLEX statements to be listed. NOSOURCE implies NOGEN and TERM.

## SSR Execution Option

### Format

```
SSR = nnnn      (The default is 4.)
```

*nnnn* is the average number of subschema records per COMMBLOCK.

The SSR option directs memory allocation for processing subschema record descriptions. This option optimizes memory usage during the execution of the PLEX processor. See **Memory Utilization during the PLEX Processor Step** on page 22-16 for more information about this execution option and utilizing memory.

## SYSSEP Execution Option

### Format

```
SYSSEP = a      (The default is the asterisk.)
```

*a* is a single alphanumeric or special character.

SYSSEP designates the character to be recognized as the system separator character for the ternary operators SPAN, SPANS, SPANNING, EQ, and NE, and in item-to-item comparisons in where-clause processing.

## TERM and NOTERM Execution Option

### Format

```
|NOTERM     (default)
|TERM
```

The TERM option directs the PLEX processor to issue abbreviated, short-form messages, which are suitable for jobs running in TP environments. The NOTERM option (default) specifies long-form PLEX processor messages. See **PLEX Processor Messages** on page 22-14 for the short-form and long-form of PLEX processor messages.

## TRANS Execution Option

### Format

```
TRANS = load-module
```

*load-module* is the name of a load module (not link-edited with PLEX processor) of a translate table routine.

The TRANS option designates a table routine that translates all source input. This table is commonly used for translation of character sets, for example, EBCDIC to ASCII. Refer to the *Principles of Operation* IBM manual for a description of the TR table. The routine must be included in JOBLIB, STEPLIB, or the system link library.

An exit routine named S2KTRANS that is developed by systems personnel at your site can be link-edited with the PLEX processor for translating source input from all jobs using the PLEX processor. The PLEX processor automatically checks whether an exit routine named S2KTRANS was link-edited with the PLEX processor. If S2KTRANS was link-edited with the PLEX processor, do not specify the TRANS execution option without approval from the systems personnel at your site because it overrides use of the standard routine.

Chapter 20

# Sample COBOL PLEX Programs

*GENERALIZED COBOL PLEX PROGRAM 20-2*

*SAMPLE PROGRAM FINDORG 20-3*

*JOB CONTROL LANGUAGE (JCL) 20-25*
- *Single-User JCL for All Job Steps 20-25*
- *Multi-User JCL for All Job Steps 20-25*
- *JCL Variations 20-28*
  - *Additional Locate Files 20-28*
  - *Additional database files 20-29*
  - *$COPY library 20-29*

This chapter contains a generalized skeleton PLEX program and a complete sample COBOL PLEX program, along with sample JCL for preprocessing, compiling, link-editing, and executing a COBOL PLEX program. See the appropriate appendix for sample programs and JCL for the other programming languages.

# GENERALIZED COBOL PLEX PROGRAM

A generalized, skeleton COBOL PLEX program is shown in Illus. 20.1. COBOL statements can appear anywhere in the program. However, executable PLEX commands appear only after the START S2K command. The rest of this chapter contains a complete, documented sample program, FINDORG, which uses the EMPLOYEE database.

**Illus. 20.1** Skeleton of a PLEX Program

```
DATA DIVISION
  WORKING-STORAGE SECTION
  COMMBLOCK(s)                        (One COMMBLOCK must be included
                                      for each database to be accessed.)
       .
       .
       .
  SSR(s)                              (One or more SSRs can
                                      be included to refer to one or more
                                      schema records.  Each SSR
                                      refers to a single schema record.)
       .
       .
       .
  $BLOCKS.                            (The program requires one $BLOCKS
                                      directive in the WORKING-STORAGE
                                      SECTION.)
LINKAGE SECTION.
  COMMBLOCK(s)                        (One or more additional COMMBLOCKs and
                                      SSRs can appear in the LINKAGE SECTION
                                      if they will be passed to this program
                                      from a calling PLEX program.)
  SSR(s)
       .
       .
       .
PROCEDURE DIVISION
       .
       .
       .
  START S2K.                          (First PLEX command executed by the
                                      program)
  (Initialize passwords in
  COMMBLOCKs and open databases)
       .
       .
       .
  (PLEX commands)
       .
       .
       .
  (CLOSE databases)
       .
       .
       .
  (PLEX commands)
       .
       .
       .
  (CLOSE remaining databases)
  STOP S2K                            (Last PLEX command executed by the
                                      program)
       .
       .
       .
```

# SAMPLE PROGRAM FINDORG

The sample program FINDORG uses PLEX retrieval commands and several stacks and Locate Files to produce a roster of all personnel in the company. In this roster, employees are listed under their supervisors.

The program has one subschema record for level 0 data records. It also has two subschema records that refer to schema record C100 (POSITION WITHIN COMPANY). The subschema record POSITION-REC retrieves the title of an employee's position. The other subschema record for C100, COMPARISON, contains two items used for comparing values in the LOCATE command's where-clause.

The program retrieves employee records according to the hierarchy of the organization. It finds the organizational hierarchy by extracting the initials from each manager's name and looking for those initials in the MANAGER item of the most current POSITION WITHIN COMPANY data record in each logical entry.

First, a GET1 command retrieves the president's level 0 data record. Then a LOCATE command establishes a Locate File pointing to all employees who report directly to him. That is, the first Locate File points to the level 0 data records of all employees who report directly to the president.

Next, a GET command retrieves a level 0 record whose address is in the first Locate File. The program extracts the initials from the employee name in this record. Then a LOCATE command finds the level 0 records of all personnel who report directly to this employee, and so on, down the chain-of-command in the company.

When an employee is retrieved who has no subordinates (that is, the LOCATE command is empty), the program logic reverts to the Locate File pointing to this employee's level 0 record. A GET command retrieves the next employee, if there is one, who works for the same manager.

In general, when the program receives return code 04 at the end of one Locate File, it issues a GET command for the previously established Locate File (whose number is one less than that of the latest Locate File). Thus, the program proceeds as far down in the organizational hierarchy as it can before moving across the hierarchy. Eventually, the end of the first Locate File is reached, which means that all subordinate employees have been retrieved.

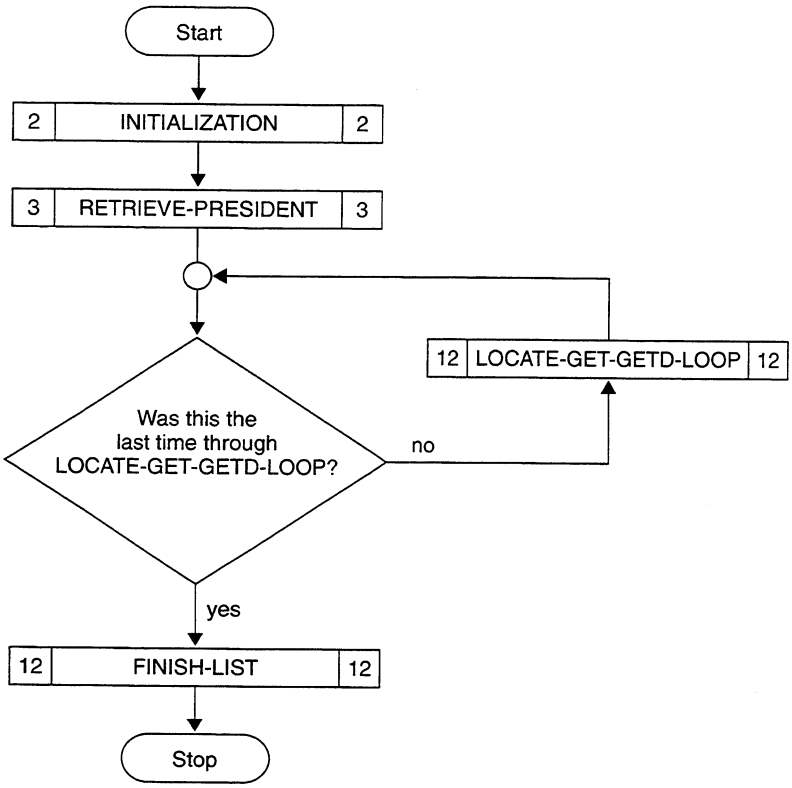
As the program retrieves each employee, it displays an output line with the employee's full name and position title. Each line is indented under the line for the employee's immediate manager.

Illus. 20.2 on page 20-4 consists of a twelve-part flowchart showing the major routines of the COBOL FINDORG program. For ease of reference, the routines are numbered, and the flowchart refers to each of them by number as well as by name.

See Illus. 20.3 on page 20-14 for the source code of FINDORG. Illus. 20.4 on page 20-24 shows the actual output from FINDORG.

**Illus. 20.2** Flowchart of FINDORG

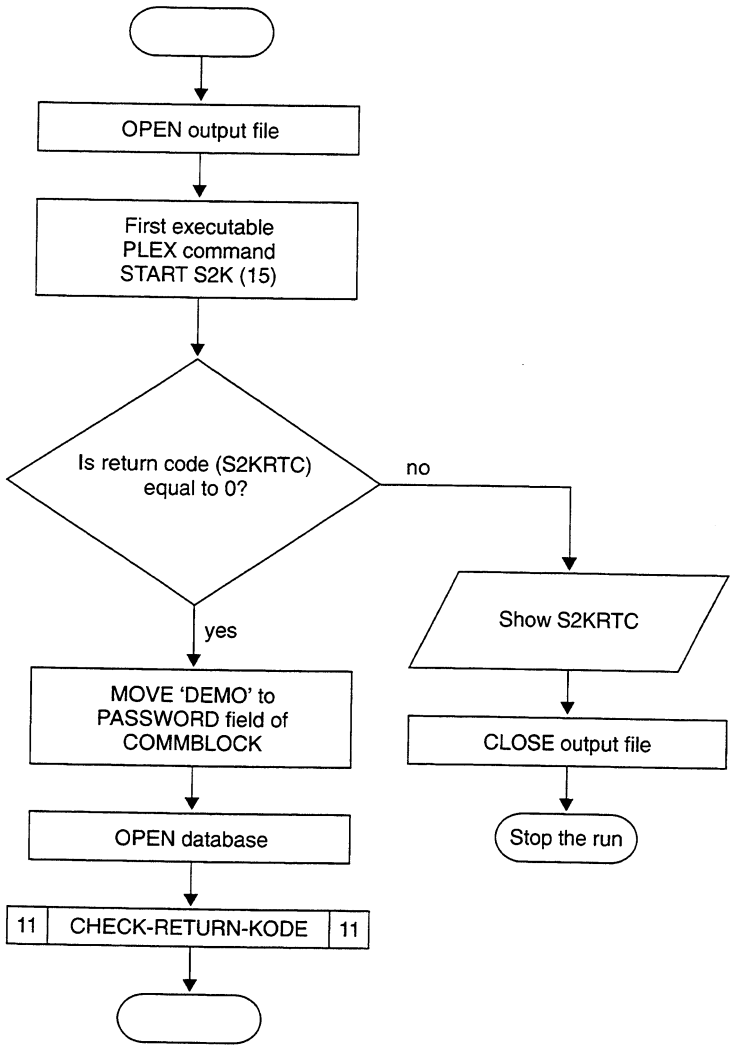
## 1. MAIN-ROUTINE




*continued on next page*

**Illus. 20.2** *continued*

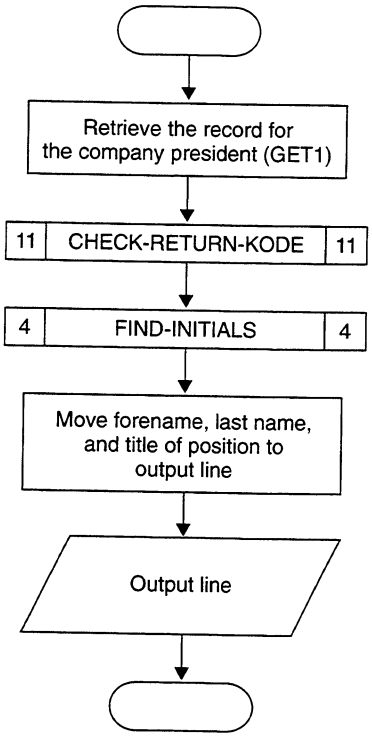
## 2. INITIALIZATION




*continued on next page*

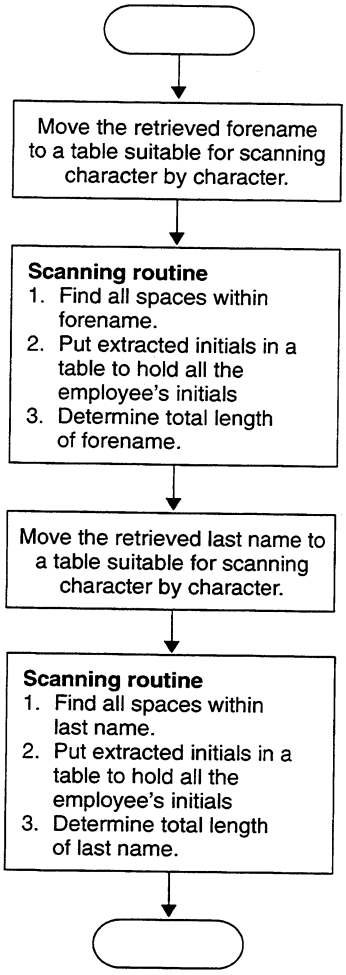
**Illus. 20.2** *continued*

### 3. RETRIEVE-PRESIDENT




### 4. FIND-INITIALS

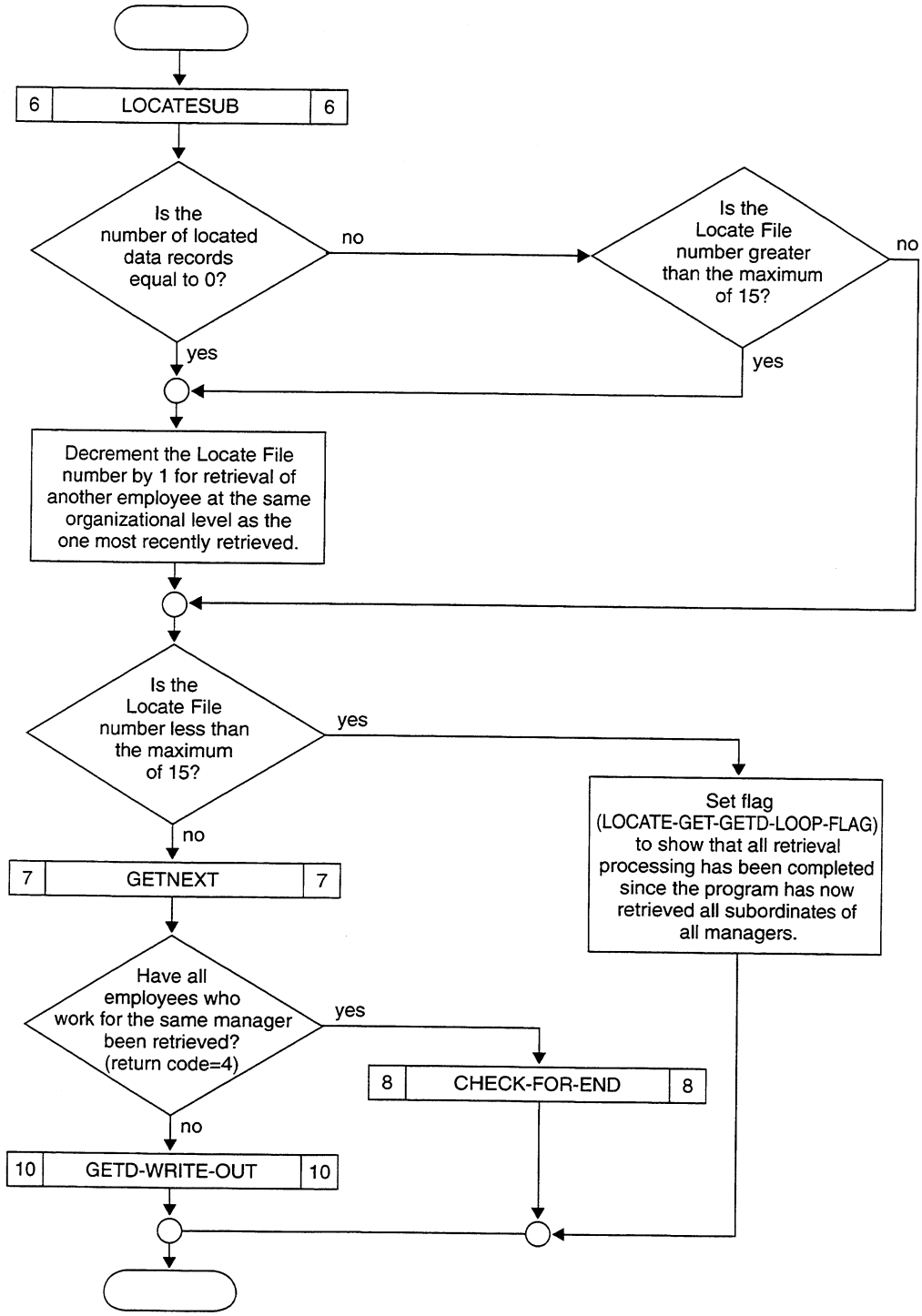
(Since the processing of this routine is only incidental to the PLEX processing in the rest of the program, a higher-level outline is presented here.)




*continued on next page*

**Illus. 20.2** *continued*

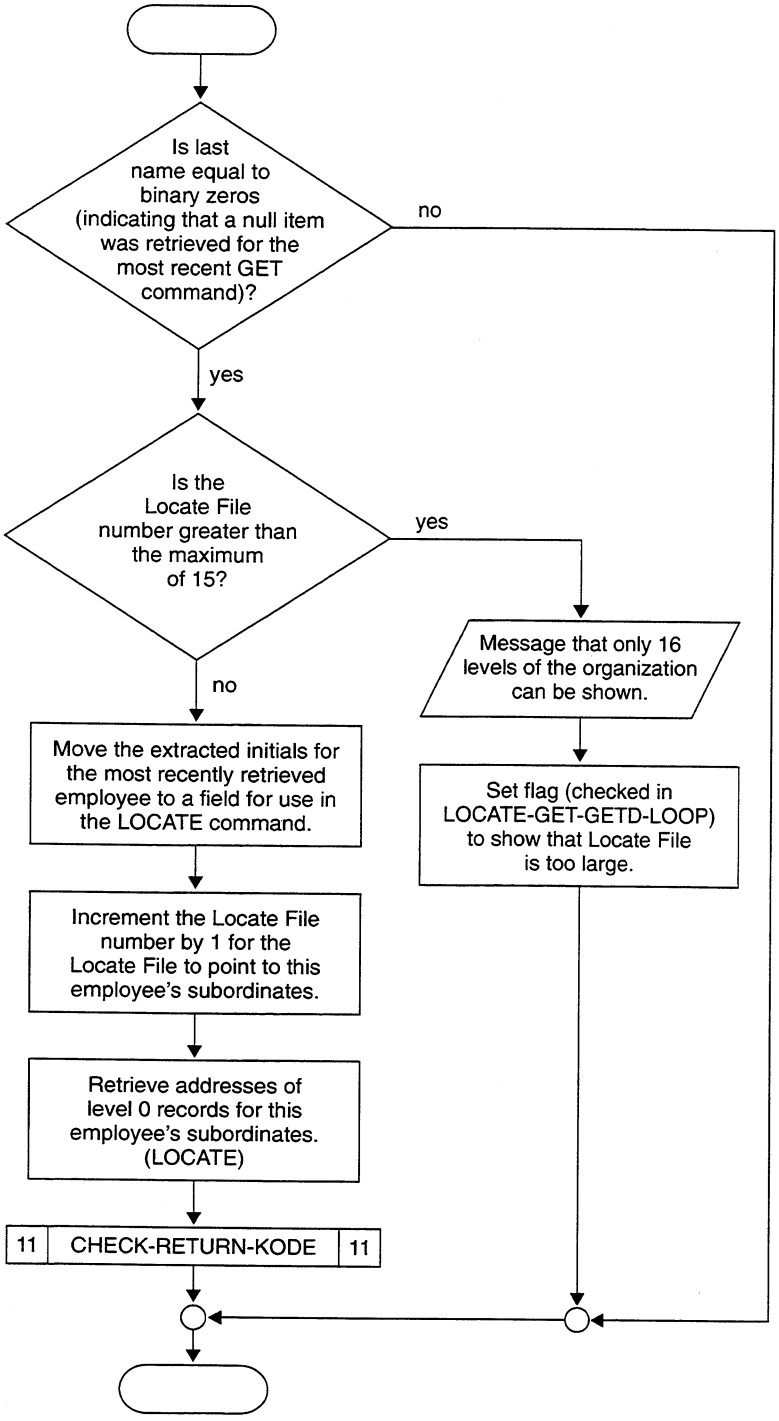
## 5. LOCATE-GET-GETD-LOOP




*continued on next page*

**Illus. 20.2** *continued*

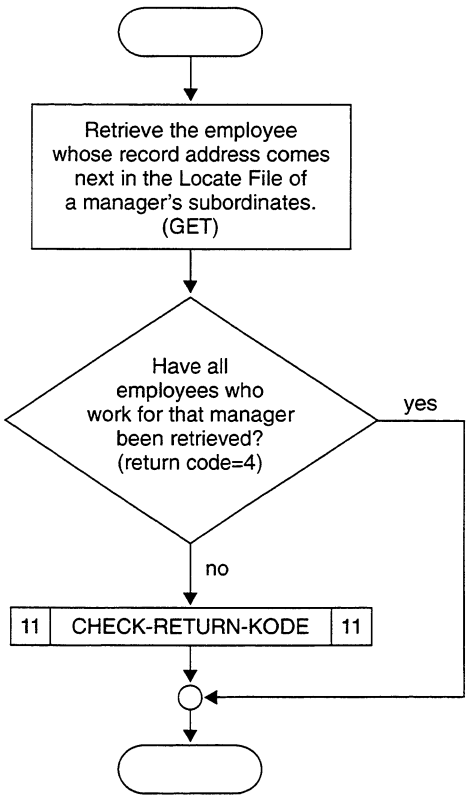
## 6. LOCATESUB




*continued on next page*

**Illus. 20.2** *continued*

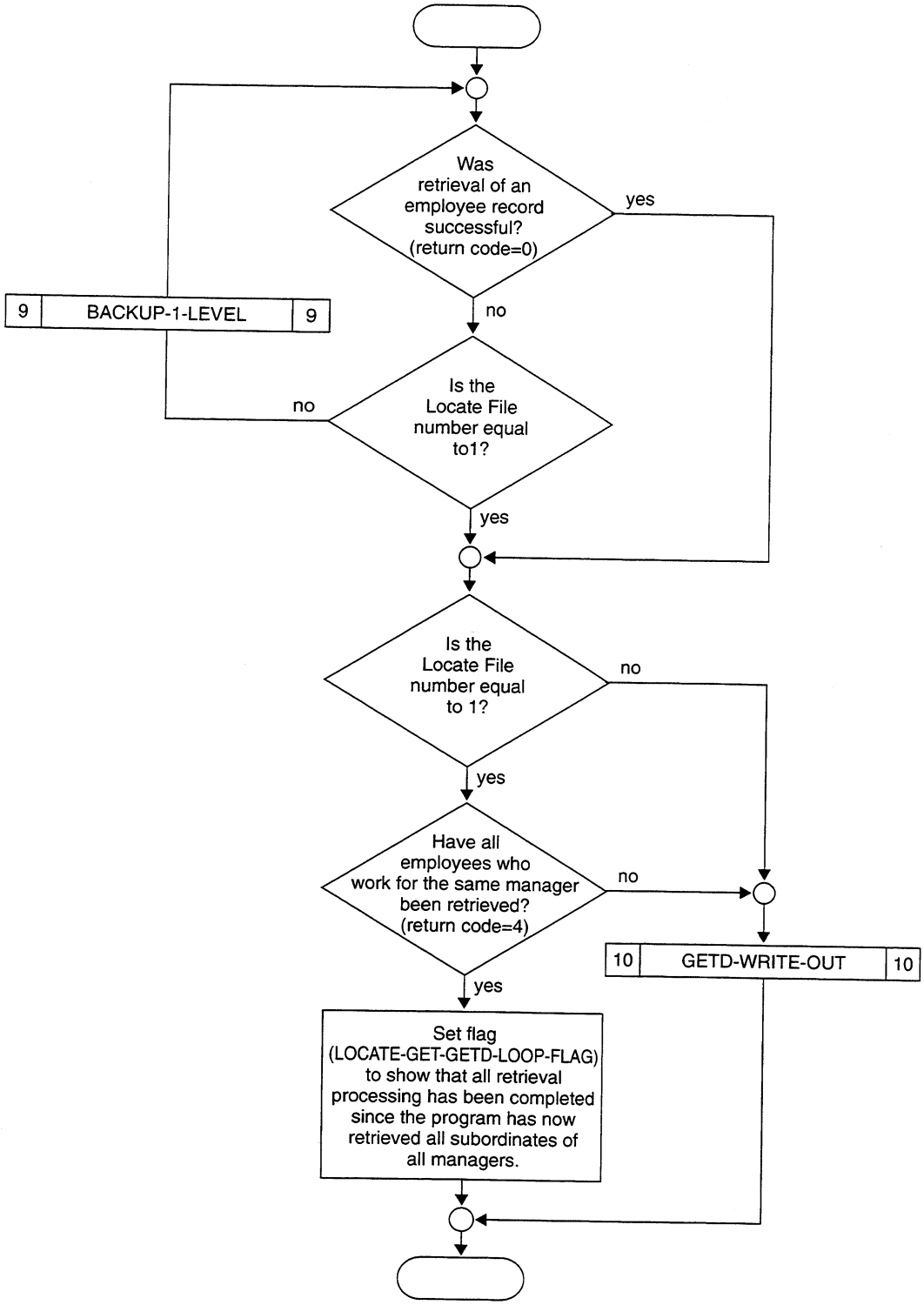
## 7. GETNEXT




*continued on next page*

**Illus. 20.2** *continued*

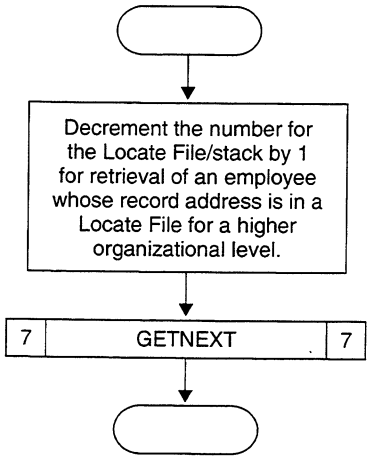
**8. CHECK-FOR-END**




*continued on next page*

**Illus. 20.2** *continued*

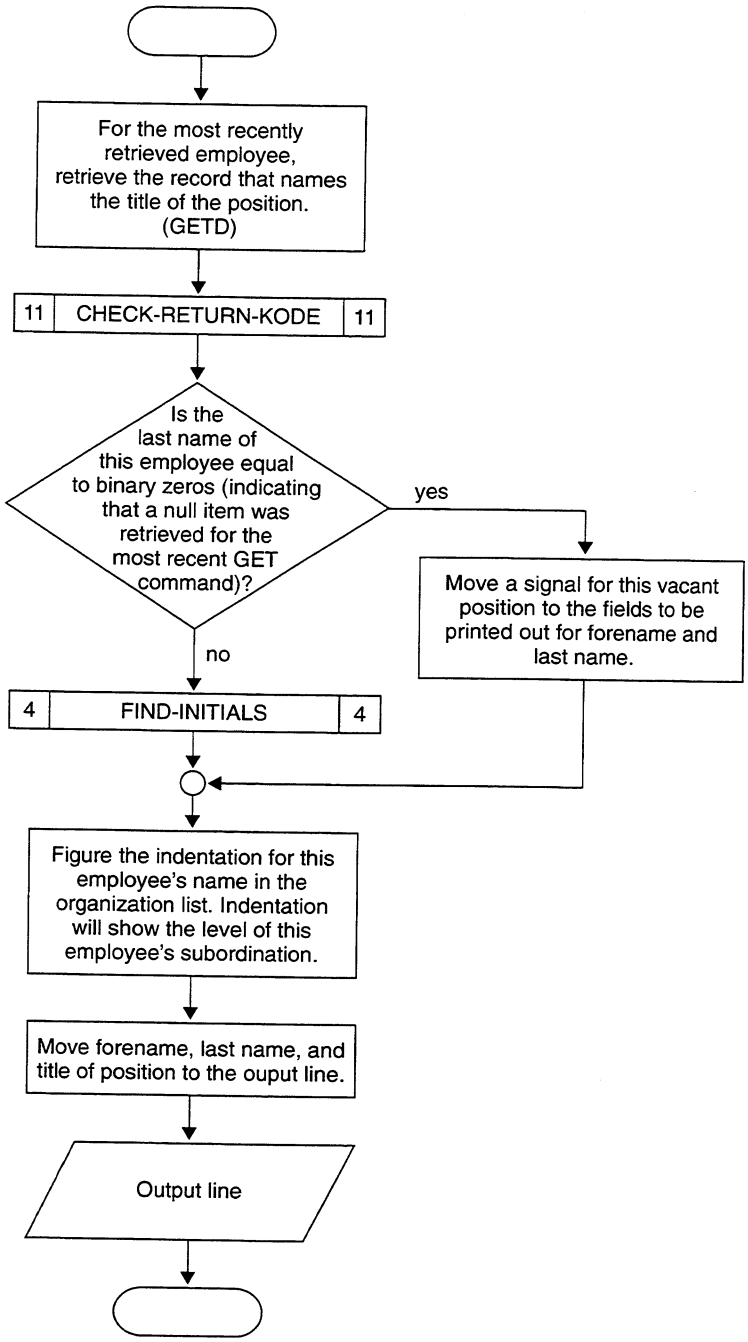
## 9. BACKUP-1-LEVEL




*continued on next page*

**Illus. 20.2** *continued*

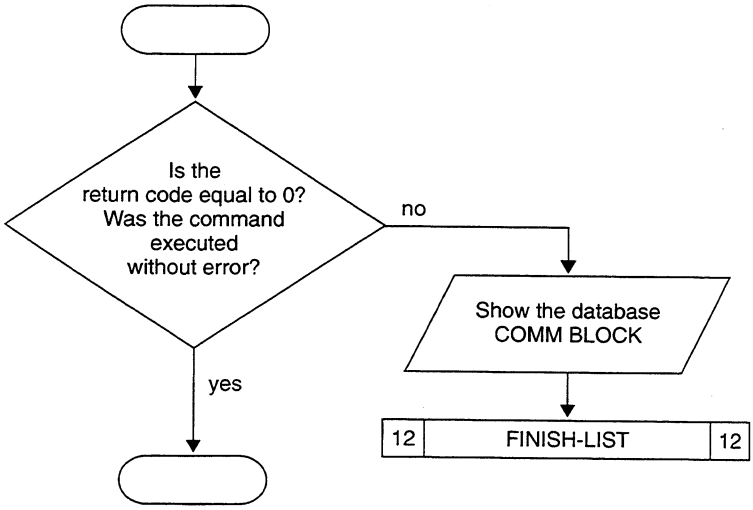
## 10. GETD-WRITE-OUT



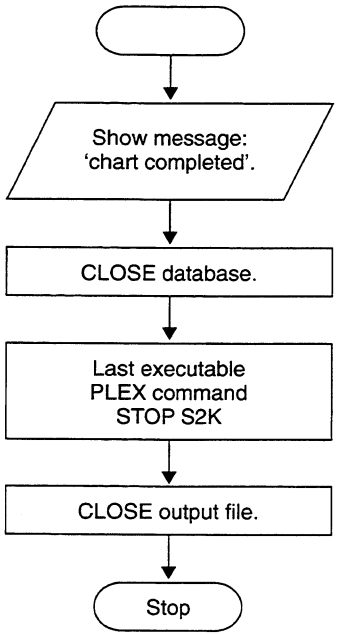

*continued on next page*

**Illus. 20.2** *continued*

## 11. CHECK-RETURN-KODE




## 12. FINISH-LIST



**Illus. 20.3** Source Statements for COBOL PLEX FINDORG Program

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  FINDORG.
*****************************************************************
*ABSTRACT:                                                      *
*    THIS PROGRAM USES THE EMPLOYEE DATABASE TO FIND THE        *
*    ORGANIZATION OF THE COMPANY.  IT USES MULTIPLE LOCATE FILES *
*    AND STACKS.  IT FINDS THE PRESIDENT, THEN ALL PEOPLE       *
*    WHO WORK FOR THE PRESIDENT, THEN ALL THEIR SUBORDINATES,   *
*    AND SO ON.  THE NAME AND TITLE OF EACH EMPLOYEE IS         *
*    PRINTED.  THE NAME OF EACH EMPLOYEE APPEARS UNDER HIS      *
*    EMPLOYER AND IS INDENTED TO SHOW THE APPROPRIATE LEVEL     *
*    OF THE ORGANIZATION.  IT IS ASSUMED THAT EACH EMPLOYEE     *
*    HAS A UNIQUE COMBINATION OF INITIALS.                      *
*                                                               *
*NARRATIVE:                                                     *
*    STEP                        ACTION                         *
*     1.  INITIALIZATION.                                       *
*     2.  RETRIEVE-PRESIDENT: FIND THE PRESIDENT OF THE COMPANY. *
*             PRINT HIS NAME AND TITLE, AND FIND HIS INITIALS.  *
*     3.  LOCATE-GET-GETD-LOOP: LOCATE ALL SUBORDINATES OF A    *
*             PERSON WHOSE INITIALS HAVE BEEN FOUND.  IF THERE  *
*             ARE NONE, CHECK IF NEED TO FIND SUBORDINATES      *
*             OF ANY OTHER PERSON.  IF NOT, GO TO STEP 4.       *
*             IF SO, GET THE NEXT SUBORDINATE.  IF THERE IS NO  *
*             SUBORDINATE AT THIS LEVEL, THEN GO BACK UP TO THE *
*             NEXT-HIGHER ORGANIZATION LEVEL (DETERMINED BY THE *
*             PREVIOUS LOCATE).  IF THERE IS NO PREVIOUS LOCATE,*
*             GO TO STEP 4.  OTHERWISE, FIND THE NAME AND (WITH *
*             A GETD COMMAND) THE TITLE OF THIS SUBORDINATE, AND*
*             PRINT THEM.  FIND THIS PERSON'S INITIALS AND THEN *
*             LOCATE HIS SUBORDINATES.  REPEAT THIS STEP UNTIL  *
*             ALL EMPLOYEES HAVE BEEN RETRIEVED WITH LOCATE AND *
*             GET COMMANDS.                                     *
*     4.  FINISH-LIST: PRINT FOOTING, CLOSE FILES, STOP.        *
*****************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-370.
 OBJECT-COMPUTER.  IBM-370.
 SPECIAL-NAMES.
     C01 IS TO-TOP-OF-PAGE.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT OUT-FILE ASSIGN TO UT-S-PRINTER.
```

*continued on next page*

**Illus. 20.3** *continued*

```
DATA DIVISION.
FILE SECTION.
FD  OUT-FILE
     LABEL RECORDS ARE OMITTED.
01  LINE-OUT                PIC X(80).
WORKING-STORAGE SECTION.

01  ENVIRON-NUM             PIC 9  COMPUTATIONAL-3.
01  COMMAND                 PIC X(6).
01  PRESIDENT               PIC X(9)  VALUE 'PRESIDENT'.
01  FORENAME-LENGTH         PIC 99  COMPUTATIONAL.
01  FORENAME-TABLE.
    05  FORENAME-CHAR       PIC X
                            OCCURS 3 TO 16 TIMES
                            DEPENDING ON FORENAME-LENGTH
                            INDEXED BY FORENAME-INDEX.
01  LASTNAME-LENGTH         PIC 99  COMPUTATIONAL.
01  LASTNAME-TABLE.
    05  LASTNAME-CHAR       PIC X
                            OCCURS 3 TO 16 TIMES
                            DEPENDING ON LASTNAME-LENGTH
                            INDEXED BY LASTNAME-INDEX.
01  INITIALS-LOCN           PIC 9  COMPUTATIONAL.
01  INITIALS-TABLE.
    05  INITIALS-CHAR       PIC X
                            OCCURS 2 TO 6 TIMES
                            DEPENDING ON INITIALS-LOCN.
01  LINE-OUT-START-COL      PIC 99.
01  LFNUM                   PIC 99  COMPUTATIONAL  VALUE 0.
01  INITS                   PIC X(4).
01  END-FORENAME-CHAR-FLAG    PIC X.
    88 END-OF-FORENAME                VALUE 'Y'.
01  END-LASTNAME-CHAR-FLAG    PIC X.
    88 END-OF-LASTNAME                VALUE 'Y'.
01  LFNUM-FLAG              PIC X  VALUE 'L'.
    88 LFNUM-IS-TOO-HIGH            VALUE 'H'.
01  LOCATE-GET-GETD-LOOP-FLAG     PIC XXXX  VALUE 'NOT '.
    88 LAST-LOOP-EXECUTION-COMPLETE         VALUE 'DONE'.
```

*continued on next page*

**Illus. 20.3** *continued*

```
* COMMBLOCK FOR THE DATABASE:
COMMBLOCK OF EMPLOYEE.
 01 EMPLOYEE.
     05  SSR-NAME.
     05  RETURN-KODE.
     05  RETURN-KODE-X  REDEFINES RETURN-KODE  PIC XXXX.
     05  FREE.
     05  LAST-RECORD-FLAG.
     05  CPASSWORD VALUE IS 'PASS'.
     05  NO-OF-RECORDS.
     05  LF-POINTER.
     05  LEVEL.
     05  CTIME.
     05  CDATE.
     05  CYCLE.
     05  SEPARATOR.
     05  TERMINATOR.
     05  DAMAGE.
* SUBSCHEMA RECORD CORRESPONDING TO LEVEL 0 SCHEMA RECORD:
SUBSCHEMA RECORD ENTRY-REC OF EMPLOYEE.
 01  ENTRY-REC.
     05  LASTNAME          PIC X(16).
         EQ C2.
     05  FORENAME          PIC X(16).
         EQ C3.
* SUBSCHEMA RECORD CORRESPONDING TO C100 SCHEMA RECORD:
SUBSCHEMA RECORD POSITION-REC OF EMPLOYEE.
 01  POSITION-REC.
     05  TITLEX            PIC X(48).
         EQ C101.
* SUBSCHEMA RECORD FOR C100 SR, USED IN LOCATE COMMAND
SUBSCHEMA RECORD COMPARISON OF EMPLOYEE.
 01  COMPARISON.
     05  MANAGER           PIC X(6).
         EQ C103.
     05  END-DATE          PIC X(6).
         EQ C106.
$BLOCKS.
```

*continued on next page*

**Illus. 20.3** *continued*

```
 LINKAGE SECTION.
 01  PARMS.
*      A PARAMETER IS PASSED TO THE PROGRAM FROM THE EXEC LINE
*      IN THE JCL, TO REPRESENT THE ENVIRONMENT SPECIFIED IN THE
*      START S2K COMMAND (IN INITIALIZATION).  THE PROGRAM CAN
*      THUS BE RUN IN EITHER A SINGLE-USER OR A MULTI-USER
*      ENVIRONMENT WITHOUT CHANGING THE SOURCE CODE.
     05  PARM-LENGTH       PIC S9(4)  COMPUTATIONAL.
     05  ENVIRON-NUM-IN    PIC 9.
*****************************************************************
*                                                               *
*                        TOP OF MODULE                          *
*                                                               *
*****************************************************************
 PROCEDURE DIVISION USING PARMS.
   MAIN-ROUTINE.
***
* STEP 1: INITIALIZATIONS.
***
     PERFORM INITIALIZATION.
***
* STEP 2: RETRIEVE THE PRESIDENT TO BEGIN THE ORGANIZATION LIST.
***
     PERFORM RETRIEVE-PRESIDENT.
***
* STEP 3: LOCATE AND RETRIEVE ACCORDING TO COMPANY HIERACHY:
*         ALL 15 LOCATE FILES ARE NOT ACTUALLY USED IF THE
*         COMPANY ORGANIZATION DOES NOT HAVE 15 LEVELS OF
*         SUBORDINATES UNDER THE PRESIDENT.  THUS, LOCATE-GET-
*         GETD-LOOP ENDS WHENEVER ALL EMPLOYEES HAVE BEEN
*         RETRIEVED.
*
***
     PERFORM LOCATE-GET-GETD-LOOP
         UNTIL LAST-LOOP-EXECUTION-COMPLETE.
```

*continued on next page*

**Illus. 20.3** *continued*

```
***
* STEP 4: PRINT FOOTING, CLOSE FILES, STOP.
***
     PERFORM FINISH-LIST.
***
* STEP 1'S PARAGRAPH:
***
   INITIALIZATION.
     OPEN OUTPUT OUT-FILE.
     IF PARM-LENGTH IS EQUAL TO 0
         MOVE 0 TO ENVIRON-NUM
     ELSE
         MOVE ENVIRON-NUM-IN TO ENVIRON-NUM.
START S2K (15)  USE (ENVIRON-NUM).
     IF S2KRTC IS NOT EQUAL TO 0
         DISPLAY 'ON START S2K, S2KRT = ' S2KRTC
         CLOSE OUT-FILE
         STOP RUN.
     MOVE 'DEMO' TO CPASSWORD.
OPEN EMPLOYEE.
     MOVE 'OPEN' TO COMMAND.
     PERFORM CHECK-RETURN-KODE.
*** * STEP 2'S MAIN PARAGRAPH:  ***
   RETRIEVE-PRESIDENT.
GET1 ENTRY-REC WHERE END-DATE FAILS AND TITLEX EQ PRESIDENT.
     MOVE 'GET1' TO COMMAND.
     PERFORM CHECK-RETURN-KODE.
     PERFORM FIND-INITIALS.
     MOVE 2 TO LINE-OUT-START-COL.
     STRING
         FORENAME-TABLE  DELIMITED BY SIZE
             SPACE
         LASTNAME-TABLE  DELIMITED BY SIZE
             ', '
         'PRESIDENT'     DELIMITED BY SIZE
         INTO LINE-OUT
         WITH POINTER LINE-OUT-START-COL.
     WRITE LINE-OUT AFTER ADVANCING TO-TOP-OF-PAGE.
```

*continued on next page*

**Illus. 20.3** *continued*

```
***
* PARAGRAPH (USED IN STEPS 2 AND 3) TO EXTRACT INITIALS FROM
* FORENAME AND LAST NAME AND ALSO TO FIND LENGTH OF NAMES:
***
   FIND-INITIALS.
*    WORK ON FORENAME.
     MOVE 16 TO FORENAME-LENGTH.
     MOVE FORENAME TO FORENAME-TABLE.
     MOVE 1 TO INITIALS-LOCN.
     MOVE FORENAME-CHAR (1) TO INITIALS-CHAR (INITIALS-LOCN).
     MOVE 'N' TO END-FORENAME-CHAR-FLAG.
     SET FORENAME-INDEX TO 1.
     PERFORM FIND-FORE-INITIALS-LENGTH
         UNTIL END-OF-FORENAME.
     SET FORENAME-INDEX DOWN BY 2.
*    FORENAME-INDEX NOW POINTS TO LAST CHARACTER OF FORENAME.
     SET FORENAME-LENGTH TO FORENAME-INDEX.
*    FORENAME-LENGTH DETERMINES SIZE OF FORENAME-TABLE FOR
*    GETD-WRITE-OUT.
*    WORK ON LASTNAME.
     MOVE 16 TO LASTNAME-LENGTH.
     MOVE LASTNAME TO LASTNAME-TABLE.
     ADD 1 TO INITIALS-LOCN.
     MOVE LASTNAME-CHAR (1) TO INITIALS-CHAR (INITIALS-LOCN).
     MOVE 'N' TO END-LASTNAME-CHAR-FLAG.
     SET LASTNAME-INDEX TO 1.
     PERFORM FIND-LAST-INITIALS-LENGTH
         UNTIL END-OF-LASTNAME.
     SET LASTNAME-INDEX DOWN BY 2.
*    LASTNAME-INDEX NOW POINTS TO LAST CHARACTER OF LASTNAME.
     SET LASTNAME-LENGTH TO LASTNAME-INDEX.
*    LASTNAME-LENGTH DETERMINES SIZE OF LASTNAME-TABLE FOR
*    GETD-WRITE-OUT.
   FIND-FORE-INITIALS-LENGTH.
     SET FORENAME-INDEX UP BY 1.
     SEARCH FORENAME-CHAR
         AT END MOVE 'Y' TO END-FORENAME-CHAR-FLAG
         WHEN FORENAME-CHAR (FORENAME-INDEX) IS EQUAL TO ' '
             PERFORM TAKE-FORE-INITIAL.
```

*continued on next page*

**Illus. 20.3** *continued*

```
      TAKE-FORE-INITIAL.
        SET FORENAME-INDEX UP BY 1.
        IF FORENAME-CHAR (FORENAME-INDEX) IS NOT EQUAL TO ' '
            ADD 1 TO INITIALS-LOCN
            MOVE FORENAME-CHAR (FORENAME-INDEX)
                TO INITIALS-CHAR (INITIALS-LOCN)
        ELSE
            MOVE 'Y' TO END-FORENAME-CHAR-FLAG.
      FIND-LAST-INITIALS-LENGTH.
        SET LASTNAME-INDEX UP BY 1.
        SEARCH LASTNAME-CHAR
            AT END MOVE 'Y' TO END-LASTNAME-CHAR-FLAG
            WHEN LASTNAME-CHAR (LASTNAME-INDEX) IS EQUAL TO ' '
                PERFORM TAKE-LASTNAME-INITIAL.
      TAKE-LASTNAME-INITIAL.
        SET LASTNAME-INDEX UP BY 1.
        IF LASTNAME-CHAR (LASTNAME-INDEX) IS NOT EQUAL TO ' '
            ADD 1 TO INITIALS-LOCN
            MOVE LASTNAME-CHAR (LASTNAME-INDEX)
                TO INITIALS-CHAR (INITIALS-LOCN)
         ELSE
            MOVE 'Y' TO END-LASTNAME-CHAR-FLAG.
***
* STEP 3'S MAIN PARAGRAPH:
***
      LOCATE-GET-GETD-LOOP.
        PERFORM LOCATESUB.
        IF NO-OF-RECORDS IS EQUAL TO 0  OR
           LFNUM-IS-TOO-HIGH
            SUBTRACT 1 FROM LFNUM.
        IF LFNUM IS LESS THAN 1
            MOVE 'DONE' TO LOCATE-GET-GETD-LOOP-FLAG
        ELSE
            PERFORM GETNEXT
            IF RETURN-KODE-X IS EQUAL TO '0004'
                PERFORM CHECK-FOR-END
            ELSE
                PERFORM GETD-WRITE-OUT.
```

*continued on next page*

**Illus. 20.3** *continued*

```
    LOCATESUB.
      IF LASTNAME IS NOT EQUAL TO LOW-VALUES
          IF LFNUM IS GREATER THAN 15
              DISPLAY 'ONLY 16 LEVELS OF COMPANY SHOWN.'
              MOVE 'H' TO LFNUM-FLAG
          ELSE
              MOVE INITIALS-TABLE TO INITS
              ADD 1 TO LFNUM
              LOCATE (LFNUM) ENTRY-REC WHERE END-DATE FAILS AND
                  MANAGER EQ INITS.
              MOVE 'LOCATE' TO COMMAND
              PERFORM CHECK-RETURN-KODE.
    GETNEXT.
GET (LFNUM) ENTRY-REC NEXT.
      IF RETURN-KODE-X IS NOT EQUAL TO '0004'
          MOVE 'GET' TO COMMAND
          PERFORM CHECK-RETURN-KODE.
* PARAGRAPH TO CHECK FOR END OF LOCATE FILES USED SO FAR:
    CHECK-FOR-END.
      PERFORM BACKUP-1-LEVEL
          UNTIL RETURN-KODE-X IS EQUAL TO '0000' OR
                LFNUM IS EQUAL TO 1.
      IF LFNUM IS EQUAL TO 1 AND
          RETURN-KODE-X IS EQUAL TO '0004'
          MOVE 'DONE' TO LOCATE-GET-GETD-LOOP-FLAG
      ELSE
          IF RETURN-KODE-X IS EQUAL TO '0000'
              PERFORM GETD-WRITE-OUT.
    BACKUP-1-LEVEL.
      SUBTRACT 1 FROM LFNUM.
      PERFORM GETNEXT.
```

*continued on next page*

**Illus. 20.3** *continued*

```
    GETD-WRITE-OUT.
GETD (LFNUM) POSITION-REC NEXT.
          MOVE 'GETD' TO COMMAND.
          PERFORM CHECK-RETURN-KODE.
      IF LASTNAME IS EQUAL TO LOW-VALUES
          MOVE 10 TO FORENAME-LENGTH
          MOVE '*** VACANT' TO FORENAME-TABLE
          MOVE 12 TO LASTNAME-LENGTH
          MOVE 'POSITION ***' TO LASTNAME-TABLE
      ELSE
          PERFORM FIND-INITIALS.
      COMPUTE LINE-OUT-START-COL = (LFNUM * 2) + 1.
*       CAUSES INDENTATION ACCORDING TO LEVEL IN COMPANY ORG.
      MOVE SPACES TO LINE-OUT.
      STRING
          FORENAME-TABLE  DELIMITED BY SIZE
              SPACE
          LASTNAME-TABLE  DELIMITED BY SIZE
              ', '
          TITLEX          DELIMITED BY SIZE
              INTO LINE-OUT
              WITH POINTER LINE-OUT-START-COL.
      WRITE LINE-OUT AFTER ADVANCING 1 LINES.
***
* STEP 4'S PARAGRAPH:
***
    FINISH-LIST.
      MOVE ' ORGANIZATION CHART COMPLETED.' TO LINE-OUT.
      WRITE LINE-OUT AFTER ADVANCING 2 LINES.
CLOSE EMPLOYEE.
STOP S2K.
      CLOSE OUT-FILE.
      STOP RUN.
```

*continued on next page*

**Illus. 20.3** *continued*

```
***
* PARAGRAPH TO CHECK RETURN-KODE AND TERMINATE IF NOT ZERO:
***
   CHECK-RETURN-KODE.
*       RETURN-KODE WAS REDEFINED AS RETURN-KODE-X (PIC XXXX) SO
*       THAT COBOL NEED NOT CONVERT IT TO PACKED-DECIMAL TO COMPARE
*       WITH ZERO.
        IF RETURN-KODE-X IS NOT EQUAL TO '0000'
            DISPLAY 'COMMBLOCK AFTER ' COMMAND
            DISPLAY EMPLOYEE
            PERFORM FINISH-LIST.
```

**Illus. 20.4** Output from FINDORG Program

```
CLIFTON P. WATERHOUSE, PRESIDENT
  YOLANDA SALAZAR, ADMINISTRATIVE ASSISTANT
  HUGH E. BOWMAN, EXECUTIVE VICE-PRESIDENT
    DARRYL NATHANIEL, SECRETARY
    OLAN M. GARRETT, VICE-PRESIDENT MARKETING
      ALTHEA KNIGHT, SECRETARY
      VIRGINA P. BROWN, MANAGER WESTERN REGION MARKETING
        FANNIE LITTLEJOHN, SECRETARY
        MADISON A. SCHOLL, SR SALES REPRESENTATIVE
          RUBEN R. BROOKS, JR SALES REPRESENTATIVE
        ARMANDO JUAREZ, SR SALES REPRESENTATIVE
        DAVID AMEER, SR SALES REPRESENTATIVE
      TRAVIS Z. RICHARDSON, MANAGER EASTERN REGION MARKETING
        MERRILEE D. WAGGONNER, SECRETARY
        ALAN F. GOODSON, SR SALES REPRESENTATIVE
          JERRY LEE SMITH, JR SALES REPRESENTATIVE
        LELAND G. SHROPSHIRE, JR SALES REPRESENTATIVE
        TAI CHAN, SR SALES REPRESENTATIVE
      GWENDOLYN VAN HOTTEN, MANAGER PUBLIC RELATIONS & EDUCATION
        ROMUALDO R. RODRIGUEZ, PR & ADVERTISING SPECALIST
        JANICE L. WILLIAMSON, INSTRUCTOR
        GEORGE J. GIBSON, INSTRUCTOR
    JOEL B. MILLSAP, VICE-PRESIDENT INFORMATION SYSTEMS
      CARRIE ANN FAULKNER, SECRETARY
      JACOB CAHILL, MANAGER SYSTEMS DEVELOPMENT & MAINTENANCE
        MOLLY I. GIBSON, TECHNICAL WRITER
        MICHAEL Y. JONES, SR SYSTEMS ANALYST
          DAVID G. REID, PROGRAMMER
          ***VACANT POSITION***, PROGRAMMER
          FRANCISCO GARCIA, JR PROGRAMMER/ANALYST
        RICHARD B. REDFOX, SYSTEMS ANALYST
      JESSE L. HERNANDEZ, MANAGER DATA BASE ADMINISTRATION
        LEOPOLD FREEMAN, SR SYSTEMS PROGRAMMER
        SOPHIA FERNANDEZ, STANDARDS & PROCEDURES ANALYST
      IVAN L. POLANSKI, MANAGER SYSTEMS OPERATIONS & RESOURCES
        JANET F. SMITH, SR SYSTEMS PROGRAMMER
          BRADFORD JOHNSON, JR SYSTEMS PROGRAMMER
        PEDRO QUINTERO, OPERATIONS SUPERVISOR
          WILLIAM D. SAVAGE, COMPUTER OPERATOR
          GARY SEATON, COMPUTER OPERATOR
          KENNETH D. REED, COMPUTER LIBRARIAN
    PATRICE R. KNAPP, VICE-PRESIDENT ADMINISTRATION & FINANCE
      FRANK A. CANADY, MANAGER PERSONNEL
        PENNY SCHMIDT, SECRETARY
        STEWART Q. THROCKMORTON, OFFICE SUPERVISOR
          LEONARD R. SLYE, GENERAL MAINTENANCE
          LILLIAN COLLINS, MAIL CLERK
          FREDDIE KAATZ, SUPPLY CLERK
      PATSY MUELLER, SECRETARY
      RITA M. JONES, MANAGER ACCOUNTING
        GARLAND P. SMITH, BOOKKEEPER

ORGANIZATION CHART COMPLETED
```

# JOB CONTROL LANGUAGE (JCL)

A PLEX job has the following four distinct steps:

- preprocessing with the PLEX processor
- compiling
- link-editing
- executing.

This section discusses sample JCL for these job steps for both single-user and Multi-User environments. **COBOL PLEX Processor Execution Options** on page 19-10 describes the execution parameters that you can use with COBOL PLEX.

## Single-User JCL for All Job Steps

In the START S2K command, you can specify that a program will be run in a single-user environment. When the subscript following the keyword USE equals zero or when USE and the subscript are not included in the command, it signifies a single-user environment. See **START S2K** on page 6-1 for information about the START S2K command.

Illus. 20.5 on page 20-26 shows a sample of the JCL required for all four steps in a single-user environment. The execution step requires scratch pads and sort files. These files are dynamically allocated and do not need to be specified in your JCL. See the *SYSTEM 2000 Product Support Manual, Version 12, First Edition* for details about changing pad and sort file default sizes.

## Multi-User JCL for All Job Steps

The single-user JCL can also be used for a Multi-User environment with the following modifications:

- Delete all DD statements for SYSTEM 2000 work files and all database files from the GO step. (These files are set up when Multi-User is initialized and are shared by all users.)
- Reduce the REGION parameter in the GO statement to 128K.

In the START S2K command, you can specify that a program will be run in a Multi-User environment by setting the subscript following the keyword USE equal to 1. See **START S2K** on page 6-1 for information about the START S2K command.

**Illus. 20.5** Sample JCL for a Single-User COBOL PLEX Program

```
//* -------------- MODIFY JOB CARD AS NEEDED -----------------
//*
//* -----  CHANGE SYSIN TO COBOL SOURCE PROGRAM   ------------
//PRECOM   EXEC PGM=PRCOMC
//STEPLIB  DD   DSN=S2K.R121.LOAD,DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DISP=SHR,DSN=S2K.R121.SOURCE(FINDORG)
//SYSGO    DD   DSN=&&CSOURCE,DISP=(MOD,PASS),UNIT=SYSDA,
//             DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//             SPACE=(CYL,(1,1))
//SYSUT1   DD   UNIT=SYSDA,SPACE=(TRK,(20,20))
//SYSUT2   DD   UNIT=SYSDA,SPACE=(TRK,(20,20))
//S2KSNAP  DD   SYSOUT=A
//* ------------------------------------------------------------
//COB      EXEC PGM=IKFCBL00,
//             PARM=('SIZE=307200',NONUM,LOAD,APOST,NODMAP,
//             NOPMAP,NODYNAM,NORES,NOOPT),COND=(4,LT,PRECOM)
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   SPACE=(460,(700,100)),UNIT=SYSDA
//SYSUT2   DD   SPACE=(460,(700,100)),UNIT=SYSDA
//SYSUT3   DD   SPACE=(460,(700,100)),UNIT=SYSDA
//SYSUT4   DD   SPACE=(460,(700,100)),UNIT=SYSDA
//SYSUT5   DD   SPACE=(460,(700,100)),UNIT=SYSDA
//SYSLIB   DD   DSN=S2K.R121.SOURCE,DISP=SHR
//SYSIN    DD   DSN=&&CSOURCE,DISP=(OLD,DELETE)
//SYSLIN   DD   DSN=&&COBOBJ,DISP=(NEW,PASS),
//             SPACE=(80,(900,100)),UNIT=SYSDA
//* -------   CHANGE SYSLMOD TO YOUR LOAD MODULE NAME   ------
//LKED     EXEC PGM=IEWL,
//             PARM='MAP,XREF,LIST,LET,SIZE=(492K,24K)',
//             COND=((4,LT,PRECOM),(8,LT,COB))
//SYSLIN   DD   DSN=&&COBOBJ,DISP=(OLD,DELETE)
//SYSLIB   DD   DSN=SYS1.COBLIB,DISP=SHR
//         DD   DSN=S2K.R121.LOAD,DISP=SHR
//SYSLMOD  DD   DISP=SHR,DSN=S2K.R121.LOAD(FINDORG),
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   SPACE=(1024,(400,20)),UNIT=SYSDA
```

*continued on next page*

**Illus. 20.5** *continued*

```
//* ------------------------------------------------------------
//GO        EXEC PGM=*.LKED.SYSLMOD,
//               COND=((4,LT,PRECOM),(8,LT,COB),(8,LT,LKED))
//STEPLIB   DD   DSN=S2K.R121.LOAD,DISP=SHR
//S2KPARMS  DD   DSN=S2K.R121.CNTL(NLPARM),DISP=SHR
//SYSOUT    DD   SYSOUT=A
//S2KMSG    DD   DCB=(BLKSIZE=1320,LRECL=132,RECFM=FB),
//               SYSOUT=A
//S2KSNAP   DD   SYSOUT=A,DCB=BLKSIZE=882
//SYSUDUMP  DD   SYSOUT=A
//S2KCOMD   DD   DUMMY
//EMPLOYE1  DD   DSN=S2K.R121.EMPLOYE1,DISP=OLD
//EMPLOYE2  DD   DSN=S2K.R121.EMPLOYE2,DISP=OLD
//EMPLOYE3  DD   DSN=S2K.R121.EMPLOYE3,DISP=OLD
//EMPLOYE4  DD   DSN=S2K.R121.EMPLOYE4,DISP=OLD
//EMPLOYE5  DD   DSN=S2K.R121.EMPLOYE5,DISP=OLD
//EMPLOYE6  DD   DSN=S2K.R121.EMPLOYE6,DISP=OLD
//EMPLOYE7  DD   DSN=S2K.R121.EMPLOYE7,DISP=OLD
//EMPLOYE8  DD   DSN=S2K.R121.EMPLOYE8,DISP=OLD
//*
```

## JCL Variations

The JCL given in the previous section is for all four steps. You can omit one or more steps, depending on the nature of the job. For example, the job might consist of only a preprocessing step or only the preprocessing and compilation steps. The PLEX processor execution options described in **COBOL PLEX Processor Execution Options** on page 19-10 also provide for variations in JCL.

The following topics discuss optional DD statements that you can use.

**Additional Locate Files** For a Multi-User environment, the PADnn parameter must be used in the Multi-User initialization job. This parameter provides scratch pads for all Locate Files and scratch files. DD statements for the LOCATE, S2KSYS, and S2KUS files are not used with the PAD parameter. The PADnn parameter is given in the Multi-User initialization job only.

For single-user jobs, you can give the PADnn parameter in the EXEC statement (and no DD statements for the Locate Files or scratch files). Optionally, if the PADnn parameter is omitted in a single-user job, you specify a DD statement for each Locate File and scratch file to be used.

See the *SYSTEM 2000 Product Support Manual, Version 12, First Edition* for complete discussions of the PADnn parameter and the POOLnn parameter.

```
//GO.EMPLOYE1 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE1
//GO.EMPLOYE2 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE2
//GO.EMPLOYE3 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE3        Database files
//GO.EMPLOYE4 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE4
//GO.EMPLOYE5 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE5
//GO.EMPLOYE6 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE6
//GO.LOCATE00 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 1
//GO.LOCATE01 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 2
//GO.LOCATE02 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 3
//GO.LOCATE03 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 4
//GO.LOCATE04 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 5
//GO.LOCATE05 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 6
//GO.LOCATE06 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 7
//GO.LOCATE07 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 8
//GO.LOCATE08 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 9
//GO.LOCATE09 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 10
//GO.LOCATE10 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 11
//GO.LOCATE11 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 12
//GO.LOCATE12 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 13
//GO.LOCATE13 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 14
//GO.LOCATE14 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 15
```

**Additional database files** The Update Log, Rollback Log, Savefile, and Keepfile, as well as database files, will be allocated dynamically by the software if you do not include DD statements for them. The first seven characters of the DDname are the first seven characters of the database name, with Xs replacing embedded (and trailing) blanks. The suffix for the Savefile is S and for the Keepfile is K. Programs that issue KEEP and APPLY commands always use the Keepfile.

**Note**: In a Multi-User environment, DD statements for database files are never included in the JCL for each program.

**Example**

```
//GO.EMPLOYE1 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE1
//GO.EMPLOYE2 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE2
//GO.EMPLOYE3 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE3
//GO.EMPLOYE4 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE4
//GO.EMPLOYE5 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE5
//GO.EMPLOYE6 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE6
//GO.EMPLOYE7 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE7              Update Log
//GO.EMPLOYE8 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE8              Rollback Log
//GO.EMPLOYEK DD DSNAME=DATABASE,UNIT=(TAPE9,,DEFER),       Keepfile
//            DISP=(OLD,PASS),VOL=SER=SAS0002
//GO.EMPLOYES DD DSNAME=DATABASE,UNIT=(TAPE9,,DEFER),       File for
//            DISP=(OLD,PASS),VOL=SER=SAS001                 Savefile
```

**$COPY library** When you use the $COPY directive to copy COMMBLOCK declarations or subschema record descriptions from a library, you must specify the library in a SYSLIB DD statement in the preprocessing step.

**Example**

```
//PRECOM    EXEC    PGM=PRECOMC, . . .
      :
      :
//SYSLIB    DD      DSN=S2K.COPYLIB,DISP=SHR      $COPY LIBRARY
      :
      :
//COBOL     EXEC . . .
```

Chapter 21

# Populating a Database

*PROCEDURES FOR LOADING 21-1*
*Prerequisites for Loading 21-2*
*Specifying records and items 21-2*
*Logical order of records 21-2*
*The loading program 21-2*
*The Input Sequence 21-3*

*LOADING TECHNIQUES 21-6*

*ERROR HANDLING DURING LOADING 21-6*

*RECOVERY DURING THE LOADING PROCESS 21-8*

*TESTING THE LOAD PROGRAM 21-8*

## PROCEDURES FOR LOADING

So far, these chapters have discussed how a database can be accessed and updated using PLEX. But how are data placed in a database? The process of entering logical entries into a database is called populating or loading the database.

Databases are usually populated by PLEX programs using load mode of execution. (Data can also be loaded into a database using the Self-Contained Facility of SYSTEM 2000 software, discussed in *SYSTEM 2000 QUEST Language and System-Wide Commands, Version 12, First Edition*.) This chapter provides guidelines for loading a database using PLEX, based on the EMPLOYEE database defined in the foldout.

Loading a database requires planning. The data records to be inserted need to be collected, edited, and sorted to prepare them for loading. Then you write the PLEX program to insert the data records in the database. Before the program can be executed, the space for the required scratch files must be determined and allocated with JCL.

After you load the database, check for records that were rejected because the data were not acceptable. You need to correct and reinsert these records in the proper place in the data structure. Then you should calculate the overall size of the database. Also, check important data items for accuracy.

## Prerequisites for Loading

The records to be inserted in the database must be edited for accuracy and arranged in logical order before they are loaded. An editing program can arrange the data, and the output can then be used by a PLEX program devoted to loading the records. Dividing the labor between an editing program and a loading program simplifies error detection and error correction.

**Specifying records and items** The preliminary task of editing and arranging the records must be carried out according to the specifications in the database schema. A database schema is composed of schema records organized in a hierarchy. Each schema record denotes a type of data record in the database and specifies the types of items in that record. In other words, the schema specifies the types of data records in the database, the type for each item in the database, and the logical order among records. See *SYSTEM 2000 DEFINE Language, Version 12, First Edition* for information about constructing a schema.

A PLEX loading program specifies the schema records relevant to its processing by including corresponding subschema records, one for each kind of data record to be loaded. The subschema records must contain items corresponding to the data items in the data records. The subschema items must be properly described and checked for compatibility with the item types in the database schema. Also, the subschema item descriptions should match the input data to avoid any loss of data. For example, if the value NINECHARS is moved into a subschema item described as COBOL PIC X(8), the last character is lost.

**Logical order of records** Not only must the input records be edited for data accuracy, but they must also be arranged in logical order. The logical (hierarchical) relationship among data records must conform to the database schema, as mentioned earlier. Using the hierarchical order of records, SYSTEM 2000 software inserts an entire tree of records at one time. The PLEX loading program issues a sequence of INSERT commands indicating each record in the tree to be inserted. If the order of INSERT commands does not correspond to the logical (hierarchical) order of data records, SYSTEM 2000 software rejects one or more of the records to be loaded.

**The loading program** After the editing program edits the data and arranges the input records in logical order, the PLEX loading program is basically concerned with only three tasks:

- reading an input record
- moving the input record to a properly described subschema record
- issuing an INSERT command for that subschema record.

Both the editing program and the loading program must process the new records according to the hierarchical order of records in the database. The next topic explains in detail how to determine the correct order of records to be loaded.

## The Input Sequence

What should the input sequence of records be? In the input sequence, a record can be followed by any of its children, any of its siblings (similar or dissimilar), or a higher-level record. The tables given here suggest possible algorithms for the program that edits and arranges the new records, as well as for the program that issues the INSERT commands to actually load the records into the EMPLOYEE database. The tables show which types of records can follow other types under certain conditions, resulting in an input sequence that is valid for the load program.

The table below shows the most general input sequence for the EMPLOYEE database.

| Record Type Encountered | Record Types That Can Follow in a Valid Input Sequence |
|---|---|
| C0 | C100, C200, C300, C400, C0 |
| C100 | C110, C130, C100, C200, C300, C400, C0 |
| C110 | C120, C110, C130, C100, C200, C300, C400, C0 |
| C120 | C120, C110, C130, C100, C200, C300, C400, C0 |
| C130 | C130, C110, C100, C200, C300, C400, C0 |
| C200 | C100, C200, C300, C400, C0 |
| C300 | C100, C200, C300, C400, C0 |
| C400 | C410, C420, C400, C0, C100, C200, C300 |
| C410 | C410, C420, C400, C0, C100, C200, C300 |
| C420 | C420, C410, C400, C0, C100, C200, C300 |

In the initial loading, if similar siblings are to be inserted one after another, with all records in the order specified by the schema, some of the table rows (marked with an asterisk) can be shortened, as shown below.

| Record Type Encountered | Record Types That Can Follow When Similar Siblings Are Inserted Together |
|---|---|
| C0 | C100, C200, C300, C400, C0 |
| C100 | C110, C130, C100, C200, C300, C400, C0 |
| C110 | C120, C110, C130, C100, C200, C300, C400, C0 |
| C120 | C120, C110, C130, C100, C200, C300, C400, C0 |
| C130 * | C130, C100, C200, C300, C400, C0 |
| C200 * | C200, C300, C400, C0 |
| C300 * | C300, C400, C0 |
| C400 * | C410, C420, C400, C0 |
| C410 * | C410, C420, C400, C0 |
| C420 * | C420, C410, C400, C0 |

Further restrictions might be required concerning which record types are valid following other record types. For example, each logical entry in a database might be allowed to have only one data record corresponding to a particular schema record. In the EMPLOYEE database, schema record C400 is a control node; it contains no schema items. But it forces schema records C410 (EDUCATION) and C420 (TRAINING) to appear at level 2 so that the number of data records at level 1 is reduced. Therefore, when a control node is used, all the education and training information for an employee appears under the same parent data record.

If this policy of allowing only one C400 data record in an entry is followed, the algorithm becomes more restrictive. Boldface record types in the chart show newly disallowed record types. This chart also reflects the shortened rows of the previous chart.

| Record Type Encountered | Record Types That Can Follow |
|---|---|
| C0 | C100, C200, C300, C400, C0 |
| C100 | C110, C130, C100, C200, C300, C400, C0 |
| C110 | C120, C110, C130, C100, C200, C300, C400, C0 |
| C120 | C120, C110, C130, C100, C200, C300, C400, C0 |
| C130 * | C130, C100, C200, C300, C400, C0 |
| C200 * | C200, C300, C400, C0 |
| C300 * | C300, C400, C0 |
| C400 * | C410, C420, **C400**, C0 |
| C410 * | C410, C420, **C400**, C0 |
| C420 * | C420, C410, **C400**, C0 |

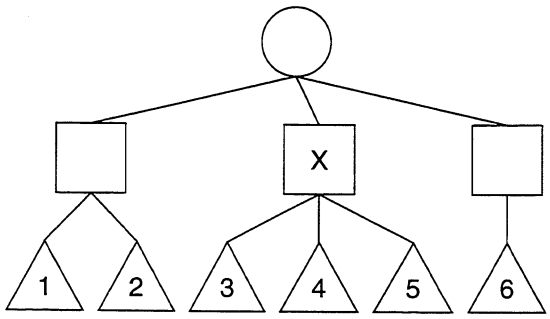
# LOADING TECHNIQUES

PLEX load mode is the most efficient way to load a database. Depending on the volume of data and the availability of machine time, one of three techniques can be used:

- one LOAD/TERMINATE session for all loading
- two or more incremental LOAD/TERMINATE sessions, followed by a REORGANIZE command in the Self-Contained Facility
- either a single LOAD/TERMINATE session or several incremental LOAD/TERMINATE sessions, with all items defined as non-key in the database schema, followed by SCF CREATE INDEX commands that change non-key items to key items as necessary.

# ERROR HANDLING DURING LOADING

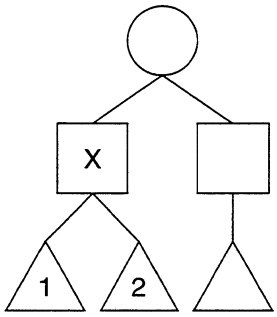
Be sure to check the return code after each INSERT command. A return code other than 00 or 50 indicates a problem; a record has been rejected and not inserted. Because the insertions must be done in hierarchical order, the program logic must be written to reject or bypass any descendants of the rejected record. For example, suppose you have the following tree:




If record X is rejected, the program should bypass triangles 3, 4, and 5. That is, it should not give INSERT commands for them. Instead, it should write the rejected record and triangles 3, 4, and 5 to an error file for later correction and reentry. If triangles 3, 4, and 5 are not bypassed, they will be inserted as descendants of the first square instead of the intended parent.

The non-zero return codes most frequently received for an INSERT command are 03, 10, 17, and 47. A brief description of these return codes follows. Also, see the discussion of return codes in **PLEX Return Codes** on page 22-17.

**03** means the stack lacks an appropriate entry. This occurs, for example, if a previous INSERT command for the parent was rejected. Suppose you have the following tree:




If record X is rejected, triangles 1 and 2 would also be rejected, and return code 03 would result in both cases.

**10** means a value in the subschema record is not compatible with the type of the corresponding item in the database schema. For example, if a subschema item corresponds to a schema item of type DATE, but the value of the subschema item is not in a valid date format, return code 10 results. In this situation, the eighth COMMBLOCK field indicates the ordinal place of the subschema item whose value is incorrect.

**17** means the picture or usage (rather than the value) of a subschema item is incompatible with the size or item type of the corresponding item in the database schema. Again, the eighth COMMBLOCK field indicates the ordinal place of the subschema item incorrectly described.

**47** means an item lacks U-authority.

# RECOVERY DURING THE LOADING PROCESS

To support a standardized recovery, you should save the database (using the SAVE command) prior to entering load mode. This is true even if the database contains no data, only a definition.

During the load, as each INSERT command is processed, data are written to database Files 5 and 6 and Scratch File 5. When the software executes your TERMINATE command, it sorts Scratch File 5 and uses it to build the index of key items. If an abend occurs before TERMINATE is issued, it causes the data written on database Files 5 and 6 to be logically canceled when the database is next opened. Consequently, the load program can be run again without any danger of duplicate data being loaded.

If an abend occurs after TERMINATE initializes Index updating (but before updating is completed), the database is damaged. Do not run the load program again until the damaged database is restored.

The database can be restored in either the Self-Contained Facility (SCF) or PLEX. SYSTEM 2000 software notifies the SCF user of a damaged database by a message when that database is next opened in SCF. For the PLEX user, a damage flag is set in the 14th COMMBLOCK field. In either situation, the damaged database can be easily restored with a few SCF commands. (These save the PLEX user from having to write a program.)

Restore a database with the following Self-Contained Facility commands:

**USER,** *password*: **DBN IS** *database-name*:

**CONTROL:**

**RELEASE:** (*releases the damaged database*)

**RESTORE** *database-name*:

**EXIT:**

The load program can now be rerun. The restore commands are also available to PLEX programs.

# TESTING THE LOAD PROGRAM

Try to test the load program with a subset of the data to be loaded. Fifty logical entries (level 0 records and all of their descendants) is usually a good number. When the logical entry count reaches 50, you can issue the TERMINATE command. Loading a subset of the data provides an opportunity to verify the integrity of the load program and to extrapolate a time estimate for the full load. Obtain two timings from the small load: the processing time spent before the TERMINATE command and the time spent after the TERMINATE command is issued.

For the complete load, the time before TERMINATE increases proportionately with the increase in data, but the time after TERMINATE increases much more due to the increased number of records being passed to the internal sort.

You can verify the integrity of the load program by interrogating the loaded database with the following Self-Contained Facility commands:

**USER,** *password*: **DBN IS** *database-name*:

**CONTROL:**

**PRINT SIZE:**

**QUEST:**

**TALLY/EACH/***key-item, key-item, . . . :*

*(Issue a TALLY EACH command for key items that have relatively few distinct values.)*

**TALLY/ALL/***key-item, key-item*

*(Issue a TALLY ALL command for key items that have many distinct values.)*

**PRINT COUNT C0, COUNT** *SR*, **COUNT** *SR*, **COUNT** *SR* , . . . :

*(Count the number of data records for each schema record. Are the counts appropriate?)*

**LIMIT 0,2/TRUNCATE:**
**PRINT C0 WHERE C1 EQ** *value1* * *value2*:

*(Are the output values appropriate from this command, limited to two complete logical entries? Notice that EQ is used as a range operator here; it spans two C1 values that were loaded into the database in the current run. The LIMIT capability can be used with other PRINT, LIST, or TALLY commands to do other validity checks appropriate for a given database.)*

**EXIT:**

Chapter 22

# Usage and Guidelines

*EFFICIENCY CONSIDERATIONS IN PLEX 22-2*
*Process Only the Necessary Data 22-2*
*Minimize the Number of SYSTEM 2000 calls 22-3*
*Consider the HAS Operator and Its Alternatives 22-4*
*Consider Other Ways to Simplify a Where-Clause 22-6*
*Consider the No-Sort Capability 22-6*
*Use the Fifteenth COMMBLOCK Field When Necessary 22-6*

*EXECUTION-TIME PLEX EXITS 22-7*

*CICS COBOL II COMMAND-LEVEL LINKAGE REQUIREMENTS 22-7*
*SYSTEM 2000 Control Blocks 22-7*

*PSEUDO-CONVERSATIONAL CICS TRANSACTIONS 22-9*
*Programming Requirements 22-10*
*Resources Required in the CICS PLEX Program 22-11*
*SYSTEM 2000 Resources Required 22-11*
*Potential Problem Areas 22-12*

*PROGRAMMING FACILITY MESSAGES 22-13*
*Introduction to Types of PLEX Messages 22-13*
*Guidelines for Avoiding Errors in PLEX 22-13*
*PLEX Processor Messages 22-14*
*Format of Long and Short Messages 22-14*
*Severity Codes 22-15*
*Sample Short-Form Messages 22-15*
*Cursor Pointing to Error in Program Listing 22-15*
*Memory Utilization during the PLEX Processor Step 22-16*
*PLEX Return Codes 22-17*
*Sample PLEX return codes 22-18*
*SCF Messages during PLEX Program Execution 22-18*
*Abnormal Job Termination 22-18*

# EFFICIENCY CONSIDERATIONS IN PLEX

Many efficiency considerations are pointed out in the individual reference sections for each command. This section presents some of the most important ones.

- Process only the necessary data.
- Minimize the number of SYSTEM 2000 calls.
- Consider the HAS operator and its alternatives.
- Consider other ways to simplify a where-clause.
- Consider the no-sort capability.
- Use the fifteenth COMMBLOCK field when necessary.

## Process Only the Necessary Data

For retrievals and updates, PLEX provides ways to prevent the conversion of unneeded data from database storage format to subschema record storage format and vice-versa. This saves CPU time, although not I/O. The items to be retrieved or updated can be restricted either by including a subschema record with only the needed items or by using an inclusion list in a retrieval or update command.

## Minimize the Number of SYSTEM 2000 calls

### Example 1

In the EMPLOYEE database, each level 0 (ENTRY-REC) record contains a unique EMPLOYEE NUMBER (EMPNUM). To produce a report showing the current position (POSITION-REC) of a specific employee, issue the following commands:

```
GET1 ENTRY-REC WHERE EMPNUM EQ VARIABLE.
GETD POSITION-REC NEXT.   (if current position is first descendant)
```

These commands retrieve the most current POSITION-REC record for the employee, that is, the first descendant. However, the following command accomplishes the same task with just one SYSTEM 2000 call:

```
GET1 POSITION-REC WHERE EMPNUM EQ VARIABLE.
```

### Example 2

This example illustrates two methods for requesting specific level 0 records. Method 1 uses LOCATE and GET.

```
LOCATE level-0 WHERE . . .  (80% of the database qualifies)

GET level-0 NEXT.
```

Method 2 uses GET1 with the NEXT option.

```
GET1 level-0 NEXT.

(followed by a programming language test for appropriate values in the level 0 records.)
```

With the I/O to the Hierarchical Table and Data Table being approximately equal for the two methods, the I/O against the Index to process the where-clause is an extra cost in the first method.

In general, the second method is more efficient. However, it loses its efficiency if the logical order of records is not close to the physical order. This discrepancy between logical and physical order is a function of the effect of updates.

## Consider the HAS Operator and Its Alternatives

Different techniques can be used to retrieve records based on conditions in two different schema paths. For this example, portions of the EMPLOYEE database schema and of a logical entry appear below, along with the corresponding subschema records declared by the program.

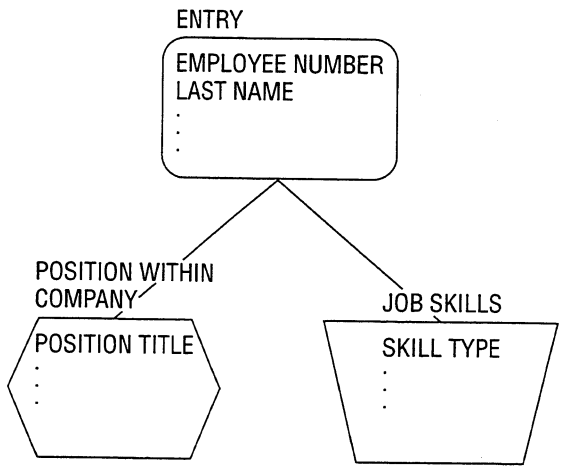
**Subschema Records**

```
SSR  ENTRY-REC OF EMPLOYEE.
01   ENTRY-REC.
       03  EMPNUM...
           EQ EMPLOYEE NUMBER.
       03  LAST-NAME...
           EQ LAST NAME.
       .
       .
       .

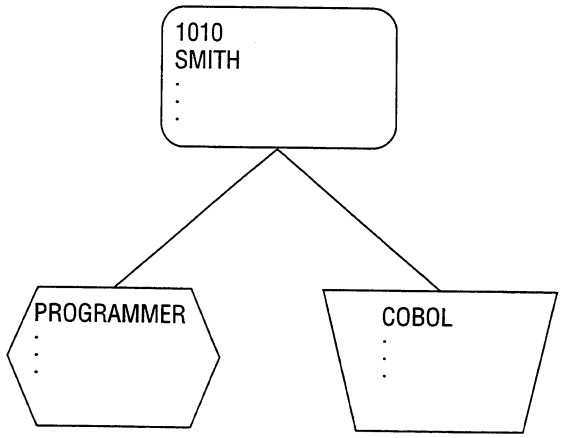
SSR  POSITION-REC OF EMPLOYEE.
01   POSITION-REC.
       03  TITLE...
           EQ POSITION TITLE.
       .
       .
       .

SSR  SKILL OF EMPLOYEE.
01   SKILL.
       03  SKILL-TYPE...
           EQ SKILL TYPE.
       .
       .
       .
```

**Schema**




**Logical Entry**



A report is needed showing programmers who know COBOL. You could issue a LOCATE command with this where-clause using HAS and the following variables:

```
MOVE 'COBOL' TO TYPEVAR.
LOCATE ENTRY-REC WHERE ENTRY-REC HAS TITLE EQ TITLEVAR
AND ENTRY-REC HAS SKILL-TYPE EQ TYPEVAR.
```

An alternative method uses a simpler where-clause in the LOCATE command. It searches for ENTRY-REC records that have appropriate SKILL descendant records, then issues a series of GETD commands to check for appropriate POSITION-REC descendant records.

```
LOCATE ENTRY-REC WHERE SKILL-TYPE EQ TYPEVAR.
     .
     .
GET ENTRY-REC NEXT.
     .
     .
GETD POSITION-REC NEXT.
IF RETURN-KODE IS = 04 . . .  (Go back to GET ENTRY-REC NEXT.)
IF TITLE IS = TITLEVAR
(Print the ENTRY-REC record)  (Then go back to GET ENTRY-REC NEXT.)
     .
     .
   ELSE . . .                 (Go back to GETD POSITION-REC NEXT.)
```

If other data are needed from the SKILL record, the command sequence would include a GET1 PRESENT command, as shown below.

```
LOCATE SKILL WHERE SKILL-TYPE EQ TYPEVAR.
     .
     .
GET SKILL NEXT.
     .
     .
GET1 ENTRY-REC PRESENT.
     .
     .
GETD POSITION-REC NEXT.
IF RETURN-KODE IS = 04 . . .  (Go back to GET SKILL NEXT.)
IF TITLE IS = TITLEVAR
(Print the ENTRY-REC record)  (Then go back to GET SKILL NEXT.)
     .
     .
   ELSE . . .                 (Go back to GETD POSITION-REC PRESENT.)
```

## Consider Other Ways to Simplify a Where-Clause

In many situations, you can increase efficiency by transferring the responsibilities of the where-clause to other retrieval commands. In some cases you can shorten a where-clause or eliminate it. For example, as an alternative to a where-clause condition specifying a value or range of values, you could issue retrieval commands such as GET1 or GETD followed by COBOL statements to check for the desired value(s) in retrieved records.

Another alternative would be using the AT operator to specify the position of a descendant record under a given record. With a retrieval command, the program can retrieve the given record and issue a GETD command with FIRST, LAST, or S2KCOUNT, depending on the desired position of the descendant to be retrieved.

## Consider the No-Sort Capability

At times, you might need to produce a report in some sorted order. The ORDER BY command sorts data record addresses in a Locate File into a requested order. When the where-clause in a LOCATE command consists of a single key condition and the report is to be in sequence by that single key item, the ORDER BY command should be omitted.

Suppose a report must be in sequence by employee number. Subschema item EMPNUM in SSR ENTRY-REC corresponds to schema item EMPLOYEE NUMBER.

```
LOCATE ENTRY-REC WHERE EMPNUM EXISTS.
ORDER BY EMPNUM.
```

The condition EMPNUM EXISTS is processed in the Index. The values for EMPNUM are retrieved in ascending order. The where-clause processor normally performs a sort of the addresses in the Locate File into physical order. In this example, the physical-order sort can be avoided by setting the no-sort flag. This flag is the variable S2KSRT, which is part of the S2KDUM control block. When S2KSRT is set equal to 1 under the conditions just described, the physical-order sort does not take place. For more information about the no-sort capability, see Appendix A.

## Use the Fifteenth COMMBLOCK Field When Necessary

The fifteenth COMMBLOCK field is optional. It indicates the parent of the data record most recently retrieved. The value in this field is the address of the parent record and is not meaningful by itself, but the value can be used to advantage when the program is retrieving a series of similar records with GET or GETD commands. If the program is concerned with associating similar records according to their parents, it can order descendant records by the parent schema record, then check this field for a change in value, which signifies a change in parent. Repeated checking of this field saves repeated retrievals of the parent by GETA commands. Thus, the program retrieves parent records only if it actually needs the data in those records.

# EXECUTION-TIME PLEX EXITS

Execution-time exits can be linked into S2KPLR. Dummy exits are provided in the standard, distributed copy of S2KPLR to avoid unresolved external references. These can be replace-linked with user-written routines.

**S2KEX0** is a fullword containing the number of additional bytes of main memory, in binary, required by the other two executable exits. S2KPLR is a reentrant module, which uses GETMAIN to obtain new working storage for each new user. The exits must conform to reentrant requirements if S2KPLR is to retain its reentrancy. S2KPLR dynamically obtains memory for the exits as requested in S2KEX0. S2KEX0 as provided by the Institute has a value of zero.

**S2KEX1** is given control via conventional linkage before each command is passed to SYSTEM 2000 software. Register 1 contains the address of a list of addresses. The first address in the list is that of the S2KDUM vector, the second is that of the COMMBLOCK (if any), and the third is that of the subschema record (if any). Registers must be saved and restored according to standard conventions.

**S2KEX2** is given control after each command is processed by SYSTEM 2000 software. Conventions are the same as those for S2KEX1.

# CICS COBOL II COMMAND-LEVEL LINKAGE REQUIREMENTS

Linking two PLEX programs requires the calling program to pass to the called subprogram addresses of the SYSTEM 2000 control block S2KDUM and optionally any COMMBLOCKs and subschema records. Prior to CICS 3.3, you had to use the EXEC CICS LINK command. The LINK command had one severe limitation -- you could only pass one address via the COMMAREA option.

As documented in the CICS 3.3 Application Programming Guide, COBOL II can use a more efficient CALL statement that does not have the single address restriction. This technique is discussed below.

## SYSTEM 2000 Control Blocks

**S2KDUM** control block is the most important control block because it has the address of the interface User Request Block (URB). Without this address, the interface issues a PLEX return code 44 because it does not know a START S2K has been executed.

**PLXCOMM** control block is required by the command-level interface. It is generated immediately after the S2KDUM control block.

**PLEX processor generated variables** are constants required by PLEX to do where-clause, ORDER BY, and other commands. They have names like S2R1, S2R2, S2A1, S2A2 ... These are local variables only and may never be passed. The called subprogram generates its own.

**COMMBLOCK** contains information about the database. The database open process stores the internal database number for future references. Although SYSTEM 2000 software

reinitializes this control block if it was not passed, certain commands do require an "initialized" COMMBLOCK. The CLOSE command gets a PLEX return code 26 with an un-initialized COMMBLOCK. Commands that force an INSERT sequence get a PLEX return code 15. These two problems can be avoided if you CLOSE the database in the same program that OPENed it and force INSERT sequences before returning to the calling program.

**SUBSCHEMA** control blocks contain information about individual items within a database. Although there are no problems or potential PLEX return codes if you do not pass the SUBSCHEMA control blocks, there is additional overhead incurred to open the SUBSCHEMA. This overhead could affect overall performance if you have a large number of subschema items. Additional table lookup is required if you use component names and you do not use the EQ options to equate the item to the component number.

The following steps describe how a PLEX CICS COBOL II program calls another PLEX CICS COBOL II subprogram:

1. In the calling program, code all the SYSTEM 2000 control blocks in the WORKING-STORAGE SECTION. Use $BLOCKS to generate S2KDUM and PLXCOMM, COMMBLOCK to describe the database, and SUBSCHEMA to describe the records.

   The CALL statement in the calling program has one restriction: the first two parameters must be DFHEIBLK and DFHCOMMAREA. After that, code any SYSTEM 2000 control blocks and other additional information required by the "called" subprogram. Here is a sample call statement:

```
CALL 'CALLED' USING DFHEIBLK DFHCOMMAREA
     S2KDUM PLXCOMM LIBRARY-0 BOOKS-0 SUBJECT-0 AUTHORS-0.
```

   **Note:** The COMMBLOCK and SUBSCHEMA record 01-level names are suffixed with a dash zero (-0).

2. In the called subprogram, all passed control blocks and any additional information are defined in the LINKAGE SECTION. When $S2KDUM is coded in the LINKAGE SECTION, S2KDUM and PLXCOMM are generated without the VALUE clauses. Also code all necessary COMMBLOCKs and SUBSCHEMAs in the LINKAGE SECTION. As discussed above, you may code COMMBLOCKs and SUBSCHEMAs in the WORKING-STORAGE SECTION, but remember they are "re-opened" by SYSTEM 2000 software.

   Code $BLOCKS in the WORKING-STORAGE SECTION. Any necessary PLEX processor generated variables are placed here.

   The PROCEDURE DIVISION statement in the called program must specify the control blocks in the same order specified in the calling program. Here is a sample PROCEDURE DIVISION statement.

```
PROCEDURE DIVISION USING DFHEIBLK DFHCOMMAREA
     S2KDUM PLXCOMM LIBRARY-0 BOOKS-0 SUBJECT-0 AUTHORS-0.
```

   When you return control back to the calling program, use a GOBACK instead of an EXEC CICS RETURN. Use of RETURN in the called subprogram terminates the calling program.

3. This call can be either static or dynamic. For static calls, the called subprogram is link-edited with the calling program. For dynamic calls, the called subprogram must be defined in the CSD.

# PSEUDO-CONVERSATIONAL CICS TRANSACTIONS

There are three ways to write a SYSTEM 2000 PLEX program in CICS:

- transactional
- conversational
- pseudo-conversational

Each way has advantages and disadvantages.

Transactional CICS transactions are totally self-contained; each transaction begins with one terminal input operation and ends with one terminal output operation. For a transactional PLEX operation, the first SYSTEM 2000 command is START S2K and the last command is STOP S2K.

These are the advantages of transactional programming:

- The program is easier to design, code, and maintain.
- The program does not depend on previous processing.
- No resources are tied up during operator "think time" in either Multi-User or CICS.

Conversational CICS transactions mean multiple terminal input and output operations. The program does not give up control, but it must wait for your next input. The advantages of conversational CICS transactions are that the program is always in control and CICS and Multi-User resources do not have to be re-obtained. The main disadvantage is all the resources required by CICS during operator "think time."

The CICS resources required are

- one copy of the program. Only one copy is needed no matter how many users are using it.
- a separate area for WORKING-STORAGE.
- memory for save areas.
- memory for the Terminal Input/Output Area.
- memory for any optional CICS facilities, such as File Control, Temporary Storage, BMS, Storage Control, and so forth.

Pseudo-conversational transactions combine the transactional and conversational methods. In the CICS environment, the transaction is transactional since the CICS resources are free while CICS waits for the next terminal input. In the Multi-User environment, the transaction is conversational since the Locate Files, stack positions, and local holds are still intact while the data are being displayed on the terminals.

The advantages of pseudo-conversational transactions are that fewer CICS resources are consumed, and SYSTEM 2000 software does not have to reprocess where-clause operations or rebuild any stack positions.

The main disadvantage is that more burden is placed on the application programmers. The code must preserve the SYSTEM 2000 control blocks (for example, save the S2KDUM control block in CICS temporary storage) and must reestablish the SYSTEM 2000 environment upon returning to the program. Other disadvantages are as follows:

- Programs are harder to design, code, and maintain.
- The program must keep track of previous processing.
- All SYSTEM 2000 resources (Multi-User tables, buffers, local holds, and so on) are tied up during operator "think time."
- A CICS interface URB (User Request Block) is required for the duration of the session.
- More resources are consumed because they cannot be shared among users.

## Programming Requirements

In order to be pseudo-conversational, your PLEX program must contain code to perform the following tasks:

- Turn on this feature with a START S2K USE(2) command. USE(2) means a pseudo-conversational request, and the CICS interface should not automatically issue a STOP S2K command upon task termination.

  The CICS interface validates the URB for the user according to whether an address exists in the S2KDUM control block. If it does, that URB must belong to the terminal ID. If not, that terminal ID must not already have a URB.

- Save and restore the SYSTEM 2000 S2KDUM control block and the appropriate COMMBLOCKs and SCHEMAs.
- Issue the STOP S2K command to explicitly free the resources.

**Note**: FOR return code checking is not allowed in pseudo-conversational applications.

The program must also include code that checks the status of processing, for example,

- whether this is the first time through the application program
- where to continue processing other than the first time through the program
- whether this is the last time through the application program
- which stack and which Locate File are being used.

## Resources Required in the CICS PLEX Program

In the CICS PLEX program, SYSTEM 2000 software and the CICS interface rely on having internal information in the following kinds of control blocks: S2KDUM, COMMBLOCKs, and SCHEMAs.

S2KDUM is perhaps the most important control block. It contains the address of the CICS URB and other information that is set at initialization time and must be preserved throughout the session. Therefore, the application program must ensure that S2KDUM is saved and restored for each CICS transaction. The Multi-User software needs the information in S2KDUM when trying to associate a CICS transaction that is restarting with a user who is already active.

The COMMBLOCKs and SCHEMAs also have internal information, such as the internal database number and internal component numbers that are resolved the first time these blocks are used. If the program does not save and restore these control blocks, SYSTEM 2000 software must go through the overhead of reinitializing this information for each transaction.

The sizes of the control blocks are as follows:

| | |
|---|---|
| **S2KDUM** | **152 bytes** |
| **COMMBLOCK** | **124 bytes** |
| **SCHEMA** | **user-defined data - variable, calculated as (56 + (56 * number of components) + summation length of each component)** |

## SYSTEM 2000 Resources Required

For pseudo-conversational support, SYSTEM 2000 software needs a PLEX CICS User Request Block (URB), which is tied up for the duration of the session. Also, Multi-User software requires an SVC slot, a communications block, a data area for each database opened, and buffers for each Locate File. The space for these resources is as follows:

| | |
|---|---|
| **PLEX CICS URB** | **840 bytes** |
| **SVC slot** | **36 bytes** |
| **Multi-User Communications Block** | **6400 bytes** |
| **Each database opened** | **8704 bytes (approximately)** |
| **Each Locate File** | **1 scratch file buffer plus scratch pad space** |

## Potential Problem Areas

Consider the following problem areas when coding your pseudo-conversational PLEX program:

- What resources are tied up if a user decides to take a break? If the user has a /HOLD in effect for a record, another user who wants that record is locked out until the first user releases the hold. Subsequent users receive PLEX Return Code 110. The /HOLD option can be even more troublesome if the multiple local holds option is enabled.

- If rollback is enabled, you should consider when to issue the COMMIT command to ensure that an update is complete.

- If rollback is not used, the database could become damaged if the system were to crash without the database buffers having been cleared. Also, a series of INSERT commands might not terminate properly.

- Consider user "think time," for example, a session lasting hours compared with a transaction lasting only seconds.

- Abnormal terminations may increase because of the resources needed to support pseudo-conversational processing. Some of the causes for termination are as follows:

| | |
|---|---|
| PLEX Return Codes | 84 - improper activity status |
| | 86 - SVC table full |
| | 89 - START S2K and a URB was already assigned. |
| | 110 - request for a record held by another user |
| SYSTEM 2000 Error Codes | 1 - not enough buffers defined |
| | 41 - buffer lockout |
| | 42 - buffer lockout |
| | 43 - database lockout |
| | 47 - scratch pad/allocation unit lockout |
| User Abends | 006 - BLOCKS execution parameter is insufficient |
| | 105 - GETMAIN failed |
| | 536 - SVC table full |

# PROGRAMMING FACILITY MESSAGES

## Introduction to Types of PLEX Messages

The SYSTEM 2000 Programming Facility consists of the Programming Language Extension (PLEX) processor, which is used for preprocessing PLEX programs, and the Program Service Processor (PSP), which is the interface between the PLEX program and the SYSTEM 2000 Database Manager. Four types of messages and codes are issued when using PLEX:

- messages issued from the PLEX processor during preprocessing
- PLEX return codes issued to the PLEX program during execution of the program
- SCF messages sent to the S2KMSG file associated with the PLEX program during program execution
- PLEX processor internal abends. (For assistance with these errors, contact your SYSTEM 2000 Software Representative.)

This section contains overviews of the first three types of messages, along with guidelines for avoiding PLEX errors. See *SYSTEM 2000 Messages and Codes, Version 12, First Edition* for the messages and their meanings, as well as PLEX program abends, SYSTEM 2000 Error Codes, and IBM user abend codes.

## Guidelines for Avoiding Errors in PLEX

Following these guidelines helps to avoid errors in PLEX processing:

- Does each COMMBLOCK reference the correct SYSTEM 2000 database name?
- Is each COMMBLOCK properly declared with 14 (or optionally 15) fields?
- Is there a subschema record for each data record to be retrieved or modified by the program?
- Does the subschema record contain items from only one schema record? Are there any duplicates?
- Does the picture of each item in a subschema record conform to the picture in the database definition?
- Does the program verify that the database is undamaged prior to attempting the first access?
- Is there a valid password in the COMMBLOCK prior to the OPEN command?
- Does the program check the return code in the appropriate COMMBLOCK after each PLEX command? Was a FOR command issued?
- Were all possible non-zero return codes that can result from a normal flow of the program considered? Is proper action being taken for all non-zero return codes?

- Could null items cause erroneous arithmetic operations?
- Are key conditions to be treated as NON-KEY indicated as such?
- Is the CLOSE command issued for each database prior to program termination?
- Is the first executable PLEX statement a START S2K command?
- Are LINK structures consistent with program logic? Do forced continuations override error controls?
- Are inclusion lists correctly valued? Are excluded values recognized as unchanged?

## PLEX Processor Messages

PLEX processor messages indicate errors in the syntax of PLEX directives and commands. These messages are issued during the preprocessing of a PLEX program, not during program execution. The messages and their meanings are discussed in *SYSTEM 2000 Messages and Codes, Version 12, First Edition.*

Each message has a long and a short form. The formats for long and short messages differ somewhat.

**Format of Long and Short Messages** The long form is the default format:

**PLX***nnnnn* **PLX***mmm* *message-text*

*nnnnn* is the program line number where the error occurs.

*mmm* is the message number.

The short-form message is intended for use at a terminal and can be requested by specifying the PLEX processor execution option TERM, discussed in **TERM and NOTERM Execution Option** on page 19-20. The format of the short-form message is as follows:

**PLX***nnnnn* **PLX***mmm* *short-message* **AT** *col*

*nnnnn* is the program line number where the error occured.

*mmm* is the message number.

*col* is the character position (within the line) where the error occured.

The message summary is followed by two messages:

```
NUMBER OF DIAGNOSTICS GENERATED IS k
HIGHEST SEVERITY CODE IS    j
```

*k* is the number of messages generated.

*j* is the highest severity code, to be interpreted as discussed in the next topic.

**Severity Codes** A symbol designating the severity of the message appears in the description of each PLEX processor message. The severity code (integer) corresponding to each symbol is shown below. The error summary at the end of the output listing indicates the highest severity code that occurred while using the PLEX processor.

| Symbol | Code | Severity: Condition |
|---|---|---|
| W | 4 | Warning: proper execution likely |
| E | 8 | Error: proper execution unlikely |
| S | 12 | Severe error: execution impossible |
| T | 16 | Terminal: execution impossible, PLEX processor unable to proceed. |

**Sample Short-Form Messages** The last two lines pertain to memory utilization, which is discussed in **Memory Utilization during the PLEX Processor Step** on page 22-16.

```
SYSTEM 2000 - PLEX PROCESSOR - R12

PLX00029 PLX202 LEVELS INCREASED     AT 12
PLX00047 PLX202 LEVELS INCREASED     AT 12
PLX00050 PLX202 LEVELS INCREASED     AT 12
PLX00054 PLX202 LEVELS INCREASED     AT 12
PLX00062 PLX202 LEVELS INCREASED     AT 12
PLX00072 PLX202 LEVELS INCREASED     AT 12
PLX00237 PLX407 SSR NOT FOUND        AT 12
PLX00254 PLX362 ITEM NOT IN SSR DEF AT 40
NUMBER OF DIAGNOSTICS GENERATED IS  8
HIGHEST SEVERITY CODE IS            8

498K WAS NOT USED FOR SSR=004 AND ITEM=005
498K WOULD BE FREE IF SSR=005 AND ITEM=005
```

**Note**: PLEX processor messages are written to the SYSPRINT file.

**Cursor Pointing to Error in Program Listing** If an error occurs, the PLEX program listing shows a cursor immediately below the command containing the error. The error message (in long or short form) is printed below the cursor. The cursor line is a series of dots (or another symbol designated by the ERRSYM execution option) ending with the letter C, which indicates the point of error.

For example, if subschema record L8 was not declared previously or was coded incorrectly, the cursor and message appear as shown below. (PLX407 is the message number.) The error message is repeated in the summary of errors at the end of the program listing.

```
*GET1 L8 FIRST.
....C
   PLX407 SSR NOT FOUND
```

**Memory Utilization during the PLEX Processor Step** The PLEX processor issues two messages that appear after any other messages in the SYSPRINT file.

```
X1K WAS NOT USED FOR SSR=00N1 AND ITEM=00M1
X2K WOULD BE FREE IF SSR=00N2 AND ITEM=00M2
```

The X1, N1, M1, X2, N2, and M2 variables are replaced in the message text with information about main memory use during the PLEX processing step. By altering the PLEX processor execution parameters SSR and ITEM, you might be able to use this information to reduce the amount of main memory required by the PLEX processor. These parameters do not affect run-time main memory requirements of PLEX programs.

The first message indicates the amount of main memory that was provided for the PLEX processor but not needed with the SSR and ITEM values that were specified or assigned by default. For example, suppose the PLEX processor is executed in an OS/MVT environment with REGION=128K provided in the JCL. The resulting message is as follows:

```
12K WAS NOT USED FOR SSR=008 AND ITEM=012
```

The region request can be changed from 128K to 116K without changing the execution parameters. If the message results from an OS/MFT or VS1 run, a smaller partition can be used.

The situation becomes more complex in an MVS environment. The PLEX processor requests up to 500K for SSR and ITEM tables after main memory was obtained for access methods, buffers, operating system overhead, and the PLEX processor itself.

MVS uses the region requested in the JCL to limit the amount of virtual memory obtainable by the PLEX processor, but the amount requested is rounded up to the next multiple of 64K bytes. In other words, REGION=128K provides 128K in MVS, but REGION=129K in MVS provides 192K. There would be no reason to specify REGION=116K instead of 128K in MVS because in both cases 128K is provided by the operating system and used by the PLEX processor. However, if the PLEX processor issues a message indicating that 64K or greater was not used, some savings in virtual memory can be achieved by reducing the amount requested in the region parameter.

The second message recommends optimal SSR and ITEM parameter values. Even if the SSR and ITEM parameters used and displayed in the first message are already optimal, the second message is issued. Likewise, both messages are issued even if there is no memory to be saved.

Consider a PLEX processor job performed in a VS1 partition. Suppose the following messages appear:

```
400K WAS NOT USED FOR SSR=004 AND ITEM=005
406K WOULD BE FREE IF SSR=001 AND ITEM=002
```

The PLEX processor obtains, as always, an amount of main memory less than or equal to 500K bytes for SSR and ITEM tables. Of this amount, 400K was not used. Without changing the SSR and ITEM parameter values, the PLEX processor step for this program could be run in a partition that is 400K smaller, in which case the PLEX processor would produce the following messages:

```
0K WAS NOT USED FOR SSR=004 AND ITEM=005
6K WOULD BE FREE IF SSR=001 AND ITEM=002
```

If the SSR and ITEM parameter values are changed as recommended and the PLEX processor is executed again in the smaller partition, the following messages appear:

```
6K WAS NOT USED FOR SSR=001 AND ITEM=002
6K WOULD BE FREE IF SSR=001 AND ITEM=002
```

There are 6K bytes unused, so that the smallest partition possible would be 6K smaller than the present one. If the PLEX processor is run again in the smallest possible partition, the following messages appear:

```
0K WAS NOT USED FOR SSR=001 AND ITEM=002
0K WOULD BE FREE IF SSR=001 AND ITEM=002
```

Such an exact fit is not necessarily achievable in an MVS environment because virtual memory is managed in 64K units, but it is possible to use these messages to avoid leaving more than 64K unused.

## PLEX Return Codes

SYSTEM 2000 software sends return codes back to a PLEX program during program execution. Each time the program executes a PLEX command, the return code reflects the success or failure of the requested operation. A return code of 00 means that SYSTEM 2000 software completed the request successfully. Other non-zero return codes inform the program of the status of the database, errors in data values, possible errors in navigation through the database, and other information that can be used in the program logic. Your program should check the return code after each PLEX command. Also, you can code FOR processing to dynamically test for ranges of return codes and to branch to appropriate code as necessary. The meanings of all PLEX return codes are given in *SYSTEM 2000 Messages and Codes, Version 12, First Edition*.

A PLEX return code is a four-digit number. Current return codes are in the range of 0000 through 0127. These can be used in FOR processing. Return codes 0900 through 0999 pertain to user exits and must be checked individually.

PLEX return codes are sent from SYSTEM 2000 software to the second field of the COMMBLOCK of the database that a PLEX command referenced. Some return codes are sent to the S2KRTC field of the S2KDUM array, for example, those pertaining to the LINK command and to general commands that do not refer to a specific database. Also, for some return codes, additional information about the error is passed back to the program in various other COMMBLOCK fields.

### Sample PLEX return codes

Here are some sample return codes.

| Return Code | Meaning |
|---|---|
| 0000 | PLEX command completed successfully |
| 0004 | End-of-data |
| 0010 | Error in data |
| 0016 | Update ignored due to damaged database |

## SCF Messages during PLEX Program Execution

During the execution of a PLEX program, some Self-Contained Facility (SCF) messages might be sent to the S2KMSG file, which is associated with every PLEX program. These SCF messages can occur because a PLEX program uses several of the processors controlled by the Database Manager. SCF messages and their meanings are listed in *SYSTEM 2000 Messages and Codes, Version 12, First Edition*.

## Abnormal Job Termination

A PLEX program could abend because of logic errors, hardware or software problems, a SYSTEM 2000 Error Code, or an IBM user abend code. For descriptions of all messages and codes pertaining to abnormal job termination, see *SYSTEM 2000 Messages and Codes, Version 12, First Edition*.

# APPENDICES

The No-Sort Capability

PL/I PLEX

FORTRAN PLEX

Assembler PLEX

Appendix A

# The No-sort Capability

The performance of PLEX where-clause processing can be appreciably improved in certain cases by no-sort processing. This appendix explains how a PLEX program can use the no-sort capability.

Processing a where-clause in a LOCATE command results in a Locate File containing addresses of the selected data records. Normally, these addresses are sorted according to the physical order of the data records. With the Locate File addresses sorted in this way, the data records can be accessed sequentially in physical order through a sequence of GET NEXT commands. When records are retrieved in physical order, no database table page is read more than once, so disk head movement and I/O are kept to a minimum.

Sorting addresses in physical order enables efficient data access, not only for a series of GET NEXT commands but also for a subsequent ORDER BY command, since data values must be retrieved before ORDER BY sorting takes place. However, in some applications, retrieval of data ordered on the same item as in the where-clause might be what the program requires for its sequence of GET commands. In these cases, the LOCATE and GET commands all refer to the same schema record and, in addition, the where-clause consists of a single key condition for that same schema record. Access to records during the processing of a key condition follows the ascending order of distinct values in the Index for the key item in the condition. Consequently, the resulting list of addresses selected for the key condition is arranged in the original order of access, that is, the ascending order of distinct values. This list constitutes the Locate File. If the order of retrievals by the sequence of GET commands is intended to follow the ascending order of values for the key item, sorting the addresses in the Locate File by physical order is unnecessary.

A PLEX program can use the no-sort capability to avoid a physical-order sort for a where-clause consisting of a single key condition. The program requests this capability by assigning the value 1 to the variable S2KSRT, which is part of the S2KDUM control block. (At run-time, this variable is initialized to zero.) If S2KSRT equals 1 and a LOCATE command where-clause consists of a single key condition, the physical-order sort is not performed in the execution of that where-clause. Since no sort occurs, the addresses in the Locate File remain arranged according to the ascending order of values for the key item. Thus, the GET FIRST command referring to that Locate File retrieves the data record with the lowest value satisfying the key condition. Conversely, a GET LAST command retrieves the record with the highest value.

A GET1 command can also contain a where-clause consisting of a single key condition. The program can affect the processing of this where-clause by setting S2KSRT to 1. In this case, where-clause processing stops when the first data record meeting the key condition is selected; no physical-order sort is performed. If the GET1 command refers to the same subschema record it names in the single key condition, the selected data record contains the lowest value satisfying the condition.

You can request the no-sort capability for any single-condition where-clause, for example, when requesting LOCATE followed by ORDER BY. If the Locate File is unordered, data value retrievals for ORDER BY are in random order. This can cause a significant increase in run-time if there are several unordered references to the same database page. For a particular application and database, you can test the use of sorting versus not sorting and choose a procedure based on time studies.

## Example

Working with the EMPLOYEE database, a program might need to retrieve data records in the order of values for the key item EMPLOYEE NUMBER. If you do not use the no-sort capability, the program would include the following commands:

```
SUBSCHEMA RECORD ENTRY-REC OF EMPLOYEE.
01  ENTRY-REC.

    05  EMPLOYEE-NUMBER . . .
      :
      :
LOCATE ENTRY-REC WHERE
 EMPLOYEE-NUMBER EXISTS.          (SYSTEM 2000 software performs a
                                  physical-order sort of the addresses of
                                  located data records.)

   (Check return code.)
ORDER BY EMPLOYEE-NUMBER.         (specifies a sort of the addresses according
                                  to the ascending order
                                  of EMPLOYEE-NUMBER values)

   (Check return code.)
GET ENTRY-REC NEXT.               (retrieves the data record whose address
                                  comes next in the Locate File)

   (Check return code)
```

But if the program sets the variable S2KSRT to 1, as shown below, it prevents the physical-order sort, and the Locate File addresses remain arranged according to the order of values for the key item. Thus, the program can omit the ORDER BY command and the return code check for that command.

```
   MOVE 1 TO S2KSRT.

   LOCATE ENTRY-REC WHERE EMPLOYEE-NUMBER EXISTS.
   (Check return code)
         :
         :
    GET ENTRY NEXT.
   (Check return code)
         :
         :
```

Appendix B

# PL/I PLEX


*FORMAT OF PLEX COMMANDS B-3*
- *FORM and NOFORM B-3*
- *Continuation of a Command B-4*
- *Comments B-4*
- *One PLEX Command per Line B-5*
- *Reserved Words B-6*

*SYSTEM 2000 ITEM TYPES AND PL/I ATTRIBUTES B-7*
- *Item Type TEXT B-7*
- *Item Type CHARACTER B-8*
- *Item Type DATE B-8*
- *Item Type INTEGER B-9*
- *Item Types DECIMAL and MONEY B-11*
- *Item Types REAL and DOUBLE B-12*
- *UNDEFINED Item Type B-13*
- *Item Types and PL/I Attributes for Items in Where-Clauses B-13*
- *Considerations B-14*
- *Summary Chart B-15*

*COMMBLOCK DECLARATIONS B-19*
- *Format B-20*
- *COMMBLOCK Fields B-21*
- *Examples B-25*
- *Considerations B-30*

*SUBSCHEMA RECORD DECLARATIONS B-31*
- *Format B-32*
- *Examples B-34*
- *Considerations for Subschema Records B-40*
- *Considerations for Items in Subschema Records B-41*

*THE START S2K COMMAND B-43*

*INCLUSION LISTS FOR RETRIEVALS AND UPDATES B-45*
- *Format B-45*
- *Item List B-47*
- *Designation Bit String B-49*
- *Designation Vector B-52*

*INCLUSION LIST IN LINK COMMANDS B-55*
- *Format B-55*

*THE FOR COMMAND B-55*
- *Format B-55*
- *NAME or USE Option B-56*

*PL/I PLEX PROCESSOR DIRECTIVES B-57*
- *$BLOCKS Directive B-57*
- *$CICS ON Directive B-61*

*Command level B-61*
*$COPY Directive B-61*
*$NAME Directive B-62*
**PROCESS Statement B-62*
*$STORAGE Directive B-63*

*PL/I PLEX PROCESSOR EXECUTION OPTIONS B-71*
*CICS Execution Option B-71*
*COPY and NOCOPY Execution Options B-71*
*DECK and NODECK Execution Options B-72*
*ECHO and NOECHO Execution Options B-72*
*ERRORS and NOERRORS Execution Options B-73*
*ERRSYM Execution Option B-73*
*EXIT Execution Option B-74*
*Modifying, replacing, and adding records B-75*
*Diagnostic messages B-75*
*Example B-76*
*Notes B-76*
*FORM and NOFORM Execution Options B-78*
*GEN and NOGEN Execution Options B-78*
*ITEM Execution Option B-79*
*LINECNT Execution Option B-79*
*LOAD and NOLOAD Execution Options B-79*
*MEMORY Execution Option B-80*
*SORMGIN Execution Option B-80*
*SOURCE and NOSOURCE Execution Options B-80*
*SSR Execution Option B-81*
*SYSSEP Execution Option B-81*
*TERM and NOTERM Execution Options B-81*
*TRANS Execution Option B-82*

*SAMPLE PL/I PLEX PROGRAM B-83*
*FINDORG Program B-84*

*JOB CONTROL LANGUAGE (JCL) B-91*
*Single-User JCL for All Job Steps B-91*
*Multi-User JCL for All Job Steps B-91*
*JCL Variations B-94*
*Additional Locate Files (single-user) B-94*
*Additional database files B-95*
*$COPY Library B-95*

This appendix provides PL/I-specific information for the PLEX facility. It includes discussions of the general format of PLEX commands, PL/I attributes for SYSTEM 2000 item types, COMMBLOCK and SUBSCHEMA declarations, PL/I PLEX processor directives and execution options, along with a sample program and JCL. You should use the PLEX material in the main part of this manual as the basic documentation.

# FORMAT OF PLEX COMMANDS

The PLEX processor scans a program for PLEX statements and commands, turns some statements and commands into PL/I comments, and generates PL/I statements to carry out the requested processing. As the PLEX processor scans, it expects to find PLEX statements and commands in a prescribed format.

In general, each executable PLEX command consists of an initial keyword, a database identifier or subschema record name, and one or more other specifications or program variable names. Although a PLEX command can extend over more than one line (see **Continuation of a Command** on page B-4), the keyword, identifiers, and other names must not be divided across lines. If a command word is divided across lines, the PLEX processor considers it to be two words.

The placement of a PLEX command on a line depends on the specifications made in the SORMGIN execution option (see **SORMGIN Execution Option** on page B-80), if that option was used. If SORMGIN is not used, the default columns for placement are as follows:

- If the NOFORM execution option is used, a PLEX command can appear anywhere between columns 1 and 73.
- If the FORM execution option is used, a PLEX command follows the characters *PL in columns 2-4 and appears anywhere between columns 4 and 73.

A PLEX command terminates with a semicolon (;).

Columns 73-80 can contain program identification information. The PLEX processor does not read this portion. The PLEX processor counts the source lines and puts the sequence numbers in these columns on the expanded source listing.

## FORM and NOFORM

Since PLEX commands are intermixed with PL/I statements in a program, the PLEX processor must recognize these two kinds of input and process only the PLEX commands. The PLEX processor offers a choice of two processing methods.

With the NOFORM option, the PLEX processor examines the first word of each line. If it is not a PLEX keyword, the line is assumed to contain only PL/I statements. If the first word is a PLEX keyword, the PLEX processor looks for a PLEX command in that line. With the NOFORM option, a PLEX command can still begin with *PL in columns 2-4, but *PL is not required. The NOFORM format is the most commonly used format.

The FORM option is a little more restrictive. You code *PL in columns 2-4 of the first line of each PLEX command, and the command appears anywhere in columns 6-72. The PLEX processor looks at a line only if *PL appears in columns 2-4, then scans until it finds a semicolon. The processor assumes that the semicolon terminates the command. Continuation lines may or may not contain *PL. The PLEX processor looks for the next line with an *PL sign. Lines that do not begin with *PL in columns 2-4 are assumed to contain PL/I statements.

This method is supported only for compatibility with an old version of PLEX; it is not widely used because of the rigid coding format. To use this processing method, specify the FORM PLEX processor execution option in the program's JCL. See also **PL/I PLEX Processor Execution Options** on page B-71.

#### Examples

```
Column
2

*PL GET SSR1 NEXT;      (acceptable with FORM and NOFORM)

*PL REMOVE SSR1;        (acceptable with FORM and NOFORM)

GET SSR1 NEXT;          (acceptable with NOFORM only)
```

## Continuation of a Command

A PLEX command can be continued on a second line when the command is too long to fit on one line. Most of the PLEX commands refer to a database either by a database or COMMBLOCK name (as in the OPEN command) or by a subschema record name (as in the retrieval and update commands). Either the subschema record name or the database or COMMBLOCK name must appear on the first line of such a command.

## Example

```
GET SSR1
FIRST;                  (acceptable)

GET
SSR1 FIRST;             (not acceptable)
```

When the FORM option is used, the *PL may be either omitted or coded in the continuation line.

```
*PL GET SSR1
   FIRST;               (acceptable)

*PL GET SSR1
*PL FIRST;              (acceptable)
```

## Comments

The PLEX processor transforms PLEX commands into PL/I comments and generates equivalent PL/I statements for them. Therefore, if a comment is present in the same line as a PLEX command, it results in nested comments, thereby creating an error during PL/I compilation.

**Example**

Suppose the input to the PLEX processor is

```
GET SSR1 FIRST;          /* THIS IS A COMMENT */
```

The output from the PLEX processor would be

```
/* GET SSR1 FIRST;        /* THIS IS A COMMENT */  */
```

The PL/I compiler would not accept the */ at the end.

Therefore, comments should be coded on separate lines, either before or after a PLEX command. There are a few declarative statements in PLEX that are not turned into comments. For example, item declarations in a subschema record are passed to the PL/I compiler directly. See **SUBSCHEMA RECORD Declarations** on page B-31 for examples of subschema record declarations and the resulting output from the PLEX processor. A complete comment can be placed on a line that contains a subschema item declaration, but not on a line that contains EQ *item name* or FORM *date format* because these specifications are changed into comments by the PLEX processor.

## One PLEX Command per Line

In subschema record (or COMMBLOCK) declarations, each subschema item declaration (or COMMBLOCK field) must appear on a separate line. In addition, the first two statements, namely, SUBSCHEMA RECORD . . . OF (or COMMBLOCK . . . OF) and DCL 1 *subschema-record-name* (or DCL 1 *COMMBLOCK name*), cannot be mixed with other declarations on one line. Also, EQ *item-name* and FORM *date-format* statements of a subschema record declaration must appear on separate lines.

**Example**

```
SUBSCHEMA RECORD A OF B; DCL 1 A,          (not acceptable)

      ------------------------------------------------------------
SUBSCHEMA RECORD A OF B;
DCL 1 A,          3 A1 CHAR(4),            (not acceptable)

      ------------------------------------------------------------
3 DATE CHAR(6), FORM YYMMDD,               (not acceptable)

      ------------------------------------------------------------
3 EN CHAR(4),
EQ EMPLOYEE NUMBER, 3 SEX CHAR(11),        (not acceptable)
```

A line cannot contain more than one PLEX command. Also, PLEX commands and PL/I statements cannot be mixed on the same line.

## Reserved Words

The following names must not be used as procedure names in a PLEX program:

```
S2KPL
PROC or PROCEDURE
SCHEMAS
S2KPLC (CICS only)
```

S2KPL is the name of the routine that is called to process PLEX commands. If PROC, PROCEDURE, or SCHEMAS is used as a procedure name, the PL/I statement END *procedure name* would be interpreted by the PLEX processor as the PLEX command END PROCEDURE or END SCHEMAS (old syntax for $BLOCKS). This can produce undesirable results.

See **Reserved Words** on page 3-3 for information about reserved words and keyword restrictions for PLEX statements.

# SYSTEM 2000 ITEM TYPES AND PL/I ATTRIBUTES

An item in a SYSTEM 2000 database can be one of the following item types: CHARACTER, TEXT, UNDEFINED, DATE, INTEGER, DECIMAL, MONEY, REAL, or DOUBLE. Since all data transfer between a PLEX program and a database takes place through a subschema record, the subschema items must have attributes compatible with the item types of the schema items they represent. SYSTEM 2000 software offers its users data independence by converting the data stored in an internal form to a program-understandable form for retrievals, updates, and comparisons in a where-clause. This chapter gives a brief description of SYSTEM 2000 item types and their correspondences with PL/I attributes.

When a SYSTEM 2000 database is defined, the picture and the size of a schema item are specified with its type. Only DATE items have a standard picture and size designation. The example below lists some component numbers and names along with their item types and pictures as they appear in a database definition.

**Example**

```
  2*    LAST NAME (CHAR X(10))
  5*    BIRTHDAY (DATE)
  6*    SECURITY CLEARANCE (INTEGER NUMBER 999)
 11*    ACCRUED VACATION (DECIMAL NUMBER 999.99)
111*    PAY RATE (MONEY $9999.99)
132*    COMMENT TEXT (TEXT X(7))
```

## Item Type TEXT

A value of item type TEXT is a string of characters including alphabetic, numeric, special characters, and blanks. In the example, COMMENT TEXT is of item type TEXT and seven bytes are reserved for each value of this item in the Data Table. When a value is shorter than its picture size, the data are stored left-justified and filled with trailing binary zeros. When a value is longer than its picture size, the number of characters stored in the Data Table is four less than the picture size; the last four bytes in the field contain the pointer to the Extended Field Table, where the rest of the characters are stored. Therefore, the picture size must be at least four in order for a value to use the Extended Field Table.

The corresponding PL/I attributes for a TEXT item are PICTURE and CHAR. When SYSTEM 2000 software puts a TEXT value into a subschema record (from the database), the value is placed left-justified and blank-filled or truncated on the right as needed. If the item is null, the field is filled with binary zeros.

When SYSTEM 2000 software takes a TEXT value from a subschema item field, it scans from left to right for the value. The scan is terminated either by the end-of-field as given in the subschema record declaration or by a byte of binary zeros. Therefore, if the first byte contains binary zeros, the item is considered null.

## Item Type CHARACTER

Item type CHARACTER is like type TEXT, except that a value's extraneous blanks (leading, trailing, or multiple embedded) are removed. Thus, JOHN SMITH is stored as JOHN SMITH. Short and long CHARACTER values are stored like those of type TEXT. The corresponding PL/I attributes for item type CHARACTER are CHAR and PICTURE.

During the transfer of a value from a database to a subschema record, SYSTEM 2000 software left-justifies the value with blank-fill if the field in the subschema record is shorter, or truncates the value if the field is longer.

When SYSTEM 2000 software receives a value of item type CHARACTER from a subschema item field, it scans from left to right, discarding leading, trailing, and extraneous (more than one) embedded blanks. The scan is terminated by the end of the field or a byte of binary zeros, whichever comes first. Therefore, a field full of blanks or a field whose first byte contains binary zeros is interpreted as null.

## Item Type DATE

DATE values are special numeric data; only dates written in conventional forms are qualified to be of this type. SYSTEM 2000 software allows arithmetic calculations on dates as on other numbers. A subschema item corresponding to a DATE item should be declared with the PL/I CHAR or PICTURE attribute and can be either six or eight bytes long. The DATE value can be placed in this field in six possible formats:

```
MMDDYYYY
MMDDYY
DDMMYYYY
DDMMYY
YYYYMMDD
YYMMDD
```

The format to be used for an item is specified in the declaration for the corresponding variable in the subschema record or is selected by default. MMDDYYYY is the default for a subschema record item that is eight bytes long. The default is MMDDYY for a six-byte field. If the field length is other than six or eight bytes, the PLEX processor issues an error message.

The value representing a date must be placed in its field in the specified format. SYSTEM 2000 software issues return code 10 if the value for an update or a where-clause does not conform to the date format. In the case of retrievals, SYSTEM 2000 software places the DATE value in the specified format in the corresponding field in the subschema record. Binary zeros represent null DATE items.

## Item Type INTEGER

In SYSTEM 2000 software, signed integer numbers are of item type INTEGER. They are stored in packed decimal representation in the database. The corresponding field in the program can be declared with any of the following attributes: FIXED DECIMAL, FIXED BINARY, FLOAT BINARY, FLOAT DECIMAL, or PICTURE. During a retrieval, SYSTEM 2000 software converts the data item from the internal packed decimal representation to the representation for the declared attribute and moves the converted value to the appropriate field. For an update or a where-clause comparison, it converts the value in the subschema record to packed decimal representation.

**FIXED DECIMAL attribute** When an INTEGER item type is declared as FIXED DECIMAL in a subschema record, no conversion is necessary since the representation of the number is packed decimal in both cases. Therefore, data transfer in this case would be slightly faster than with the other available attributes.

The size of the FIXED DECIMAL field (total number of bytes) must be the same as the size of the corresponding INTEGER field in the database. There should not be any digits to the right of the decimal point in the FIXED DECIMAL field. Thus, a valid declaration for an item defined as SECURITY CLEARANCE (INTEGER NUMBER 999) would be FIXED DECIMAL (3,0).

When a subschema record is referenced for the first time, all the items in the subschema record are validated. This is called "the opening of a subschema record." If the attributes of a subschema record item do not agree with the item type of the corresponding item in the database, return code 17 is placed in the second field of the COMMBLOCK for the database, and the ordinal position of that item within the subschema record is placed in the eighth field of the COMMBLOCK. For example, if the declaration for the subschema record item SECURITY_CLEARANCE were FIXED DECIMAL (5,0) or FIXED DECIMAL (3,2), return code 17 would result when the subschema record was referenced.

With packed decimal representation, the number of bytes required to store an even number of digits is the same as that required to store the next higher odd number of digits. Therefore, storage for the field with attributes FIXED DECIMAL (3,0) is also the same as that for INTEGER 99. In this case, although a return code of zero is issued during validation, return code 10 results if an attempt is made to update a two-digit value in the database with a three-digit FIXED DECIMAL field. Therefore, it is good practice to declare the subschema item with the same number of digits as the corresponding schema item declaration. In retrievals as well as in updates, a short value is stored in the receiving field (the subschema record in retrievals, the database in updates) right-justified and filled with packed decimal zeros.

Binary zeros are placed in a subschema record as a result of the retrieval of a null. SYSTEM 2000 software interprets a field of binary zeros as a null for an update or a comparison in a where-clause.

**FIXED BINARY attribute** When an INTEGER item is retrieved into a FIXED BINARY field, the value is converted into its binary representation and is moved into the field right-justified. If the value to be retrieved is too large (or negative and too small) to be represented in a fullword, an abnormal termination of the job occurs with a completion code of OC9 and a SYSTEM 2000 Error Code. Truncation occurs when the value can be placed in a fullword, but the receiving field is shorter than the length of the value.

During updates, if the converted value is shorter than its corresponding size in the database definition, the value is stored right-justified and filled with packed decimal zeros. If the converted value is longer than the corresponding field in the database, return code 10 is issued.

Null items are represented by the characters NULL if the field is a fullword and by the characters NL if the field is a halfword.

**FLOAT DECIMAL and FLOAT BINARY attributes** An INTEGER item can be declared as FLOAT DECIMAL or FLOAT BINARY in a subschema record. During the retrieval of such an item, the integer value is converted to floating point representation and is moved into the subschema record field. The least significant digits of a number of very large magnitude may be rounded or truncated during this process.

With updates, if the value is not an integer, it is rounded during conversion. If the converted value is shorter than the receiving field in the database, it is moved right-justified and zero-filled (packed decimal zeros). If the converted value is longer than the receiving field, return code 10 is issued.

Null items are represented as NULL and NL in fullword and halfword fields, respectively.

**PICTURE attribute** The picture must be numeric and without edit characters in order to be accepted by the PLEX processor. For retrievals, the value is converted to character representation and is moved into the field right-justified on the left side of the decimal point and truncated or zero-filled (with character 0) as necessary. The right side of the decimal point is zero-filled. For updates, the value is converted into packed decimal form and is moved right-justified and zero-filled (with packed decimal zeros) or truncated as necessary. Because of the truncation on the left, the most significant digits may be lost during retrievals and updates. Binary zeros in the subschema record represent null items in retrievals as well as in updates and where-clauses.

## Item Types DECIMAL and MONEY

In SYSTEM 2000 software, decimal numbers can be item type DECIMAL or MONEY. Both DECIMAL and MONEY type items are stored in packed decimal representation in a SYSTEM 2000 database. MONEY items are similar to DECIMAL items except that they can be displayed with insertion characters, for example, $ and CR, using the Self-Contained Facility of SYSTEM 2000 software.

The corresponding field in the program can be declared with any of the following attributes: FIXED DEC, FIXED BIN, FLOAT DEC, FLOAT BIN, and PICTURE. For example, for an item defined as ACCRUED VACATION (DECIMAL NUMBER 999.99), the following attributes in a subschema item declaration are valid:

```
FIXED DEC (5,2)
FIXED BIN (15)
FLOAT BIN (21)
FLOAT DEC (6)
PICTURE '999V99'
```

Rules governing the data transfer between a subschema record and a database are similar to those for INTEGER type items. The representation of nulls is also the same.

**FIXED DECIMAL attribute** When a DECIMAL or MONEY type item is declared as FIXED DECIMAL in a subschema record, no conversion is necessary. If the subschema record field does not have the same field length in bytes and the same number of digits to the right of the decimal point as compared to the corresponding schema picture, return code 17 is issued during the opening of the subschema record. During a data transfer (in retrievals as well as in updates), the decimal points are aligned and the data are moved right-justified on the left side of the decimal point and left-justified on the right side of the decimal point. If the value is shorter than the receiving field, the receiving field is filled with packed decimal zeros. If the value in the sending field is longer on the left side of the decimal point, so that the data transfer would cause the truncation of the most significant digit if the move were allowed, return code 10 is issued. This happens only when the database picture is different than the size allowed by the subschema record. Nulls are represented as binary zeros.

**FIXED BINARY attribute** A DECIMAL or MONEY type item with the decimal point at the right end of its picture can be declared FIXED BINARY in a subschema record. For retrievals, the value is converted to its fixed binary representation and is moved into the field right-justified; the higher order bits are truncated if necessary. If the binary representation requires more than 32 bits, the job is terminated abnormally with a completion code of OC9 and a SYSTEM 2000 Error Code. Truncation occurs when the value can be represented in 32 bits, but is longer than the receiving field.

For updates, the value is converted to packed decimal and is stored in the corresponding field in the database right-justified and zero-filled, if necessary, on the left side of the decimal point. The right side of the decimal point is filled with zeros. When the converted value is longer than the number of digits on the left of the decimal point in the receiving field, return code 10 is issued to avoid the truncation of the most significant digits.

Null items are represented by NULL in a fullword and by NL in a halfword.

**FLOAT DECIMAL and FLOAT BINARY attributes** A DECIMAL or MONEY item in the database declared as FLOAT DECIMAL or FLOAT BINARY in a subschema record is converted from packed decimal to its floating point representation when retrieved into a subschema record. The least significant digits may be rounded or truncated in the process.

For an update, the floating point value is converted to packed decimal and is moved into the appropriate field in the database. The decimal points are aligned. The digits to the right of the decimal are left-justified and zero-filled or rounded. The digits to the left of the decimal point are right-justified and zero-filled if necessary. No truncation of the most significant digits occurs; return code 10 is issued if the converted decimal value has more digits to the left of the decimal point than the receiving field allows.

Null items are represented by NULL or NL as in the case of INTEGER items.

**PICTURE attribute** The rules for DECIMAL and MONEY items with PICTURE attributes in a subschema record are similar to those for INTEGER items declared with PICTURE attributes in a subschema record. For retrievals, the value is converted to character representation; for updates, the value is converted to packed decimal; and for both updates and retrievals, the value after conversion is moved into the receiving field with the decimal points aligned. Truncation or zero-filling may occur on either side of the decimal point. Loss of the most significant bits may occur when a long value is retrieved into a short field. Binary zeros represent nulls.

## Item Types REAL and DOUBLE

REAL and DOUBLE items are acceptable in all PLEX syntax, including the DYNAMIC where-clause.

For REAL and DOUBLE items, internal normalized floating point numbers must be passed by the PLEX program to SYSTEM 2000 software. SYSTEM 2000 software performs no conversions. REAL and DOUBLE values returned to the PLEX program are also internal normalized floating point numbers. External floating point numbers are not allowed in a subschema. NULL or NL in the first word signifies a null value.

You can use FLOAT BIN(21) or FLOAT BIN(53) for REAL and DOUBLE items. The values must be properly aligned in a subschema record definition. SYSTEM 2000 software does not require word alignment when moving data to or from a PLEX subschema record, but subsequent program statements that reference REAL and DOUBLE subschema items do require proper alignment. Therefore, the PLEX processor issues message 361 if you have not aligned the subschema items correctly.

Two SYSTEM 2000 execution parameters, OPT039 and OPT040, allow you to specify several alternatives that check for padding and truncation of floating point numbers. When you transfer single and double precision numbers to and from the database, you can request

complete data integrity or you can check return codes 98 and 99 to see whether padding or truncation occurred. For details about OPT039 and OPT040, see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*.

## UNDEFINED Item Type

All 256 EBCDIC characters are allowed for an UNDEFINED item. The format for UNDEFINED input and output is character format, which means each byte represents one byte of data. Data conversions from hexadecimal to display are not performed. Also, UNDEFINED values cannot overflow; if a value exceeds its picture, return code 10 is issued.

You can use PICTURE or CHAR for UNDEFINED type subschema items. Nulls are represented by all binary zeros.

## Item Types and PL/I Attributes for Items in Where-Clauses

Among other things, the where-clause provides two kinds of comparison of data values: comparison of two data values in the database and comparison of a value in the database with a value in a program variable (which may belong to a subschema record).

In the first case, both data values in the database should be from the same group of compatible item types. There are two such groups: TEXT, CHARACTER, and UNDEFINED items belong to one group, and INTEGER, DECIMAL, MONEY, REAL, DOUBLE, and DATE items belong to the other group. Before the comparison is made, the value of the right operand is reformatted according to the item type and picture of the left operand. Thus, if the left operand is an INTEGER value and the right operand is a DECIMAL item, the DECIMAL or MONEY value is truncated before the comparison. Therefore, place the INTEGER value as the right operand to avoid the truncation of the DECIMAL value. When a DATE value is the right operand and the left operand is an INTEGER value, the DATE value is converted to an INTEGER value, which is the difference in number of days between the date and October 15, 1582. If the left operand is a DATE value and the right operand is an INTEGER value *n*, the latter is treated as the date which comes *n* days after October 15, 1582. CHARACTER, TEXT, and UNDEFINED items are compared as they appear, that is, extraneous blanks are not suppressed from TEXT and UNDEFINED values for comparison with CHARACTER values.

In the second kind of comparison, the left operand is always a subschema item. The right operand of a binary operator and the two right operands of a ternary operator are program variables which may be subschema items. For the left operand, the value is taken from the database; for the other operands, the value to be compared is taken from the program storage area for the variables. SYSTEM 2000 software uses the attributes of the left operand to interpret the right operand.

That is, the type and length of the left operand, as declared in a subschema record, are understood by SYSTEM 2000 software as the type and length of the program variable(s) in the right operand. The left and right operands should be of equal length. If, however, the declared length of a program variable in the right operand is shorter than the length of the item in the left operand, the right operand value must be followed by a byte of zero bits; otherwise, results of the comparison of operands are unpredictable.

If the left operand has an invalid type, return code 17 is issued even when the type of the right operand is valid. If the left operand has a valid type and the right operand does not, there is no error during validation, but the error is detected during comparison, and return code 17 is issued.

SYSTEM 2000 software handles the values for a where-clause comparison in a manner similar to the way it handles values for updates. The difference is that in a where-clause comparison the attributes of the left operand are used, and the data value to be compared is placed in a work area and not in the database. The validation, conversion, and placement of the value in a field are identical in both cases.

## Considerations

- TEXT values require special care. SYSTEM 2000 software fills TEXT fields with binary zeros on the right since blanks are significant characters in TEXT items. Therefore, when a TEXT value in a subschema record variable is shorter than its field length and is used in a where-clause or for an update, the PLEX program must fill the field with binary zeros on the right. Otherwise, the where-clause comparison would fail or the updated value would have extra trailing blanks in the database.

- Data items defined as INTEGER, DECIMAL, MONEY, and DATE are stored in the database as packed decimal. INTEGER, DECIMAL, and MONEY items can be declared in PL/I PLEX with five different attributes, as mentioned earlier. It is significantly more efficient, where possible, to let the PL/I compiler generate the necessary code for conversion through an assignment statement than to leave the conversion up to SYSTEM 2000 software. The compiler can examine the pictures of the sending and receiving fields at compile time and generate appropriate machine instructions, whereas SYSTEM 2000 software must interpret both fields at execution time (whenever the component is referenced) and do some relatively complex work to perform the conversion. A conversion by SYSTEM 2000 software is not necessary if subschema items for INTEGER, MONEY, and DECIMAL schema items are declared as FIXED DECIMAL.

- When an INTEGER item is declared as FIXED BINARY, FLOAT DECIMAL, or FLOAT BINARY in a subschema record, the characters NULL (or NL) are placed in the field when a null item is retrieved. Similarly for updates and where-clauses, if the characters NULL (or NL) appear in a field, SYSTEM 2000 software interprets that as a null item.

  When the field is declared as BINARY FIXED, the binary representation of NULL and NL are the same as that of the numbers -706,423,853 and -10,797, respectively. Therefore, if those numbers are stored in the database, it is better to declare them as FIXED DECIMAL or PICTURE, so that there is no misinterpretation on the part of the user or SYSTEM 2000 software.

- Two return codes are associated with item types. When a subschema record or any of its items is referenced for the first time, all the items in that subschema record are validated for acceptable attributes. A return code of 17 is issued if the attributes do not conform to the corresponding item type and picture of the schema item. If all of its items have valid attributes, the subschema record is opened. However, if an unacceptable data value is placed in a correctly declared field and the data value cannot be moved into the requested field during an operation, SYSTEM 2000 software issues a return code of 10.

  When return codes 10 and 17 are issued, the ordinal position of the item in error within the subschema record is placed in the eighth field of the appropriate COMMBLOCK. (See also the discussion of using the ordinal position for rejected conditions in a where-clause, under **COMMBLOCK Fields** on page 5-3.)

## Summary Chart

The correspondence between SYSTEM 2000 item types and PL/I attributes is summarized in Illus. B.1 on page B-16 . This table shows how a field corresponding to a schema item must be declared in a program and how a value is to be placed in the field so it can be interpreted correctly by SYSTEM 2000 software. It also shows how a value retrieved from a SYSTEM 2000 database is placed in the corresponding field in the program. Representations of null items of all types are given.

**Illus. B.1** SYSTEM 2000 Item Types and PL/I Attributes

| Item Type | PL/I Attributes | Requirements/ Restrictions | From Database | To Database |
|---|---|---|---|---|
| TEXT | PICTURE or CHAR | You must fill with binary zeros to avoid storage of trailing blanks for updates and to ensure successful comparisons in where-clause.<br><br>There is no way of knowing the exact number of trailing blanks with a retrieval. | Stored left-justified, blank-filled or truncated on right as necessary.<br><br>Null = binary zeros. | Placed in the database left-justified, binary zero-filled or carried over to the Extended Field Table as necessary.<br><br>You should place left-justified and binary zero-filled.<br><br>Null = binary zeros in the first byte. |
| CHARACTER | PICTURE or CHAR | | Same as in TEXT. | Placed same as for TEXT.<br><br>Null = binary zeros in the first byte or all blanks. |
| DATE | PICTURE or CHAR | Must be 8 or 6 bytes long. | Placed according to format chosen.<br><br>Null = binary zeros. | You should place according to date format chosen.<br><br>Null = binary zeros. |
| INTEGER | FIXED DECIMAL or CHAR | Total length in bytes must be equal to that of schema item. Number of digits to the right of the decimal point must be zero. | Stored right-justified and zero-filled. Truncation is not permitted.<br><br>Null = NULL or NL. | Placed in the database right-justified and zero-filled. Truncation of the most significant digits is not permitted.<br>Null = binary zeros. |
| INTEGER | FIXED BINARY | Abend if the value cannot be represented in 32 bits. | Binary representation is placed. Truncation may occur on the left.<br><br>Null = NULL or NL. | Same as above.<br><br>Null = NULL or NL. |
| INTEGER | FLOAT BINARY or FLOAT DECIMAL | | Floating point representation is placed. The least significant digits may be truncated or rounded.<br><br>Null = NULL or NL. | Same as above.<br><br>The least significant digits may be rounded or truncated.<br><br>Null = NULL or NL. |
| INTEGER | PICTURE | Only numeric picture, no edit characters. | Character representation is placed with aligned decimal point and zero-filled. Either side may be truncated if necessary.<br><br>Null = binary zeros. | Placed in the database right-justified and zero-filled. The least significant digits may be rounded or truncated; the most significant digits may be truncated if necessary.<br>Null = binary zeros. |

*continued on next page*

**Illus. B.1** *continued*

| Item Type | PL/I Attributes | Requirements/ Restrictions | From Database | To Database |
|---|---|---|---|---|
| DECIMAL or MONEY | FIXED DECIMAL | Must match schema item picture size and number of digits to the right of the decimal point. | Stored decimal point aligned. Truncation is not permitted.<br><br>Null = binary zeros. | Placed in the database decimal-point-aligned and zero-filled. Truncation of the most significant digits is not permitted.<br><br>Null = binary zeros. |
| DECIMAL or MONEY | FIXED BINARY | Decimal point should be at the right end in the schema item picture.<br><br>Abend if binary representation exceeds 32 bits. | Binary representation is placed right-justified; truncation of the most significant bits may occur.<br><br>Null = NULL or NL. | Same as above. |
| DECIMAL or MONEY | FLOAT BINARY or FLOAT DECIMAL | | Floating point representation is placed; the least significant bits may be rounded or truncated.<br><br>Null = NULL or NL. | Decimal point is aligned and zero-filling occurs when placed in the data base. Truncation of the most significant digits is not permitted. Truncation or rounding of the least significant digits is allowed.<br><br>Null = NULL or NL. |
| DECIMAL or MONEY | PICTURE | Only numeric pictures without edit characters are permitted. | Character representation is aligned by decimal points and zero-filled. Truncation may occur on either side as necessary.<br><br>Null = binary zeros. | Placed in the database decimal-point-aligned, zero-filled if necessary. Truncation may occur on either side.<br><br>Null = binary zeros. |
| REAL | FLOAT BIN(21) or FLOAT BIN(53) | Requires fullword alignment or doubleword alignment. The OPT039 and OPT040 execution parameters determine how the system handles padding and truncation of floating point numbers (see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*).<br><br>Zero means a positive zero. Negative zero is not allowed; causes return code 10. | Internal normalized floating point number is placed in the subschema. External floating point numbers are not allowed in the subschema.<br><br>If you use FLOAT BIN(53), see the OPT039 execution parameter for details about padding in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*<br><br>Null = NULL. | Must be internal normalized floating point number. SYSTEM 2000 software performs no conversions. External floating point numbers are not allowed.<br><br>If you use FLOAT BIN(53), see the OPT040 execution parameter for details about truncation in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*<br><br>Null = NULL. |

*continued on next page*

**Illus. B.1** *continued*

| Item Type | PL/I Attributes | Requirements/ Restrictions | From Database | To Database |
|---|---|---|---|---|
| DOUBLE | FLOAT BIN(53) or FLOAT BIN(21) | Requires doubleword alignment or fullword alignment. The OPT039 and OPT040 execution parameters determine how the system handles padding and truncation of floating point numbers (see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*).<br><br>Other rules same as for REAL. | Internal normalized floating point number is placed in the subschema. External floating point numbers are not allowed in the subschema.<br><br>If you use FLOAT BIN(21), see the OPT040 execution parameter for details about truncation in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*<br><br>Null = NULL in the first word. | Must be internal normalized floating point number. SYSTEM 2000 software performs no conversions. External floating point numbers are not allowed.<br><br>If you use FLOAT BIN(21), see the OPT039 execution parameter for details about padding in the *SYSTEM 2000 Product Support Manual, Version 12, First Edition.*<br><br>Null = NULL or NL in the first word. |
| UNDEFINED | PICTURE or CHAR | Each byte represents one byte of data. Data conversions from hexadecimal to display are not performed. If an UNDEFINED value overflows its picture, return code 10 is issued. | Value placed left-justified in subschema with trailing zero-fill.<br><br>Null = all binary zeros. | All 256 EBCDIC characters are allowed. Placed in database left-justified with trailing zero-fill.<br><br>Null = all binary zeros. |

# COMMBLOCK DECLARATIONS

The COMMBLOCK declaration sets up a communication area for a database to keep the status flag, return code, and other identification information.

- A COMMBLOCK can be passed as a parameter.
- Any acceptable PL/I storage class attribute can be given to a COMMBLOCK.
- A COMMBLOCK can be given a name that is different from the database name.

A separate COMMBLOCK declaration is required for each database to be accessed by a program. COMMBLOCK declarations can appear anywhere in a program, although it is a good practice to have them at the beginning where they can be found easily.

A COMMBLOCK declaration has two parts. The first part gives the name of the database for which the communication area is to be set up, and it specifies whether the COMMBLOCK is to be received as a parameter. In the second part, which is like a PL/I structure declaration, the name of the COMMBLOCK and the names of its fields are declared.

The PLEX processor converts the COMMBLOCK declaration to a PL/I DECLARE statement. The structure name is generated by adding the suffix '_0' to the name of the COMMBLOCK. The PLEX processor creates a 28-byte control block and puts it before your COMMBLOCK fields.

## Format

```
COMMBLOCK OF database-name [PARAMETER]     ;
DECLARE      |1  name [storage-attribute],
             |01
n  var-1      ,
n  var-2      ,
n  var-3    [INIT (initial-value)],
n  var-4      ,
n  var-5    [INIT (initial-value)],
n  var-6      ,
n  var-7      ,
          :
          :
n  var-14   [,n  var-15]   ;
```

*database-name* is the name of a SYSTEM 2000 database.

*name* is any valid PL/I name of 29 characters or less.

*storage-attribute* is any valid PL/I storage class attribute. The default is the PL/I default storage parameter.

*n* is any valid PL/I level number greater than 1.

All fields in a COMMBLOCK must have the same level number. If this level number is less than 3, the PLEX processor substitutes a level number that is one greater than the given level number. If the level numbers for all fields are not identical, the PLEX processor issues an error message.

*var-n* is any valid PL/I name of 30 characters or less, naming a field in the COMMBLOCK.

*initial-value* is a four-character initial value assigned to the field.

**[PARAMETER]** If omitted, the COMMBLOCK cannot be received as a parameter. If the COMMBLOCK is to be received as a parameter, the called procedure must use the PARAMETER option in the first statement of the COMMBLOCK declaration. No storage attribute should be assigned to such a COMMBLOCK. When a COMMBLOCK is passed as a parameter, the invoking routine must use the suffixed name of the COMMBLOCK generated by the PLEX processor, for example, LIBRARY_0. The COMMBLOCK can be declared with a storage attribute in the invoking routine.

Abbreviations:
PARAMETER = PARM, PARA
DECLARE = DCL

## COMMBLOCK Fields

*var-1* (subschema record name)

The PLEX processor declares this field as CHAR(32) and initializes it to blanks. If a PLEX command refers to a subschema record, SYSTEM 2000 software places the name of the record in this field. If the command does not reference a subschema record, SYSTEM 2000 software puts blanks in this field. When a command is rejected because of an error in a subschema item, SYSTEM 2000 software stores the name or alias of that subschema item in this field. Thus, you can use this field as a debugging aid.

*var-2* (return code)

The PLEX processor declares this field FIXED BIN(31) and initializes it to zero. The values returned to this field are very important. This field contains the return code from SYSTEM 2000 software after the execution of each command referring to that COMMBLOCK. Every PLEX program should check the value in this field after each PLEX command and take any action necessary. Failure to check the return code immediately can multiply errors and cause problems which would be difficult to solve. See *SYSTEM 2000 Messages and Codes, Version 12, First Edition* for the meaning of each return code.

**Note**: The variable S2KRTC in the S2KDUM control block contains the return code after every command, regardless of whether the command refers to a database. Since some commands, such as START S2K, do not refer to a database, their return codes are placed only in S2KRTC, not in any COMMBLOCK.

*var-3* (for the user)

The PLEX processor declares this field as CHAR(4). This field is not used by SYSTEM 2000 software. You can store information about the database in this field and can initialize it with the INITIAL attribute.

*var-4* (last data record indicator)

The PLEX processor declares this field as FIXED BIN(31) and initializes it to zero. SYSTEM 2000 software sets this field to one when the data record for the last address in a Locate File is referred to by a GET command, regardless of whether the command is successful. Otherwise, it is set to zero. You can use this field after a LOCATE command in order to control the retrieval of data records from a located subset as follows.

Suppose LAST is the name of the fourth COMMBLOCK field.

```
        .
        .
        .
   LOCATE SSR WHERE . . . ;
   (Check return code)
        .
        .
        .
DO WHILE (LAST = 0);
        .
        .
        .
   GET  SSR . . . ;
   (Check return code)
        .
        .
        .
END;
        .
        .
        .
```

For the GETP command only, this field has a different function. SYSTEM 2000 software sets this field to one if the database contains a specified subschema item value that equals the value in the where-clause condition. Otherwise, for GETP, this field is set to zero.

*var-5* (password)

The PLEX processor declares this field as CHAR(4). This field must contain a valid password before the execution of any PLEX command. You can initialize the password with the INITIAL attribute, or you can move a password into this field with an assignment or input statement before opening the database.

*var-6* (number of records selected by a LOCATE command)

This field is declared as FIXED BIN(31) and is initialized to zero by the PLEX processor. SYSTEM 2000 software sets this field to a binary number after each successful LOCATE command or GET1 command with a where-clause for the database. The value of this field indicates the number of data records selected as a result of a LOCATE command or the number of qualified records resulting from a GET1 command with a where-clause. You can use this field after a LOCATE command to control the retrieval of data records from a selected subset of records.

Suppose the name of the sixth COMMBLOCK field is NUMREC.

```
          .
          .
          .
   LOCATE SSR WHERE . . . ;
   (Check return code)
          .
          .
          .
DO I = 1 TO NUMREC;
          .
          .
          .
   GET SSR . . . ;
   (Check return code)
          .
          .
          .
END;
          .
          .
          .
```

*var-7* (Locate File pointer: position of retrieved data record in Locate File)

The PLEX processor declares this field as FIXED BIN(31) and initializes it to zero. In this field SYSTEM 2000 software places the ordinal of the data record address in the Locate File when a data record is retrieved. You can use this field for debugging when an error return code occurs.

*var-8* (level)

The PLEX processor declares this field as FIXED BIN(31) and initializes it to zero. During the normal execution of a command, SYSTEM 2000 software uses this field to store the hierarchical level of the data record accessed. However, under error conditions this field contains information helpful for debugging, such as the ordinal of a rejected item in its subschema record, the ordinal of a component in a where-clause responsible for the rejection of the where-clause, or the level of a data record removed from the database.

*var-9* (time)

The PLEX processor declares this field as CHAR(8) and initializes it to blanks. This field contains the time of the latest update cycle in the format HH:MM:SS, that is, hours, minutes, seconds. SYSTEM 2000 software enters the time in this field whenever the database is opened or updated.

*var-10* (date)

The PLEX processor declares this field as CHAR(8) and initializes it to blanks. In this field, SYSTEM 2000 software places the date of the latest update cycle whenever the database is opened or updated. The default format is MM/DD/YYYY (month/day/year).

*var-11* (cycle number)

This field is declared as FIXED BIN(31) and is initialized to zero by the PLEX processor. SYSTEM 2000 software places the cycle number of the database in this field whenever the database is opened or updated. SYSTEM 2000 software maintains a cycle number for each database internally in the database files. The cycle number indicates how many times the data in the database were changed. The internal cycle number is moved into this field of the COMMBLOCK so that you can associate the cycle number with the updates to the database.

*var-12* (system separator)

The PLEX processor declares this field as CHAR(4) and initializes it to blanks. When the database is opened, SYSTEM 2000 software places the system separator (right-justified and binary zero-filled) in this field. The default system separator is an asterisk.

**Note**: The PLEX program must use an asterisk as the system separator with ternary operators in a where-clause unless you specify a different character with the SYSSEP execution option (see **SYSSEP Execution Option** on page B-81).

*var-13* (entry terminator)

The PLEX processor declares this field as CHAR(4) and initializes it to blanks. When the database is opened, SYSTEM 2000 software places the entry terminator word in this field. The default entry terminator word is END, left-justified and blank-filled. The entry terminator word is not used in PLEX.

*var-14* (database damage flag)

The PLEX processor declares this field as FIXED BIN(31) and initializes it to zero. When the database is opened, SYSTEM 2000 software moves the internal damage flag for the database into this field. A zero value indicates an undamaged database, and a value of one indicates that the database is damaged. The program should check this field after the OPEN command is given. Updates are not permitted for a damaged database; however, data retrievals are allowed.

*var-15* (parent - optional field)

The PLEX processor declares this optional field as FIXED BIN(31) and initializes it to zero. When any retrieval command is executed, SYSTEM 2000 software uses this field to store the value of the Hierarchical Table pointer for the parent of the data record retrieved.

You can use this field to distinguish between similar data records under different parents. The value of this field, being an internal address, is not meaningful to you. However, you can save it after a retrieval and then compare it with the value of the parent in the next retrieval.

## Examples

These examples show how the program actually declares the COMMBLOCK, followed by the expanded source code produced by the PLEX processor as input to the PL/I compiler.

### Example 1

This example shows a COMMBLOCK for the EMPLOYEE database. Note that no attributes or initial values are declared for any of the variables.

```
COMMBLOCK OF EMPLOYEE;
     DCL 1 EMPLOYEE,
         3 RECORD_NAME,
         3 RETURN_CODE,
         3 FREE,
         3 LAST_RECRD_FLAG ,
         3 PASSWORD,
         3 NO_OF_RECCRDS,
         3 LF_POINTER,
         3 LEVEL,
         3 CTIME,
         3 CDATE,
         3 CYCLE,
         3 SEPARATOR,
         3 TERMINATOR,
         3 DAMAGE;
```

Here is the same COMMBLOCK declaration, augmented by the PLEX processor for input to the PL/I compiler. Note that the PLEX processor makes a comment out of the first two lines of the declaration. It inserts the control block and also adds attributes and initial values to the declarations for the COMMBLOCK fields.

```
/*
COMMBLOCK OF EMPLOYEE;
     DCL 1 EMPLOYEE,
*/
     DCL 1 EMPLOYEE_0,
         2  S2K0 CHAR(16) INIT ('EMPLOYEE'),
         2  S2K1(3) FIXED BIN(31) INIT (0,120,0),
         2 EMPLOYEE,
         3 RECORD_NAME          CHAR(32)      INIT (' '),
         3 RETURN_CODE          FIXED BIN(31) INIT (0),
         3 FREE                 CHAR(4)       INIT (' '),
         3 LAST_RECRD_FLAG      FIXED BIN(31) INIT (0),
         3 PASSWORD             CHAR(4)       INIT (' '),
         3 NO_OF_RECORDS        FIXED BIN(31) INIT (0),
         3 LF_POINTER           FIXED BIN(31) INIT (0),
         3 LEVEL                FIXED BIN(31) INIT (0),
         3 CTIME                CHAR(8)       INIT (' '),
         3 CDATE                CHAR(8)       INIT (' '),
         3 CYCLE                FIXED BIN(31) INIT (0),
         3 SEPARATOR            CHAR(4)       INIT (' '),
         3 TERMINATOR           CHAR(4)       INIT (' '),
         3 DAMAGE               FIXED BIN(31) INIT (0);
```

**Example 2**

Here is a COMMBLOCK for the EMPLOYEE database, declared this time with the PARM option. This option indicates that the COMMBLOCK will be used as a parameter in the procedure.

```
COMMBLOCK OF EMPLOYEE PARM;
     DCL 1 EMPLOYEE,
         2 RECORD_NAME,
         2 RETURN_CODE,
         2 FREE,
         2 LAST_RECRD_FLAG ,
         2 PASSWORD,
         2 NO_OF_RECORDS,
         2 LF_POINTER,
         2 LEVEL
         2 CTIME,
         2 CDATE,
         2 CYCLE,
         2 SEPARATOR,
         2 TERMINATOR,
         2 DAMAGE;
```

As expanded by the PLEX processor, the COMMBLOCK declaration includes attributes for the declared variables, but no initial values.

```
/*
COMMBLOCK OF EMPLOYEE PARM;
     DCL 1 EMPLOYEE,
*/
     DCL 1 EMPLOYEE_0,
         2  S2K0 CHAR(16)
         2  S2K1 (3) FIXED BIN(31),
         2 EMPLOYEE,
         3 RECORD_NAME           CHAR(32)      ,
         3 RETURN_CODE           FIXED BIN(31),
         3 FREE                  CHAR(4)       ,
         3 LAST_RECRD_FLAG       FIXED BIN(31),
         3 PASSWORD              CHAR(4)       ,
         3 NO_OF_RECORDS         FIXED BIN(31),
         3 LF_POINTER            FIXED BIN(31),
         3 LEVEL                 FIXED BIN(31),
         3 CTIME                 CHAR(8)       ,
         3 CDATE                 CHAR(8)       ,
         3 CYCLE                 FIXED BIN(31),
         3 SEPARATOR             CHAR(4)       ,
         3 TERMINATOR            CHAR(4)       ,
         3 DAMAGE                FIXED BIN(31),
```

### Example 3

This sample COMMBLOCK includes the optional fifteenth field as an indicator of the parent of a record retrieved by the program. The program declares the COMMBLOCK with the attribute EXTERNAL and initializes both the third and the fifth fields.

The first line of the declaration must include the full name of the database INVENTORY. However, in the second line, notice the five-character name INVNT. It will be extended to seven characters when the PLEX processor adds '_0' to it. In all cases, the COMMBLOCK name can be different from the database name. Also, no attributes are declared for any of the fields. In some cases, more than one variable is declared on a single line.

```
COMMBLOCK OF INVENTORY;
    DCL  1 INVNT  EXTERNAL,
         3 INV_RECORD_NAME,
         3 INV_RETURN_CODE,
         3 INV_FREE  INIT ('CHEK'),
         3 INV_LAST_RECORD_FLAG,
         3 INV_PASSWORD  INIT ('ABCD'),
         3 INV_NUM_OF_RECORDS,
         3 INV_LF_POINTER,
         3 INV_LEVEL,
         3 INV_CTIME,   3 INV_CDATE,  3 INV_CYCLE,
         3 INV_SEPARATOR,  3 INV_TERMINATOR,
         3 INV_DAMAGE_FLAG,
         3 INV_PARENT_RECORD;
```

In this expansion, the variable declarations are all aligned, and they now include attributes as well as the initial values chosen by the program.

```
/*
COMMBLOCK OF INVENTORY;
    DCL 1  INVNT  EXTERNAL,
*/
    DCL 1 INVNT_0 EXTERNAL,
         2  S2K0 CHAR(16) INIT ('INVENTORY'),
         2  S2K1(3) FIXED BIN(31) INIT (0,124,0),
         2 INVNT,
         3 INV_RECORD_NAME      CHAR(32)      INIT (' '),
         3 INV_RETURN_CODE      FIXED BIN(31) INIT (0),
         3 INV_FREE             CHAR(4)       INIT ('CHEK'),
         3 INV_LAST_RECORD_FLAG FIXED BIN(31) INIT (0),
         3 INV_PASSWORD         CHAR(4)       INIT ('ABCD'),
         3 INV_NUM_OF_RECORDS   FIXED BIN(31) INIT (0),
         3 INV_LF_POINTER       FIXED BIN(31) INIT (0),
         3 INV_LEVEL            FIXED BIN(31) INIT (0),
         3 INV_CTIME            CHAR(8)       INIT (' '),
         3 INV_CDATE            CHAR(8)       INIT (' '),
         3 INV_CYCLE            FIXED BIN(31) INIT (0),
         3 INV_SEPARATOR        CHAR(4)       INIT (' '),
         3 INV_TERMINATOR       CHAR(4)       INIT (' '),
         3 INV_DAMAGE_FLAG      FIXED BIN(31) INIT (0),
         3 INV_PARENT_RECORD    FIXED BIN(31) INIT (0);
```

As mentioned above, the PLEX processor adds '_0' to the declared name of the COMMBLOCK. Since the declared name has only five characters, the length of the resulting name is within the PL/I compiler restriction of seven characters in external names. If the declared name is longer than five characters, the resulting name would exceed the PL/I seven-character limit and the PL/I compiler would issue an error-level message. The seven-character limit on the length of declared COMMBLOCK names applies only to the case in which the COMMBLOCK is declared EXTERNAL, as it is in this example.

**Example 4**

A declaration of the COMMBLOCK for the EMPLOYEE database appears below. Note that the program uses level number 2 in the declarations for the COMMBLOCK fields.

```
COMMBLOCK OF EMPLOYEE;
    DCL 1 EMPLOYEE,
          2 RECORD_NAME,
          2 RETURN_CODE,
          2 FREE,
          2 LAST_RECRD_FLAG ,
          2 PASSWORD,
          2 NO_OF_RECORDS,
          2 LF_POINTER,
          2 LEVEL,
          2 CTIME,
          2 CDATE
          2 CYCLE,
          2 SEPARATOR,
          2 TERMINATOR,
          2 DAMAGE;
```

In the following output from the PLEX processor, note that a warning message indicates that the level number in each item declaration is 3, so the higher-level control block can be inserted in the correct place in the COMMBLOCK.

```
 COMMBLOCK OF EMPLOYEE;
      DCL 1 EMPLOYEE,
. . . . C
****   PLX202 ALL LOWER LEVEL NUMBERS HAVE BEEN INCREASED BY ONE
        3 RECORD_NAME          CHAR(32)        INIT (' '),
        3 RETURN_CODE          FIXED BIN(31)   INIT (0),
        3 FREE                 CHAR(4)         INIT (' ')
        3 LAST_RECRD_FLAG      FIXED BIN(31)   INIT (0),
        3 PASSWORD             CHAR(4)         INIT (' '),
        3 NO_OF_RECORDS        FIXED BIN(31)   INIT (0),
        3 LF_POINTER           FIXED BIN(31)   INIT (0),
        3 LEVEL                FIXED BIN(31)   INIT (0),
        3 CTIME                CHAR(8)         INIT (' '),
        3 CDATE                CHAR(8)         INIT (' '),
        3 CYCLE                FIXED BIN(31)   INIT (0),
        3 SEPARATOR            CHAR(4)         INIT (' '),
        3 TERMINATOR           CHAR(4)         INIT (' '),
        3 DAMAGE               FIXED BIN(31)   INIT (0);
```

**Example 4 (continued)**

The following input to the PL/I compiler shows that the control block was inserted.

```
/*
COMMBLOCK OF EMPLOYEE;
     DCL 1 EMPLOYEE,
*/
     DCL 1 EMPLOYEE_0,
         2  S2K0 CHAR(16) INIT ('EMPLOYEE'),
         2  S2K1(3) FIXED BIN(31) INIT (0,120,0),
         2 EMPLOYEE,
         3 RECORD_NAME          CHAR(32)      INIT (' '),
         3 RETURN_CODE          FIXED BIN(31) INIT (0),
         3 FREE                 CHAR(4)       INIT (' '),
         3 LAST_RECRD_FLAG      FIXED BIN(31) INIT (0),
         3 PASSWORD             CHAR(4)       INIT (' '),
         3 NO_OF_RECORDS        FIXED BIN(31) INIT (0),
         3 LF_POINTER           FIXED BIN(31) INIT (0),
         3 LEVEL                FIXED BIN(31) INIT (0),
         3 CTIME                CHAR(8)       INIT (' '),
         3 CDATE                CHAR(8)       INIT (' '),
         3 CYCLE                FIXED BIN(31) INIT (0),
         3 SEPARATOR            CHAR(4)       INIT (' '),
         3 TERMINATOR           CHAR(4)       INIT (' '),
         3 DAMAGE               FIXED BIN(31) INIT (0);
```

## Considerations

- The COMMBLOCK contains important debugging information for checking program logic. It should be printed in an understandable format when an unexpected return code is issued.

- The characters PARM, PARA, and PARAMETER cannot be used in a database name. The scanner looks ahead for these keywords as a signal for the end of the database name. For example, the database names PARADE or THE PARABOLA cannot be used.

- No storage attribute should be given to a COMMBLOCK declared as a parameter.

- A COMMBLOCK must contain 14 fields. If it does not, the PLEX processor gives an error message. You can also include the optional fifteenth field.

- A program can supply initial values for only the third and fifth fields in the COMMBLOCK. If it assigns an initial value to some other COMMBLOCK field, the PLEX processor issues an error message.

- If the name of a COMMBLOCK is not unique within the program, the PLEX processor issues an error message.

- When a Locate File is currently in use by the program, the fourth, sixth, and seventh fields of the COMMBLOCK are set for that Locate File. When a program uses one Locate File, then switches to a second one, those COMMBLOCK fields are set with the appropriate information for the second Locate File. If the program resumes retrievals with the first Locate File, the information for that Locate File is brought back into the COMMBLOCK. For example, a program can give GET commands to retrieve records for the first five addresses in the first Locate File, then work with another Locate File for a while, and later resume retrievals at the point where it stopped earlier in the first Locate File.

- The sixth field of the COMMBLOCK is set after the program issues a GET1 command with a where-clause.

- If the COMMBLOCK is declared with the attribute EXTERNAL, the declared name of the COMMBLOCK should be no longer than five characters. If it is longer, an error results during compilation.

- COMMBLOCKs can be placed in the source Library and copied into the program during PLEX processing with the $COPY directive (see **$COPY Directive** on page B-61 and **$COPY Library** on page B-95).

# SUBSCHEMA RECORD DECLARATIONS

A SUBSCHEMA RECORD declaration associates PL/I identifiers with a database schema record and the items in the schema record. The subschema record represents a schema record, and the items in the subschema record represent the corresponding items in the schema record. The subschema record declaration also allocates a storage area for items in the corresponding data records. Update and retrieval commands use subschema records as input and output areas, respectively, for data records in a database. A database can be accessed only through a subschema record.

- A subschema record can be passed as a parameter.
- Any valid PL/I storage class attribute can be given to a subschema record.
- The name of a subschema record can be different from the name of the corresponding schema record.
- The names of subschema items can be different from the names of the corresponding schema record items.
- Items can appear in any order in a subschema record, regardless of their corresponding order in a schema record.
- Subschema record items can be initialized in the declaration.
- Six different display formats are available for items of type DATE.

A subschema record (SSR) declaration has two parts. The first part associates the subschema record name with a database and specifies whether the subschema record is to be used as a parameter. The second part looks similar to a PL/I structure declaration. It consists of variables that you declare with appropriate attributes and that correspond to selected items of a single schema record.

The name (alias) of a variable in a subschema record can be the component name or C-number (the letter C followed by the component number) of a schema item, or it can be an alternate identifier that is acceptable in PL/I. If a component name contains blanks, it cannot be an acceptable PL/I identifier, but each C-number is. The use of an alternate identifier is especially helpful when the component name is not acceptable in PL/I and yet an identifier is required that is more meaningful than the C-number. If the alias is not the component name or C-number, the component name or C-number must appear, preceded by the keyword EQ, on a line below the alias in an item declaration.

Each item in a subschema record can be declared with any valid PL/I attributes. However, the attributes should be compatible with the item type of the corresponding item in the database. See **SYSTEM 2000 Item Types and PL/I Attributes** on page B-7 for information about compatibility of item types. A subschema item for a DATE item can be declared with any one of six date formats with a FORM statement. This statement must appear on a line below the item declaration line and can precede or follow an EQ statement if there is one.

The PLEX processor converts the subschema record declaration to a PL/I structure declaration. The structure name is created by adding _0 to the name of the subschema record. The PLEX processor creates a control block for the subschema record and its items to store information, such as the name, type, and length of each variable. The actual input/output area for the subschema record items follows the control block and appears as a

minor structure. The PLEX processor uses the subschema record name, aliases, and attributes specified in the subschema record declaration to generate this structure.

## Format

```
SUBSCHEMA [RECORD] name OF database-identifier [PARAMETER]

DECLARE     |1     name  [storage-attribute],
            |01

item-declarations
      ;
```

*name* is any valid PL/I identifier of 29 characters or less.

*database-identifier* is the name of a SYSTEM 2000 database or name of the COMMBLOCK.

*storage-attribute* is any valid PL/I storage class attribute. The default is the PL/I default storage attribute.

*item-declarations*

```
item-declaration
[item-declarations]
```

*item-declaration*

```
n alias declaration [INITIAL value]
[EQ  |C-number,      ]
     |component-name,

[FORM date-format, ]
```

*n* is any valid PL/I level number greater than 1.

All the items in a subschema record must have the same level number. If this level number is less than 3, then the PLEX processor substitutes a level number greater than the given level number by one. If the level numbers for all fields are not identical, the PLEX processor issues an error message.

*alias* is any valid PL/I identifier of 29 characters or less.

*declaration* is any acceptable PL/I declaration (without edit characters) compatible with the database definition for the item.

Abbreviations:
SUBSCHEMA = SSR
PARAMETER = PARM, PARA
DECLARE = DCL
INITIAL = INIT

*value* is an initial value compatible with the PL/I declaration and with the database definition for the item.

*C-number* is the letter C followed by a SYSTEM 2000 component number. The number can have up to four digits, including zeros. For example, C0001 is interpreted as C1 by SYSTEM 2000 software.

*component-name* is the name of a component in a SYSTEM 2000 database.

*date-format* is MMDDYYYY, MMDDYY, DDMMYYYY, DDMMYY, YYYYMMDD, or YYMMDD. The default date format is MMDDYY when the field is six bytes and MMDDYYYY when the field is eight bytes.

**[PARAMETER]** If this option is omitted, the subschema record cannot be received as a parameter. PARA and PARM are acceptable abbreviations for PARAMETER.

If the subschema record is to be received as a parameter, the called procedure must use the PARAMETER option in the first statement of the subschema record declaration. No storage attribute should be assigned to such a subschema record. When a subschema record is passed as a parameter, the invoking routine must use the suffixed name of the subschema record generated by the PLEX processor, for example, LIBRARY_0. The subschema record can be declared with a storage attribute in the invoking routine.

**EQ** *component-name* **or** *C-number* If this statement is omitted, the alias is assumed to be a component name or C-number.

## Examples

Sample SUBSCHEMA RECORD declarations for the EMPLOYEE database are presented here. Each example shows how the program actually declared the subschema record, then how the PLEX processor translated the declaration as input to the PL/I compiler.

### Example 1

In this example, the declaration of subschema record DIRECTORY includes four items, each of which is declared with appropriate attributes. Note that the alias LNAME is equated with the C-number C2 and that the alias OFFICE is equated with the component name OFFICE-EXTENSION.

```
SUBSCHEMA RECORD DIRECTORY OF EMPLOYEE;
DECLARE 1 DIRECTORY,
        3 C1 PIC'9999',
        3 LNAME CHAR(10),
          EQ C2 ,
        3 OFFICE PIC '999XXX999',
          EQ OFFICE-EXTENSION,
        3 FORENAME CHAR(20);
```

Here is the same subschema record declaration, augmented by the PLEX processor. Note that the PLEX processor makes a comment out of the first two lines of the program's declaration. It inserts the control block and makes comments out of the EQ lines on which the program indicated the C-number and the component name for two aliases.

```
/*
SUBSCHEMA RECORD DIRECTORY OF EMPLOYEE;
DECLARE 1 DIRECTORY,
*/
DCL 1 DIRECTORY_0,
        2  S2K(5),
        3  S2K0 CHAR(32) INIT ('DIRECTORY','C1','C2',
        'OFFICE-EXTENSION','FORENAME'),
        3  S2K(6) FIXED BIN(31) INIT (43,(2)0,4,0,134217728,224,(4)0,
        131076,172,(4)0,10,126,(4)0,9,79,(4)0,20),
        2 DIRECTORY,
        3 C1 PIC'9999',
        3 LNAME CHAR(10),
/*
          EQ C2 ,
*/
        3 OFFICE PIC'999XXX999',
/*
          EQ OFFICE-EXTENSION,
*/
        3 FORENAME CHAR(20);
```

### Example 2

This example shows a subschema record DIRECTORY, declared with the PARM option. This option indicates that the subschema record will be used as a parameter in the procedure.

```
SUBSCHEMA RECORD DIRECTORY OF EMPLOYEE PARM;
DECLARE 1 DIRECTORY,
        3 C1 PIC'9999',
        3 LNAME CHAR(10),
          EQ C2 ,
        3 OFFICE PIC'999XXX999',
          EQ OFFICE-EXTENSION,
        3 FORENAME CHAR(20);
```

As expanded by the PLEX processor, the declaration is identical to the one in Example 1, except that the control block variables are not initialized.

```
/*
SUBSCHEMA RECORD DIRECTORY OF EMPLOYEE PARM;
DECLARE 1 DIRECTORY,
*/
DCL 1 DIRECTORY_0,
        2  S2K(5),
        3  S2K0 CHAR(32),
        3  S2K1(6) FIXED BIN(31),
        2 DIRECTORY,
        3 C1 PIC'9999',
        3 LNAME CHAR(10),
/*
          EQ C2 ,
*/
        3 OFFICE PIC'999XXX999',
/*
          EQ OFFICE-EXTENSION,
*/
        3 FORENAME CHAR(20);
```

**Example 3**

This example shows a subschema record called ADDRS declared with the attribute EXTERNAL. The subschema record name given in the first two lines of the declaration consists of five characters. The name in the DCL line will have seven characters when the PLEX processor adds _0 to it. Note that the program uses level number 5 rather than 3, which was used in the two previous examples. The alias for each subschema item is equated with a corresponding component name or C-number. Note also that since an EQ line occurs last in this declaration, the final semicolon is placed on a separate line.

```
SSR  ADDRS OF EMPLOYEE;
DCL 1  ADDRS EXTERNAL,
    5 LASTNAME CHAR(12),
      EQ LAST NAME,
    5 FNAME CHAR(25) VARYING,
      EQ C3 ,
    5 STREET CHAR (20),
      EQ STREET ADDRESS,
    5 CITYSTATE CHAR(15),
      EQ C15,
    5 ZIP PICTURE"(5)9',
      EQ ZIP CODE,
    ;
```

For this sample declaration, the PLEX processor turns the EQ lines into comments.

```
/*
SSR  ADDRS  OF EMPLOYEE;
DCL 1  ADDRS  EXTERNAL,
*/
DCL 1 ADDRS_0 EXTERNAL,
    2  S2K(6),
    3  S2K0 CHAR(32) INIT ('ADDRS','LAST NAME','C3','STREET ADDRESS',
    'C15','ZIP CODE'),
    3  S2K1(6) FIXED BIN(31) INIT (79,(2)0,5,0,134217728,280,(4)0,12,
    236,(4)0,1114139,207,(4)0,20,171,(4)0,15,130,(4)0,131077),
    2 ADDRS,
    5 LASTNAME CHAR(12),
/*
      EQ LAST NAME,
*/
    5 FNAME CHAR(25) VARYING,
/*
      EQ C3 ,
*/
    5 STREET CHAR (20),
/*
      EQ STREET ADDRESS,
*/
    5 CITYSTATE CHAR (15),
/*
      EQ C15 ,
*/
    5 ZIP PICTURE'(5)9',
/*
      EQ ZIP CODE,
*/
   ;
```

As mentioned above, the PLEX processor adds _0 to the declared name of the subschema record. Since the declared name had only five characters, the length of the resulting name is within the PL/I compiler restriction of seven characters in external names. If the declared name is longer than five characters, the resulting name would exceed the PL/I seven-character limit, and the PL/I compiler would issue an error-level message. The seven-character limit on the length of the subschema record name applies only to the case in which the subschema record is declared EXTERNAL, as it is in this example.

**Example 4**

In the sample subschema record PAYINFO, the program explicitly specifies the default AUTOMATIC attribute. A date format is specified for item HIREDATE that is different from the default. The declaration equates the items with their respective component names or C-numbers. The program uses level number 2 in the item declarations.

```
SUBSCHEMA PAYINFO OF EMPLOYEE;
DCL 1 PAYINFO AUTOMATIC,
    2 HIREDATE CHAR(8) ,
      EQ HIRE DATE ,
      FORM YYYYMMDD,
    2 VACTOTAL FIXED DECIMAL (7,3),
      EQ C11 ,
    2 SICTOTAL FIXED DECIMAL (7,3),
      EQ C12 ,
    2 SSN PICTURE'(11)X',
      EQ SOCIAL SECURITY NUMBER,
    ;
```

Note that in the output from the PLEX processor, a warning message indicates that the level numbers in the item declarations were increased to 3, so the control block can be inserted in the correct place in the subschema record declaration.

```
SUBSCHEMA PAYINFO OF EMPLOYEE;
DCL 1 PAYINFO AUTOMATIC,
    3 HIREDATE CHAR(8) ,
....C
****    PLX202 ALL LOWER LEVEL NUMBERS HAVE BEEN INCREASED BY ONE
      EQ HIRE DATE ,
      FORM YYYYMMDD,
    3 VACTOTAL FIXED DECIMAL (7,3),
      EQ C11 ,
    3 SICTOTAL FIXED DECIMAL (7,3),
      EQ C12 ,
    3 SSN PICTURE'(11)X',
      EQ SOCIAL SECURITY NUMBER
    ;
```

**Example 4 (continued)**

The following input to the PL/I compiler shows that the control block was inserted and that the FORM line and EQ lines were all turned into comments.

```
/*
SUBSCHEMA PAYINFO OF EMPLOYEE;
DCL 1 PAYINFO AUTOMATIC,
*/
DCL 1 PAYINFO_0 AUTOMATIC,
    2  S2K(5),
    3  S2K0 CHAR(32) INIT ('PAYINFO','HIRE DATE','C11','C12',
    'SOCIAL SECURITY NUMBER'),
    3  S2K1(6) FIXED BIN(31) INIT (27,(2)0,4,0,134217728,224,(4)0,
    983048,176,(4)0,459524,124,(4)0,459524,72,(4)0,11),
    2 PAYINFO,
    3 HIREDATE CHAR(8) ,
/*
      EQ HIRE DATE ,
*/
/*
      FORM YYYYMMDD,
*/
    3 VACTOTAL FIXED DECIMAL (7,3),
/*
      EQ C11 ,
*/
    3 SICTOTAL FIXED DECIMAL (7,3),
/*
      EQ C12 ,
*/
    3 SSN PICTURE'(11)X',
/*
      EQ SOCIAL SECURITY NUMBER,
*/
    ;
```

### Example 5

A subschema record can be declared without items, regardless of whether the corresponding schema record has items. For this kind of subschema record, the declared name must be either the component name or the C-number of the schema record. The program specifies the C-number C400 in this sample subschema record declaration.

```
SSR C400 OF EMPLOYEE;
DCL 1 C400;
```

As expanded by the PLEX processor, the declaration includes the control block.

```
/*
SSR C400 OF EMPLOYEE;
DCL 1 C400;
*/
DCL 1 C400_0,
    2  S2K(1),
    3  S2K0 CHAR(32) INIT ('C400'),
    3  S2K1(6) FIXED BIN(31) INIT (0,(4)0,134217728);
```

## Considerations for Subschema Records

- A subschema record declaration can appear anywhere in a program as long as the COMMBLOCK associated with the subschema record is declared first.

- The subschema record name can be different from the schema record name if the subschema record has at least one item in it. SYSTEM 2000 software associates a subschema record containing items with the corresponding schema record by associating item names, rather than by matching record names.

- You can declare a subschema record without any items in it. The syntax for this kind of subschema record is as follows:

```
SUBSCHEMA [RECORD]  name OF database-name [PARAMETER] ;

DECLARE  |1   name  [storage-attribute];
         |01
```

  You can use a subschema record without items when there is no item in the schema record or when retrieval of the data values corresponding to schema items is not required. Such a subschema record can be used in a where-clause or for address retrieval before an update. Since the subschema record has no items, its name must be either the component name or the C-number of the corresponding schema record. In this case, SYSTEM 2000 software associates the two records by matching their names. If the name of such a subschema record is not the component name or C-number of the corresponding schema record, any command referring to the subschema record results in return code 13.

  On the DECLARE line, a semicolon must appear immediately after the subschema record name or after the optional storage attribute specification.

- A program can have more than one subschema record corresponding to the same schema record. The only requirement for these subschema records is that they have unique names (not necessarily the name of the corresponding schema record). If the name of a subschema record is not unique within the program, the PLEX processor issues an error message.

- If the subschema record name specified in the SUBSCHEMA RECORD line does not match the name specified in the DECLARE line, the PLEX processor gives an error message.

- Subschema records can be placed in the source Library and copied into the program during PLEX processing with the $COPY directive (see **$COPY Directive** on page B-61 and **$COPY Library** on page B-95).

- Subschema record declarations must end with a semicolon. If the last declared item includes a FORM or EQ option, the semicolon must be on a separate line.

---

Abbreviations:

SUBSCHEMA = SSR
PARAMETER = PARM, PARA
DECLARE = DCL

- When a subschema record is passed as a parameter, the suffixed name of the subschema record generated by the PLEX processor must be used. For example, a subschema record named DIRECTORY must be referred to as DIRECTORY_0 when it is passed as a parameter. The called procedure must declare the subschema record with the PARAMETER option. No other storage attribute need be declared for a subschema record declared with the PARAMETER option.

- The characters PARM, PARA, and PARAMETER cannot be used in the database name. The scanner looks ahead for these keywords as a signal for the end of the database name.

- If a subschema record is declared with the attribute EXTERNAL, the name of the subschema record should be no longer than five characters. If it is longer, an error results during compilation.

## Considerations for Items in Subschema Records

- If the alias of a subschema item is different from the corresponding schema item, you must specify the EQ option to establish the correspondence between that alias and the component name or C-number of the schema item. In this situation, if you do not specify the EQ option, return code 13 results from each command referring to this subschema record.

- If the last declared item in a subschema record includes an EQ or FORM specification, the semicolon ending the subschema record declaration must appear on a separate line.

  In addition, if the PL/I Optimizing compiler Release 3.0 is being used and Program Temporary Fix 67 was applied to the compiler, the comma following the attributes of the last item must appear on the same line as the EQ or FORM specification, as shown here:

```
SUBSCHEMA RECORD EDUCATION OF EMPLOYEE;

      :
      :
      :

3  MINOR CHAR(20)

   , EQ MINOR FIELD,
;
```

  When the EQ specification is turned into a comment by the PLEX processor, the leading comma is included in the comment.

  If, however, the comma appears on the same line as the item attributes, it is not included in the EQ comment. The PL/I Optimizing compiler issues an error message pointing out that a comma immediately precedes a semicolon (in this case, with an intervening comment).

- The attributes of a subschema record variable must be compatible with the item type of the corresponding schema item (see **SYSTEM 2000 Item Types and PL/I Attributes** on page B-7). If the INITIAL attribute is used, the value provided must be compatible with the declaration. The PLEX processor does not check the validity of the PL/I attributes or the initial values. If the declarations are not valid in PL/I, then errors occur during compilation. Incompatible item types result in run-time errors.

- When a subschema record is first referenced at execution time, all the items of the subschema record are validated. This is called "the opening of a subschema record." If the attributes of a subschema item are not compatible with the item type or picture of the corresponding item in the database (as discussed in **SYSTEM 2000 Item Types and PL/I Attributes** on page B-7), return code 17 is placed in the COMMBLOCK for that subschema record. The ordinal of that item within its subschema record is placed in the eighth field of the COMMBLOCK.

- The names of items specified in a PLEX command must appear exactly as they do in the subschema item declarations. The PLEX processor issues an error message if a command includes an alias that does not appear in a subschema record declaration.

  For C-numbers declared as aliases for subschema items, SYSTEM 2000 software allows some flexibility that PL/I, and therefore the PLEX processor, do not. For example, if you declare the alias C010 for a subschema item corresponding to C10 in the database schema, the program does not need to include the number C10 with EQ in the item declaration. SYSTEM 2000 software understands the implicit equivalence at execution time. When a PLEX command includes a C-number such as C010, SYSTEM 2000 software knows that it refers to component C10 in the database. If the number C010 is declared as the alias, this number must be used in a PLEX command referring to that item. In this case, if C10 is used, the PLEX processor issues an error message, since the alias is actually C010 in the subschema item declaration.

  A C-number declared as an alias can have up to four digits, besides C. Thus, the component whose C-number is C1 can be declared in a subschema record with the alias of C1, C01, C001, or C0001. Again, no EQ option is necessary to equate C1 with any of its variations in the subschema record declaration.

- If the subschema record refers twice to the same database item by specifying the same alias, component name, or C-number twice, the PLEX processor issues an error message. This error can occur in the following three situations:

  - The same alias is declared twice in the same subschema record.

  - The subschema record includes an EQ option specifying a component name or C-number also declared as the alias for another item in the same subschema record.

  - Two EQ specifications in the subschema record specify the same component name or C-number.

- Return code 13 results from a command for a subschema record that refers to the same database item twice, that is, if the component name is specified in one item declaration and the associated C-number is specified in another item declaration. The same error results if different forms of the same C-number appear in different item declarations in the same subschema record. This error is not detected in preprocessing because the PLEX processor does not access the database definition and therefore does not detect such a double reference. This error can occur in four situations:

  - The component name and C-number are both used as item aliases in the same subschema record,

  - The component name or C-number are both used as item aliases in the same subschema record,

  - The component name and C-number are named with the EQ option in two different item declarations,

  - The C-number C1, for example, is included in one item declaration, while a variant such as C001 is included in another item declaration.

- The level numbers for all items in a subschema record declaration must be identical. If they are not, the PLEX processor issues an error message.

- Return code 13 results from a command for a subschema referring (by C-number or component name) to an item not in the database schema.

- When a date format is specified in an item declaration, the length of the item (either six or eight characters) must match the length of the FORM specification. If the lengths are not the same, the PLEX processor issues an error message.

- Up to 1000 items can be declared in a subschema record, provided that all of these belong to the same schema record. If a program declares more than 1000 items in a subschema record, the PLEX processor issues an error message.

# THE START S2K COMMAND

Detailed information about the START S2K command is presented in the main text of this manual. The examples given here show various ways of issuing the START S2K command in a PL/I program.

The START S2K command requests a number of extra Locate Files to be used by the program and specifies the environment in which the program will run. The START S2K must be issued before any other executable PLEX command. However, it can appear after the declarations of COMMBLOCKs and subschema records.

**Example 1**

The START S2K command in this example contains constants.

```
COMMBLOCK OF . . .

      :

SSR . . .

      :

START S2K (5) USE (1);          (First PLEX command executed, preceding
(Check S2KRTC for return code)   all other PLEX commands; requests five
                                 Locate Files in addition to Locate
                                 File 0; specifies Multi-User)

OPEN . . .

      :

   Program statements
```

**Example 2**

This example uses variables in the START S2K command.

```
COMMBLOCK OF . . .

      :

SSR . . .

      :

DCL LFNUM FIXED BIN(15);
DCL ENVIRON FIXED BIN(15);

LFNUM=13;
ENVIRON=0;

START S2K (LFNUM) USE (ENVIRON);  (First PLEX command executed; requests
(Check S2KRTC for return code)     thirteen extra Locate Files (LFNUM=13);
                                   Specifies single-user (ENVIRON=0))

(Check S2KRTC for return code)

OPEN . . .

      :

   Program statements
```

**Example 3**

This example illustrates using a parameter for the environment number. To enable a PL/I PLEX program to be run in either a single-user or a Multi-User environment without recompilation, you can pass a parameter to the program from the JCL.

In this example, note the parameter (1 for Multi-User) in the EXEC job control statement. This parameter becomes available through the PROCEDURE statement. The PLEX program can then use the parameter value in the START S2K command. The program logic shown here provides for the case in which a parameter is not specified (LENGTH(PARAM)=0) in the JCL; in this case, ENVIRON is set to zero for the START S2K command.

```
//EXEC . . . PARM.GO='/1'
       :
       :
TEST:  PROCEDURE (PARAM) OPTIONS (MAIN);
       :
       :
DCL PARAM CHAR (250) VARYING;
DCL ENVIRON FIXED BIN(31);
       :
       :
IF LENGTH (PARAM)=0 THEN ENVIRON=0;
ELSE
IF PARAM='1' THEN ENVIRON=1;
ELSE ENVIRON=0;
START S2K . . . USE (ENVIRON);
```

# INCLUSION LISTS FOR RETRIEVALS AND UPDATES

You can use an inclusion list to indicate the subschema items that will be used in a retrieval or update operation.

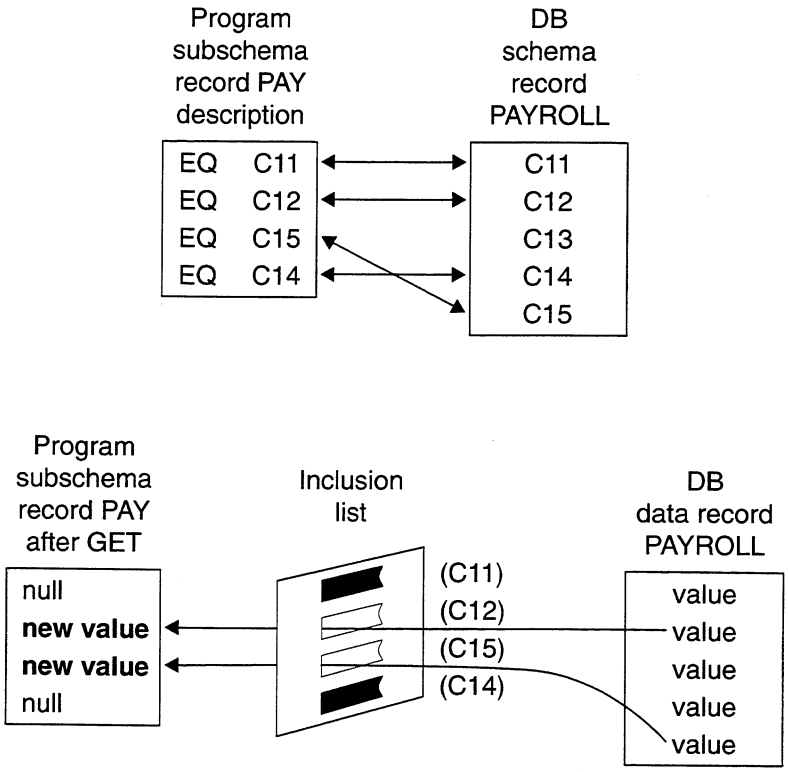
## Format

```
|item-list

|USING |designation-bit-string
       |word-table
```

You can use an inclusion list with the retrieval commands GET1, GET, GETA, GETD, and GETP, and also with the update commands MODIFY, REMOVE, and INSERT. If you do not specify an inclusion list for the command, all items in the subschema record are affected by the command. However, if you specify an inclusion list, it indicates the items of the subschema record that will be used in that operation. The process of retrieving values with an inclusion list is shown in Illus. B.2 on page B-46.

**Illus. B.2** Retrieval with an Inclusion List




In this illustration, the subschema record contains four of the five items in the corresponding data record. However, the inclusion list screens out two of those four values, allowing the other two to be passed to the subschema record.

You can specify several kinds of inclusion list:

- an item list, which is a series of aliases of subschema items that will be affected.
- a designation bit string, which is a table of component names or C-numbers of subschema items that will be affected.
- a designation vector word table, which is a table of one-word indicators (containing 1 or 0) corresponding to all items in the subschema record. A 1 means that the item will be affected, and a 0 means that the item will not be affected. In each indicator, only the low-order bit is checked to see if it is zero or one.

An item list is static; that is, it must be encoded in the PLEX command and parsed by the PLEX processor. If you want to generate an inclusion list dynamically (for example, based on a varying set of data), you can use a designation bit string or vector.

## Item List

*item-list*

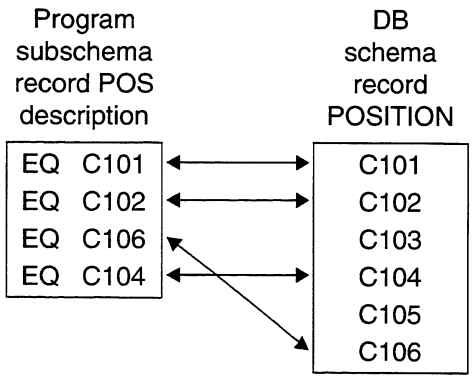
An item list is part of the command. It is a series of aliases, separated by commas, for subschema items that are to be included in the command's operation. If an item list includes an item that does not belong to the specified subschema record, the PLEX processor issues an error message. For example, Illus. B.3 shows the items of the schema record called POSITION and the corresponding items of the subschema record called POS.

Four items are included in the subschema record, and, therefore, four values would ordinarily be retrieved from the corresponding data record. However, in this example, the GET1 command includes an item list.

```
GET1 POS C102, C106 NEXT.
```

According to the item list, the values for subschema items C102 and C106 will be retrieved into the subschema record area, and the other two values will not, as shown in Illus. B.4.

**Illus. B.3** Correspondence between the Schema Record and the SSR Description




**Illus. B.4** Processing GET1 with an Item List

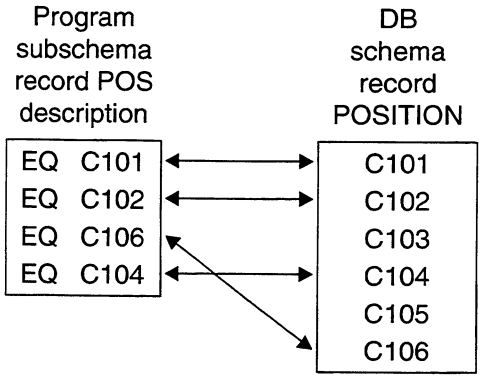
```
GET1 POS C102, C106 NEXT;
```

| | Program subschema record POS after GET1 |
|---|---|
| (C101) | null |
| (C102) | **new value** |
| (C106) | **new value** |
| (C104) | null |

This example illustrates a REMOVE command with an item list. Only the specified items become null in the corresponding data record, as Illus. B.5 shows.

**Illus. B.5** Processing REMOVE with an Item List

**REMOVE POS C102, C106;**




DB data record after REMOVE

| | |
|---|---|
| value | (C101) |
| **null** | (C102) |
| value | (C103) |
| value | (C104) |
| value | (C105) |
| **null** | (C106) |

## Designation Bit String

---

**USING** *designation-bit-string*

---

You specify the name of the designation bit string in the command, following the keyword USING. It must be a bit string identifier, with a length equal to or greater than two plus the number of items in the subschema record. One bit is used for each item in the subschema record, plus two prefix bits to indicate to SYSTEM 2000 software that a designation bit string is being used. Unpredictable results will occur if you specify a designation bit string that has fewer than the required number of bits. Also, bit string identifiers should have the ALIGNED attribute.

At the time the command is executed, the first two bits must have values of 1 and 0, respectively. The rest of the bit string must consist of 0s and 1s, where the third bit corresponds to the first item in the subschema record, the fourth bit to the second item, and so on. A 0 in a bit position indicates that the corresponding item is not to participate in the command, and a 1 indicates that it is.

For example, suppose the subschema record EDUC contains the following items:

**SSR EDUC**

| |
|---|
| C411 |
| C412 |
| C415 |
| C414 |

The PL/I PLEX program can include a designation bit string that specifies inclusion of only three of the four subschema items as follows:

```
          :

DCL 1 EDUC_NAME_BITS,
    05  PREFIX_BITS     BIT(2)   INIT ('10'B),.
    05  EDUC_BITS       BIT(4);

          :

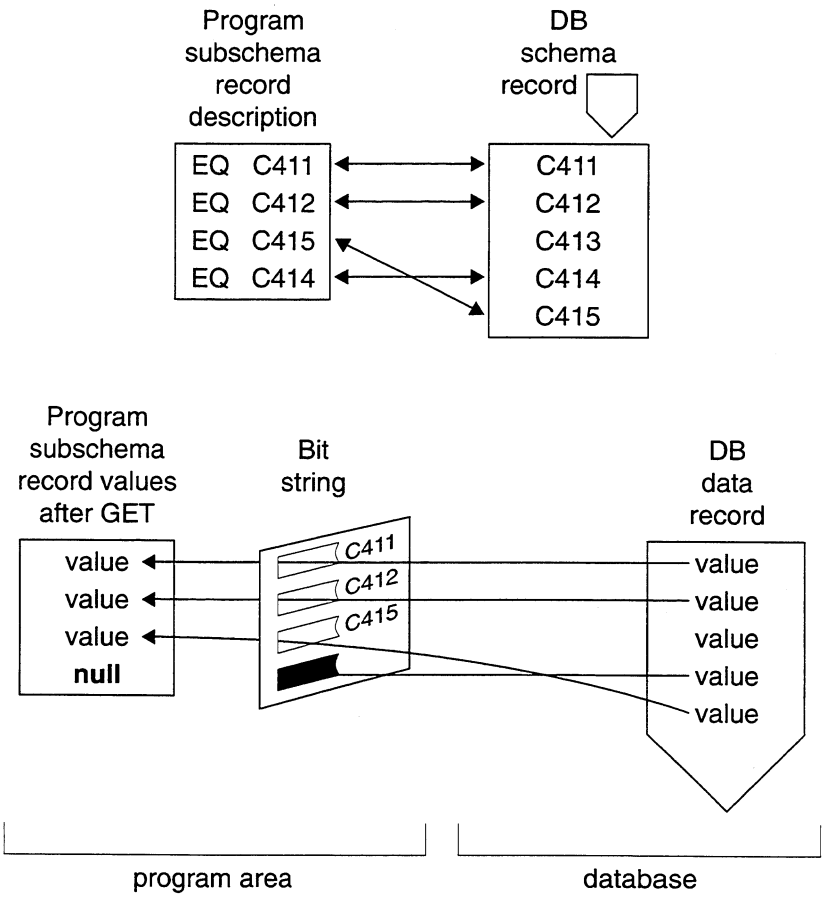
EDUC_BITS='1110'B;

          :

GETD EDUC USING EDUC_NAME_BITS NEXT;
```

According to this sample designation bit string, the values for subschema items C411, C412, and C415 will be brought into the subschema record area, but the value for C414 will not, as shown Illus. B.6.

**Illus. B.6** Processing GETD with a Designation Bit String



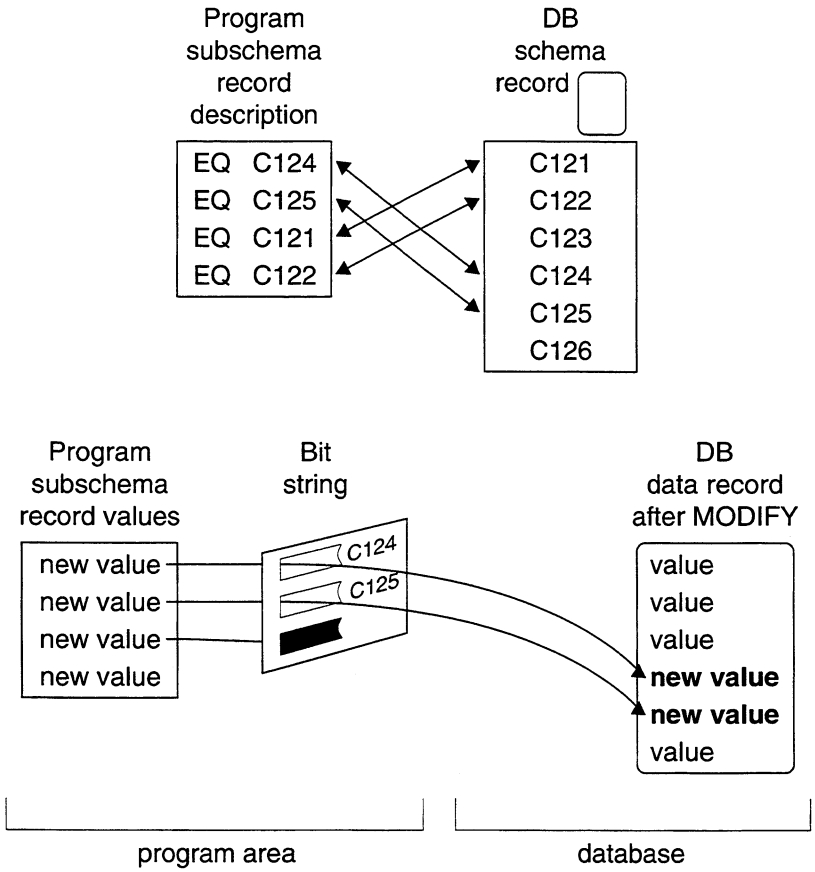
Similarly, for a MODIFY command, you can declare a designation bit string and use it to change two values in a data record, as shown in Illus. B.7.

**Illus. B.7** Processing MODIFY with a Designation Bit String

```
      .
      .
      .
DCL PAYROLL_NAME_ITEMS BIT(6) ALIGNED;

      .
      .
      .
PAYROLL_NAME_ITEMS='101100'B;

MODIFY PAYROLL USING PAYROLL_NAME_ITEMS;
```



## Designation Vector

---

**USING** *designation-vector*

---

Like a designation bit string, you specify the name of a designation vector in a command, following the keyword USING. It is an array of one-word indicators, one indicator for each item in the specified subschema record. Each word contains zero (0) or one (1). In each indicator, only the low-order bit is checked to see if it is zero or one.

An item is included in the operation of the command if its indicator in the table is one. Conversely, an item is excluded from the operation if its indicator is zero. In the PLEX program, you must specify FIXED BINARY (31) as the attributes of the designation vector.

For example, suppose the description of the subschema record PAYROLL contains the following items:

**SSR PAYROLL**

| |
|---|
| C121 |
| C124 |
| C125 |
| C126 |
| C122 |
| C123 |

You can include a word table in the program, specifying that values for only four of the six items be retrieved.

```
        :

DCL PAYROLL_WORD_ITEM (6) FIXED BINARY (31);

        :

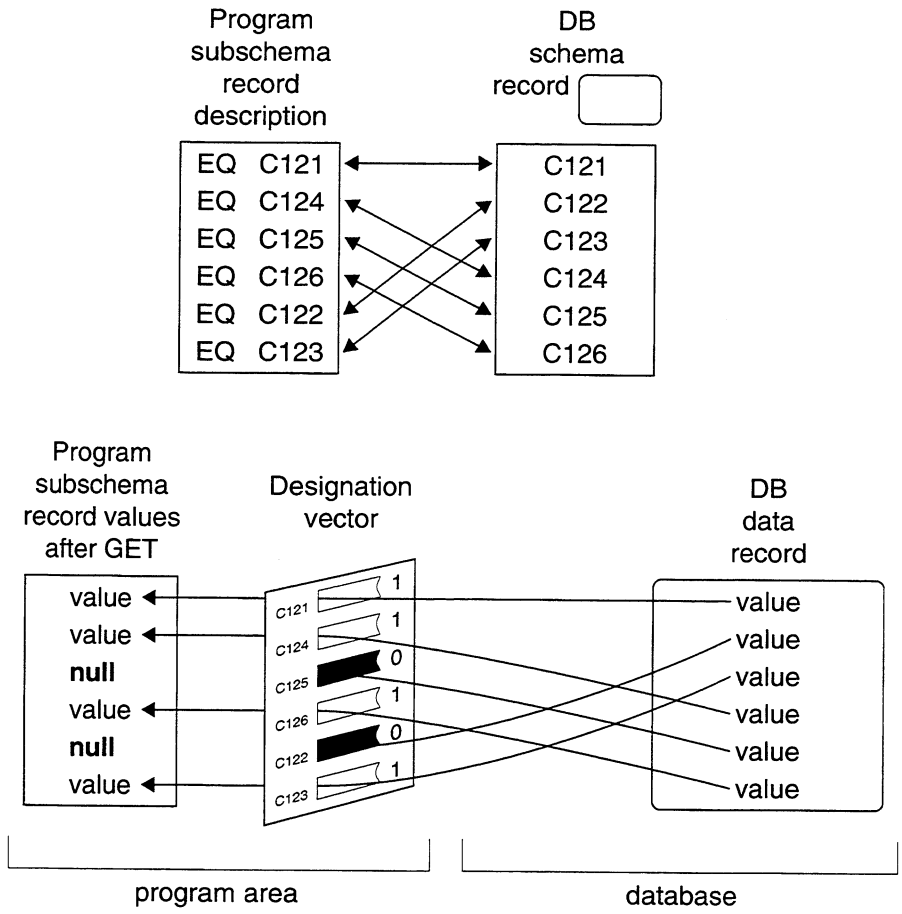
PAYROLL-WORD-ITEM (1)=1;
PAYROLL-WORD-ITEM (2)=1;
PAYROLL-WORD-ITEM (3)=0;
PAYROLL-WORD-ITEM (4)=1;
PAYROLL-WORD-ITEM (5)=0;
PAYROLL-WORD-ITEM (6)=1;
```

Here is a sample retrieval command that refers to that designation vector.

```
GETA PAYROLL USING PAYROLL_WORD_ITEM NEXT;
```

According to the sample designation vector, the values for subschema items C121, C124, C126, and C123 are brought into the subschema record area, but the values for C125 and C122 are not, as shown in Illus. B.8 on page B-53.

**Illus. B.8** Retrieval Processing with a Designation Vector



Similarly, for an INSERT command, you can specify a designation vector and use it to insert a new data record, as illustrated here.

**Illus. B.9** Processing INSERT with a Designation Vector

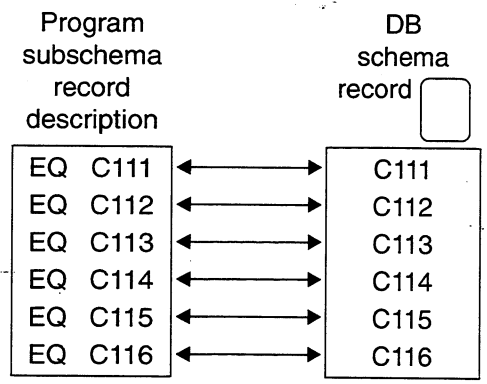
```
        :

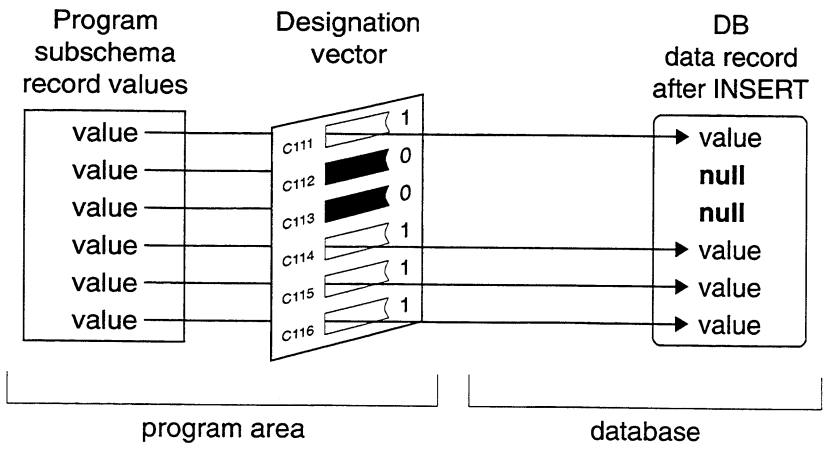
DCL WORD_ITEM(6) FIXED BINARY(31);

        :

WORD_ITEM(1)=1;
WORD_ITEM(2)=0;
WORD_ITEM(3)=0;
WORD_ITEM(4)=1;
WORD_ITEM(5)=1;
WORD_ITEM(6)=1;


INSERT [ ]     USING WORD_ITEM;
```



# INCLUSION LIST IN LINK COMMANDS

## Format

```
link verb  [(m)] [source-control] source SSR TO [/HOLD]
   [target-control] target-SSR [inclusion-list] via-clause.
```

```
inclusion-list  =   |item-list
                    |USING |designation-bit-string
                           |designation-vector
```

See **Inclusion Lists for Retrievals and Updates** on page B-45 for details about inclusion lists.

Retrieval of data for the source subschema record is accomplished with a LOCATE and a GET FIRST of the target subschema record of the LINK command. If you specify an inclusion list in the LINK command, it is used in the implied GET FIRST retrieval. The inclusion list has the same syntax and semantics here as in any retrieval command. If you do not specify an inclusion list, the entire subschema record participates in the retrieval.

# THE FOR COMMAND

The FOR command is an aid for isolating return code checking from the program logic. Also, it automatically initializes an error recovery routine (user-supplied) when specified return codes occur.

Refer to Chapter 12, "FOR, $FOR, and $CFOR" for a complete discussion of the FOR command, and the $FOR and $CFOR directives that replace it for Release 12.1.

## Format

```
FOR database-identifier|RETURN-CODE code-list |GO TO label
                       |NEW                   |NAME variable
                       |REVERT                |IGNORE
                       |CANCEL                |USE routine-name
```

*database-identifier* is the name of a SYSTEM 2000 database or name of the COMMBLOCK.

```
code-list   [code   |-- code  ] [, code-list]
                    |THRU code
```

*code* is a valid PLEX return code. The return code on the right of THRU or -- must be greater than that on the left.

*label* is the label of an error-processing routine in the same procedure that has the associated FOR command.

*variable* is a PL/I variable containing the left-justified, blank-filled name or alias of a module in JOBLIB, STEPLIB, or the system link library.

*routine-name* is the name of a PL/I error processing routine declared as ENTRY EXTERNAL in the procedure that has the FOR command.

In the following example, the module named ERRORS must be linked with the module containing the FOR command. If a return code other than 00 or 04 occurs for the EMPLOYEE database, ERRORS is called.

```
FOR EMPLOYEE RC 1-3, 5-127 USE ERRORS;
```

## NAME or USE Option

When a FOR command includes the NAME option, SYSTEM 2000 software uses the specified module for error processing. This module is dynamically loaded from the JOBLIB, STEPLIB, or system link library the first time it is invoked. It is deleted only when the program issues another FOR command (with REVERT or CANCEL) that deactivates the associated level of FOR processing for the database. Each time the module is invoked, it is passed four parameters: S2KDUM, the database COMMBLOCK, the SSR specified in the PLEX command causing the return code, and a table of addresses pointing to the remaining parameters for the PLEX command. This same list of parameters is also passed to a routine invoked through the use of the USE option.

For a complete discussion of the FOR command, $FOR, and $CFOR directives, more examples, and considerations, see Chapter 12, "FOR, $FOR, and $CFOR."

# PL/I PLEX PROCESSOR DIRECTIVES

You code the PLEX processor directives like PLEX commands in your source program. These directives become signals to the PLEX processor. This section discusses each PL/I PLEX processor directive.

## $BLOCKS Directive

### Format

```
$BLOCKS;
```

In response to the $BLOCKS directive, the PLEX processor generates the PL/I structure for the S2KDUM control block. Preceding the S2KDUM declaration is the declaration of the interface to be invoked by each PLEX command: S2KPL for non-CICS programs, PLXPBLD for CICS programs. The PLEX processor also generates other declarations, depending on other PLEX processor directives included in the program.

One $BLOCKS directive must appear in each external procedure submitted to the PLEX processor. If $BLOCKS is not included in a separately compiled procedure or if it appears twice, the PLEX processor issues an error message.

If your program issues the $STORAGE directive specifying STATIC for one or more control variables, the declarations of those STATIC variables appear adjacent to the declaration of S2KDUM. Declarations for control variables that are not specified as STATIC in the $STORAGE directive appear in the generated code adjacent to their related PLEX commands.

Illus. B.10 on page B-58 shows the code generated by the PLEX processor in response to the $BLOCKS directive. The PLEX processor converts the $BLOCKS directive into a PL/I comment. In this case, $BLOCKS was issued in an external procedure that did not include a $CICS ON directive, or a $STORAGE directive applying to S2KDUM. The generated code consists of the following:

- the declaration of the interface S2KPL as a separately compiled module
- the declaration of S2KDUM, showing initial values for each of the variables in this control block.

Each of the variables in S2KDUM is described in Illus. B.11 on page B-59.

**Illus. B.10** Code Generated by the PLEX Processor for $BLOCKS

```
/*
$BLOCKS;
*/
     DCL S2KPL ENTRY OPTIONS(ASM,INTER);
     DCL 1 S2KDUM,
           2 S2KDU1       FIXED BIN(31) INIT (0),
           2 S2KRTC       FIXED BIN(31) INIT (0),
           2 S2KCOUNT     FIXED BIN(31) INIT (0),
           2 S2KDU2       FIXED BIN(31) INIT (0),
           2 S2KDU3       FIXED BIN(31) INIT (0),
           2 S2K1         FIXED BIN(31) INIT (3),
           2 S2KDU4       FIXED BIN(31) INIT (0),
           2 S2K2         FIXED BIN(31) INIT (0),
           2 S2K3         CHAR(8)       INIT ('        '),
           2 S2K4         CHAR(4)       INIT ('0290'),
           2 S2K5         CHAR(8)       INIT ('79115   '),
           2 S2K6         CHAR(8)       INIT ('2004292 '),
           2 S2K7         CHAR(4)       INIT ('    '),
           2 S2KDU5       FIXED BIN(31) INIT (0),
           2 S2KDU6       FIXED BIN(31) INIT (-1),
           2 S2KDU7       CHAR(4)       INIT ('    '),
           2 S2K8         FIXED BIN(31) INIT (0),
           2 S2K9         FIXED BIN(31) INIT (0),
           2 S2K10        FIXED BIN(31) INIT (0),
           2 S2KDU8       FIXED BIN(31) INIT (0),
           2 S2KDU9       FIXED BIN(31) INIT (0),
           2 S2KSRT       FIXED BIN(31) INIT (0),
           2 S2K11        FIXED BIN(31) INIT (0),
           2 S2K12        FIXED BIN(31) INIT (0),
           2 S2K13        FIXED BIN(31) INIT (0),
           2 S2KLKH(10)   FIXED BIN(31) INIT ((10)-1);
```

**Illus. B.11** Description of S2KDUM Control Block Items

| Offset HEX | Offset DEC | Name | Length/ Alignment | Type/ Value | Field Description |
|---|---|---|---|---|---|
| 0 | 0 | S2KDU1 | 4 | Binary | SYSTEM 2000 operation code; set by the expanded PLEX code at execution time. |
| 4 | 4 | S2KRTC | 4 | Binary | SYSTEM 2000 return code; set by a PLEX command execution. |
| 8 | 8 | S2KCOUNT | 4 | Binary | Value of S2KCOUNT; set by the PLEX program. |
| C | 12 | S2KDU2 | 4 | Binary | Stack or Locate File number; set by the expanded code at run time. |
| 10 | 16 | S2KDU3 | 4 | Binary | Maximum number for Locate File; set at run time by the expanded START S2K code. Also, the number of the target stack or Locate File referred to in a COPYS or COPYL command; set at run time. |
| 14 | 20 | FILLER | 4 | Binary<br>1<br>2<br>3<br>4<br>5 | PLEX program type; set by PLEX processor.<br>FORTRAN<br>COBOL<br>PL/I - optimizing or checkout<br>PL/I (F)<br>ASSEMBLER |
| 18 | 24 | S2KDU4 | 4 | Binary<br>0<br>1 | SYSTEM 2000 Multi-User indicator; set at run time by the expanded START S2K code.<br>Single-user<br>Multi-User |
| 1C | 28 | FILLER | 4 | Alpha-numeric | If S2KDU4 is neither fullword binary zero or one, then S2KDU4 concatenated with this field contains the name of a special module to be used (left-justified and blank-filled). |
| 20 | 32 | FILLER | 8 | Alpha-numeric | Name of the program (left-justified and blank-filled); set by the PLEX processor as specified in the $NAME directive. |
| 28 | 40 | FILLER | 4 | Alpha-numeric | Release of the SYSTEM 2000 PLEX processor used; set by the PLEX processor. If changes are made to the Program Service Processor, this field identifies the PLEX processor release that is used. |
| 2C | 44 | FILLER | 8 | Alpha-numeric | Date of the PLEX processor run (YYDDD, left-justified and blank-filled); set by the PLEX processor. |
| 34 | 52 | FILLER | 8 | Alpha-numeric | Time of PLEX processor run (HHMMSSS, left-justified and blank-filled); set by the PLEX processor. |
| 3C | 60 | FILLER | 4 | Alpha-numeric | SYSTEM 2000 release number; set at execution time by SYSTEM 2000 software. |

*continued on next page*

**Illus. B.11** *continued*

| Offset HEX | Offset DEC | Name | Length/ Alignment | Type/ Value | Field Description |
|---|---|---|---|---|---|
| 40 | 64 | S2KDU5 | 4 | Binary | Reserved for internal use. |
| 44 | 68 | S2KDU6 | 4 | Binary | Reserved for internal use. |
| 48 | 72 | S2KDU7 | 4 | Alpha | Reserved for internal use. |
| 4C | 76 | FILLER | 4 | Binary | Address of run-time interface (S2KPLR). |
| 50 | 80 | FILLER | 4 | Binary | Address of SYSTEM 2000 software (S2KPLI, MUPLINT). |
| 54 | 84 | FILLER | 4 | Binary | Address of working storage; set by run-time interface. This address may be used by installation run-time exit routines S2KEX1 or S2KEX2. |
| 58 | 88 | S2KDU8 | 4 | Binary<br>0<br>1 | Snap request indicator; set by expanded code.<br>Off<br>On: SYSTEM 2000 software will provide snapshot data. |
| 5 | 92 | S2KDU9 | 4 | Binary<br>0<br>1 | Timing request indicator; set by the expanded code.<br>Off<br>On |
| 60 | 96 | S2KSRT | 4 | Binary | No-sort request indicator; set by the PLEX program.<br>0 = Sort normally.<br>1 = Do not sort. |
| 64 | 100 | | 4 | Binary | Reserved for internal use. |
| 68 | 104 | | 4 | Binary | Reserved for internal use. |
| 6C | 108 | | 1 | Bit<br>.... ...0<br>.... ...1 | Switches used at run time.<br>Link history was not modified.<br>Link history was modified. |
| 6D | 109 | | 3 | | Reserved for internal use. |
| 70 | 112 | S2KLKH | 40 | Binary | Ten full words representing LINK0 thru LINK9 return codes; set by linked retrievals. Initialized by the PLEX processor to ten words of binary minus one. |

## $CICS ON Directive

### Format

```
$CICS  |ON    ;
       |OFF   (default)
```

**Command level** If your program contains command-level CICS code and uses the $CICS ON directive, you must issue the $CICS OFF directive immediately after $CICS ON. This causes the PLEX processor to generate appropriate calls to PLXPBLD.

## $COPY Directive

### Format

```
$COPY   member ;
```

*member* is the name of a member of the SYSLIB copy library.

The $COPY directive instructs the PLEX processor to copy source statements from a member of a prepared library into the PLEX program where it appears. That is, the $COPY command specifies the member of the SYSLIB that provides the statements, and the placement of $COPY indicates where the copied statements go.

You must specify the COPY execution option (see **COPY and NOCOPY Execution Options** on page B-71) in the JCL for a job that contains a program that issues the $COPY directive. Only entire commands, COMMBLOCKs, or SUBSCHEMA RECORD declarations can be copied. For example, you cannot copy a where-clause or a portion of a subschema record.

The PLEX processor supports up to ten levels of $COPY nesting. If you include a $COPY directive in a member that is copied, the PLEX processor obtains the additional commands from the indicated member. If you concatenate $COPY files, the first must have the largest block size.

Special Zap #124 is available to require that all subschema record declarations be copied from the $COPY library. Contact your SYSTEM 2000 Software Representative.

## $NAME Directive

### Format

```
$NAME  name  ;
```

*name* is an eight-character name, left-justified and blank-filled.

The $NAME directive instructs the PLEX processor to initialize S2K3, a variable in the S2KDUM control block, to the specified name. Also, this name is printed at the top of each page of output from the PLEX processor.

When Multi-User accounting is activated, a PL/I program running in a Multi-User environment should include the $NAME directive so that the program name can be included in accounting output.

## *PROCESS Statement

The PLEX processor reinitializes its internal settings when it encounters the PL/I statement *PROCESS. Two external procedures that are to be separately processed by the PLEX processor must be separated by a *PROCESS statement. If external procedures are not separated by a *PROCESS statement, the PLEX processor treats them as one unit. In this case, the PLEX processor issues error messages if the $BLOCKS directive appears twice, if two COMMBLOCKs or subschema records have the same name, or if a command refers to an unqualified subschema item name that is found in more than one subschema record. However, if a *PROCESS statement does separate the external procedures, no errors regarding duplicates occur. Note that *PROCESS must start in column 1 of the statement.

## $STORAGE Directive

### Format

```
$STORAGE  |[STATIC    ]   [S2KDUM |PARM     ][FOR |STATIC    ]
          |[AUTOMATIC             |EXTERNAL       |AUTOMATIC
                                  |BASED

          [ACTION |STATIC    ]  [WHERE |STATIC    ]   ;
                  |AUTOMATIC            |AUTOMATIC
```

The $STORAGE directive specifies attributes for certain control variables generated by the PLEX processor. One or more control variables can be specified in the directive:

**S2KDUM** is a control block generated by the PLEX processor in response to the $BLOCKS directive. Values in this control block change during the execution of every PLEX command.

**FOR** generated for each FOR command. The value(s) for this control variable change during the execution of the command.

**ACTION** generated for PLEX commands containing item lists or requiring that special information be passed to SYSTEM 2000 software during execution. The value(s) for this control variable do not change during the execution of the command.

**WHERE** generated for each LOCATE command and each GET1 command with a where-clause. The value(s) for this control variable do not change during the execution of the command.

The PARM option or the EXTERNAL or BASED attribute can be explicitly assigned to S2KDUM. The STATIC or AUTOMATIC attribute can be assigned to S2KDUM and one or more of the control variables.

**PARM** specifies that the external procedure will receive S2KDUM as a parameter. The PLEX processor generates attributes, not initial values, for the variables in S2KDUM.

**EXTERNAL** specifies that S2KDUM is to be placed in an external data area. The PLEX processor generates both attributes and initial values for the S2KDUM structure.

**BASED** specifies that the address of S2KDUM can be found in a pointer variable. This pointer, S2KDUM_A, is specified by the PLEX processor. It is not named in the syntax of the $STORAGE directive. Your program must set this pointer to the appropriate address. For the variables in this based structure, the PLEX processor generates attributes, not initial values.

**STATIC or AUTOMATIC** can be specified immediately following the keyword $STORAGE. In this case, if the directive does not specify S2KDUM, the STATIC or AUTOMATIC attribute is assigned to the structure.

The STATIC or AUTOMATIC attribute can be assigned to the FOR, ACTION, or WHERE control variables by either

- specifying the attribute explicitly with the keyword FOR, ACTION, or WHERE, or
- specifying the attribute immediately following the keyword $STORAGE, in which case the attribute is assigned to the type(s) of control variables (S2KDUM, FOR, ACTION, WHERE) not specified in the directive.

The $STORAGE directive can appear at any point in your program. More than one $STORAGE directive can be specified, in which case only the most recent one is effective.

**Example 1**

This example shows the use of a $STORAGE directive that is equivalent to the default. This could be used in a simple (one external procedure) command-level CICS program.

```
$STORAGE AUTOMATIC;
```

**Example 2**

The following example shows the use of $STORAGE in a subroutine, to allow the subroutine to use the same SYSTEM 2000 control blocks as those used by the calling routine. The expanded source program is shown first, followed by the subroutine source.

**Expanded Source**

```
TEST: PROCEDURE OPTIONS (MAIN);
     DCL 1 FTITLE CHAR(40)
           INIT('HOUND OF THE BASKERVILLES      ');
     DCL 1 LIB CHAR (4) INIT ('LIB ');
     DCL I FIXED BINARY(15) INIT(0);
     DCL (PTR_DUM,PTR_LIB,PTR_BOOK) PTR;
     DCL ADDR BUILTIN;
     DCL SUB1 ENTRY EXTERNAL;
     /*
     $BLOCKS
     */
DCL S2KPL ENTRY OPTIONS(ASM,INTER);
DCL 1 S2KDUM,
      2 S2KDU1     FIXED BIN(31) INIT (0),
      2 S2KRTC     FIXED BIN(31) INIT (0),
      2 S2KCOUNT   FIXED BIN(31) INIT (0),
      2 S2KDU2     FIXED BIN(31) INIT (0),
      2 S2KDU3     FIXED BIN(31) INIT (0),
      2 S2K1       FIXED BIN(31) INIT (3),
      2 S2KDU4     FIXED BIN(31) INIT (0),
      2 S2K2       FIXED BIN(31) INIT (0),
      2 S2K3       CHAR(8)       INIT ('        '),
      2 S2K4       CHAR(4)       INIT ('12.1'),
      2 S2K5       CHAR(8)       INIT ('95117   '),
      2 S2K6       CHAR(8)       INIT ('1602244 '),
      2 S2K7       CHAR(4)       INIT ('    '),
      2 S2KDU5     FIXED BIN(31) INIT (0),
      2 S2KDU6     FIXED BIN(31) INIT (-1),
      2 S2KDU7     CHAR(4)       INIT ('    '),
      2 S2K8       FIXED BIN(31) INIT (0),
      2 S2K9       FIXED BIN(31) INIT (0),
      2 S2K10      FIXED BIN(31) INIT (0),
      2 S2KDU8     FIXED BIN(31) INIT (0),
      2 S2KDU9     FIXED BIN(31) INIT (0),
      2 S2KSRT     FIXED BIN(31) INIT (0),
      2 S2K11      FIXED BIN(31) INIT (0),
      2 S2K12      FIXED BIN(31) INIT (0),
      2 S2K13      FIXED BIN(31) INIT (0),
      2 S2KLKH(10) FIXED BIN(31) INIT ((10)-1);
     /*
     COMMBLOCK OF LIBRARY
     DCL 1  BRANCH_LIBRARY,
     */
     DCL 1 BRANCH_LIBRARY_0,
           2  S2K0 CHAR(16) INIT ('LIBRARY'),
           2  S2K1(3) FIXED BIN(31) INIT (0,120,0),
```

*continued on next page*

**Expanded Source continued**

```
          2 BRANCH_LIBRARY,
          3 SCHEMA_NAME          CHAR(32)       INIT (' '),
          3 RETURN_CODE          FIXED BIN(31) INIT (0),
          3 FLAGS                CHAR(4)        INIT ('A 01'),
          3 FINAL_DATA_SET       FIXED BIN(31) INIT (0),
          3 PASSWORD             CHAR(4)        INIT ('PASS'),
          3 #_OF_DATA_SETS       FIXED BIN(31) INIT (0),
          3     LOCATE                  FIXED BIN(31) INIT (0),
          3 LEVEL                FIXED BIN(31) INIT (0),
          3 CYCLE_TIME           CHAR(8)        INIT (' '),
          3 CYCLE_DATE           CHAR(8)        INIT (' '),
          3 CYCLE_#              FIXED BIN(31) INIT (0),
          3 ASTERISK             CHAR(4)        INIT (' '),
          3 END_ENTRY            CHAR(4)        INIT (' '),
          3 STATUS_FLAG          FIXED BIN(31) INIT (0);
/*                                                                    */
   /*
   SUBSCHEMA BOOK OF LIBRARY
  DECLARE 1 BOOK,
   */
  DCL 1 BOOK_0,
          2  S2K(6),
          3  S2K0 CHAR(32) INIT ('BOOK','TITLE','C2','C3','C004',
          'C005'),
          3  S2K1(6) FIXED BIN(31) INIT (67,(2)0,5,0,134217728,280,
          (4)0,40,264,(4)0,10,218,(4)0,5,167,(4)0,460037,116,(4)0,
          7),
          2 BOOK,
          5 TITLE CHAR (40),
          5 C2 PICTURE '(10)X',
          5 C3 PICTURE 'XXXXX',
          5 C004 FIXED DECIMAL (8,5),
          5 C005 PICTURE 'XXXXXXX';

   DCL SWITCH FIXED BIN (31) INIT (1);
  /*
*PL START S2K(1) USE(SWITCH)
  */
  DO;
  S2KDU1=91;
  S2KDU3=1;
  S2KDU4=SWITCH;
  CALL S2KPL(S2KDUM);
  END;
   PASSWORD = LIB;
   /*
*PL  OPEN LIBRARY
   */
   DO;
   S2KDU1=60;
   CALL S2KPL(S2KDUM,BRANCH_LIBRARY_0);
   END;
```

*continued on next page*

**Expanded Source continued**

```
    /*
*PL LOCATE BOOK WHERE TITLE GT FTITLE
    */
    DO;
    DCL S2R1(2) FIXED BIN(31) INIT(2,531);
    S2KDU1=10;
    S2KDU2=0;
    CALL S2KPL(S2KDUM,BRANCH_LIBRARY_0,BOOK_0,S2R1,BOOK_0.S2K(2),FTITL E
    );
    END;

     PTR_DUM = ADDR(S2KDUM);
     PTR_LIB = ADDR(BRANCH_LIBRARY_0);
     PTR_BOOK= ADDR(BOOK_0);

     CALL SUB1(PTR_DUM,PTR_LIB,PTR_BOOK);
    /*
*PL CLOSE LIBRARY
    */
    DO;
    S2KDU1=97;
    CALL S2KPL(S2KDUM,BRANCH_LIBRARY_0);
    END;
    /*
*PL STOP S2K
    */
    DO;
    S2KDU1=92;
    CALL S2KPL(S2KDUM);
    END;
END TEST;
```

*continued on next page*

**Subroutine Source**

```
SUB1: PROCEDURE (PTR_A,PTR_B,PTR_C);

      DCL (PTR_A,PTR_B,PTR_C) PTR;

      DCL (S2KDUM_A,COMM_PTR,BOOK_PTR) PTR;

      /*
      $STORAGE S2KDUM BASED
      */

      /*
      COMMBLOCK OF LIBRARY

      DCL 1 LIBRARY BASED(COMM_PTR),
      */
      DCL 1 LIBRARY_0 BASED(COMM_PTR),
             2  S2K0 CHAR(16),
             2  S2K1(3) FIXED BIN(31),
             2 LIBRARY,
             3 SCHEMA_NAME          CHAR(32)      ,
             3 RETURN_CODE          FIXED BIN(31),
             3 FLAGS                CHAR(4)       ,
             3 FINAL_DATA_SET       FIXED BIN(31),
             3 PASSWORD             CHAR(4)       ,
             3 #_OF_DATA_SETS       FIXED BIN(31),
             3      LOCATE                 FIXED BIN(31),
             3 LEVEL                FIXED BIN(31),
             3 CYCLE_TIME           CHAR(8)       ,
             3 CYCLE_DATE           CHAR(8)       ,
             3 CYCLE_#              FIXED BIN(31),
             3 ASTERISK             CHAR(4)       ,
             3 END_ENTRY            CHAR(4)       ,
             3 STS                  FIXED BIN(31);

      /*
      SSR BOOK OF LIBRARY
      DCL 1 BOOK BASED(BOOK_PTR),
      */
      DCL 1 BOOK_0 BASED(BOOK_PTR),
      2  S2K(6),
      3  S2K0 CHAR(32),
      3  S2K1(6) FIXED BIN(31),
```

*continued on next page*

**Subroutine Source continued**

```
     2 BOOK,
     5 TITLE CHAR(40),
     5 C2 CHAR(10),
     5 C3 CHAR(5),
     5 C4 FIXED DEC(8,5),
     5 C5 CHAR(7);
     /*
     $BLOCKS
     */
DCL S2KPL ENTRY OPTIONS(ASM,INTER);
DCL 1 S2KDUM BASED(S2KDUM_A),
      2 S2KDU1     FIXED BIN(31),
      2 S2KRTC     FIXED BIN(31),
      2 S2KCOUNT   FIXED BIN(31),
      2 S2KDU2     FIXED BIN(31),
      2 S2KDU3     FIXED BIN(31),
      2 S2K1       FIXED BIN(31),
      2 S2KDU4     FIXED BIN(31),
      2 S2K2       FIXED BIN(31),
      2 S2K3       CHAR(8)      ,
      2 S2K4       CHAR(4)      ,
      2 S2K5       CHAR(8)      ,
      2 S2K6       CHAR(8)      ,
      2 S2K7       CHAR(4)      ,
      2 S2KDU5     FIXED BIN(31),
      2 S2KDU6     FIXED BIN(31),
      2 S2KDU7     CHAR(4)      ,
      2 S2K8       FIXED BIN(31),
      2 S2K9       FIXED BIN(31),
      2 S2K10      FIXED BIN(31),
      2 S2KDU8     FIXED BIN(31),
      2 S2KDU9     FIXED BIN(31),
      2 S2KSRT     FIXED BIN(31),
      2 S2K11      FIXED BIN(31),
      2 S2K12      FIXED BIN(31),
      2 S2K13      FIXED BIN(31),
      2 S2KLKH(10) FIXED BIN(31);
     S2KDUM_A = PTR_A;
     COMM_PTR = PTR_B;
     BOOK_PTR = PTR_C;
      /*
     GET BOOK NEXT
     */
     DO;
     S2KDU1=24;
     S2KDU2=0;
     CALL S2KPL(S2KDUM,LIBRARY_0,BOOK_0,S2KDU6);
     END;
 END SUB1;
```

# PL/I PLEX PROCESSOR EXECUTION OPTIONS

You set the PLEX processor execution options with the PARM parameter on the EXEC statement. For example, if the name of the job step for the PLEX processor is PRECOM, the parameter list for the job step appears as follows:

```
//PRECOM  EXEC..., PARM='parameter-list'
```

Illus. B.15 on page B-91 shows a complete example of JCL for a single-user PLEX program.

## CICS Execution Option

### Format

```
CICS
```

If your PLEX program will be running under CICS, you must specify either the CICS execution option or the $CICS ON directive. If you issue either the CICS execution option or the $CICS ON directive, the PLEX processor generates code that calls the interface PLXPBLD for each command.

## COPY and NOCOPY Execution Options

### Format

```
|COPY
|NOCOPY    (default)
```

If you issue a $COPY directive in your PLEX program (see **$COPY Library** on page B-94), you must specify the COPY execution option. If the COPY execution option is not specified or if the NOCOPY execution option (default) is specified, the PLEX processor issues an error message if it encounters a $COPY directive in your program. The COPY option does not force the use of $COPY.

Special Zap #124 is available to force all subschema record declarations to be copied from the $COPY library. Contact your SYSTEM 2000 Software Representative.

## DECK and NODECK Execution Options

**Format**

```
|DECK
|NODECK    (default)
```

The DECK execution option instructs the PLEX processor to punch the expanded source on SYSPUNCH.

## ECHO and NOECHO Execution Options

**Format**

```
|NOECHO
|ECHO    (default)
```

The ECHO execution option (default) specifies that source PLEX commands and PLEX processor directives should be included as comments in the input (on SYSPUNCH and SYSGO) to the PL/I compiler. NOECHO specifies that source commands and directives should be omitted from the input to the PL/I compiler.

## ERRORS and NOERRORS Execution Options

**Format**

```
|NOERRORS
|ERRORS    (default)
```

The ERRORS execution option (default) causes diagnostic messages to be printed together on SYSPRINT following other output from the PLEX processor. NOERRORS suppresses printing of those diagnostic messages at the end of the PLEX processor output, but not within the body of the source listing. Either option allows other messages to appear at the end of PLEX processor output on SYSPRINT. These other messages give the total number of diagnostics issued for the PLEX program, the highest severity level of those diagnostics, and the memory utilization report.

## ERRSYM Execution Option

**Format**

```
ERRSYM = character
```

*character* is a single alphanumeric or special character. The default is the period.

The ERRSYM execution option specifies the character that fills blank columns preceding the cursor 'C', which is displayed above certain PLEX processor error messages such as message 406: DATA BASE IDENTIFIER NOT FOUND.

## EXIT Execution Option

### Format

```
EXIT = load-module
```

*load-module* is the name of a load module (not link-edited with PLEX processor).

The EXIT execution option names a user-written exit routine for the PLEX processor that edits source input (including statements input by a $COPY directive) before processing by the PLEX processor. The routine must reside in JOBLIB, STEPLIB, or the system link library.

As the PLEX processor receives each input record, it calls the specified exit routine and passes the following three parameters:

- The address of the current source record as it stands before processing by the PLEX processor. You can use this address if you want to add records.
- The address of a fullword value indicating the current level of $COPY nesting. A value of binary zero means that the current record was not generated for a $COPY directive (see **$COPY Directive** on page B-61). A value of binary 1 signifies that a $COPY directive in the original source code caused the record to be read from a member of the copy library. Higher numbers indicate the current value of nested $COPY logic, up to a maximum of 10.
- The address of a fullword value of the action code, which will be set by the exit routine to indicate the disposition of the current record processed by the exit routine. The action code is initialized to zero by the PLEX processor before each call to the exit routine. The exit routine then sets the action code before passing control back to the PLEX processor:

  0 means the PLEX processor should accept the current record as passed back (and possibly modified) by the exit routine.

  1 means the PLEX processor should delete the current record.

  2 means the PLEX processor should accept the current record as passed back (and possibly modified) by the exit routine and should allow another record to be added on the next call to the exit routine.

  >2 is the address of the diagnostic message that the PLEX processor will write to output after the current record.

**Modifying, replacing, and adding records** The exit routine can modify or replace the current record in the source input by receiving it in a call from the PLEX processor, making the appropriate substitution in the field containing the received record, then sending it back to the PLEX processor. If the action code from the exit routine is not 1 (delete record), the PLEX processor accepts what it receives as the current record.

If the action code is 2, the PLEX processor accepts and processes the current record from the exit routine, then immediately makes another call to the exit routine, without taking another record from the source input. The exit routine then supplies a record to be added to the PLEX program, sets the action code, and returns control to the PLEX processor. If the exit routine sets the action code to 2 again this time, the PLEX processor accepts the new record and again calls the exit routine. In this way, you can add one or more records to the PLEX program with multiple calls to the exit routine.

To discontinue this sequence for adding a group of records, the exit routine must set the action code to 0 before it passes the last new record. Action code 0 thus instructs the PLEX processor to accept the last new record and to read another record from the source input before calling the exit routine again.

**Diagnostic messages** The format for the standard PLEX processor diagnostics is also used for printing diagnostic messages issued by an exit routine:

| **Column** | **Column** | **Column** |
|---|---|---|
| **1** | **10** | **17** |
| **PLX 5** *digits* | **PLX 3** *digits* | *message-text* |
| **statement sequence number** | **message number** | |

The PLEX processor sets the statement sequence number automatically. The exit routine determines the message number and message text and passes both to the PLEX processor for printing.

The message text must be in two forms: the long form, occupying 16 words, followed by the short form, occupying five words. The PLEX processor prints the long form of the message when the NOTERM (default) execution option is in effect, and the short form when the TERM execution option was specified (see **TERM and NOTERM Execution Options** on page B-80).

In addition, the exit routine must determine both the severity level of the diagnostic and the column in which the cursor for the diagnostic message should appear.

The field containing the specifications discussed above for a diagnostic message must be arranged as follows:

| Word(s) | Contents |
|---|---|
| 1 | is the fullword value of the column number in which the error in the current record was detected; determines positioning of the PLEX processor error cursor, which is printed on a line between the current record and the diagnostic message text. If the column number is less than 4, the cursor is not printed. |
| 2 | is the fullword value of the severity level of the error. |
| 3 | is a message number of up to four digits in EBCDIC character representation, left-justified and blank-filled. If the message number is four characters long, no space appears between the message number and the message text when the diagnostic message is displayed. |
| 4-19 | is the long-form message text (64 characters); used if the NOTERM (default) execution option is in effect. |
| 20-24 | is the short-form message text (20 characters); used if the TERM execution option is in effect. |

**Example**

**Notes**

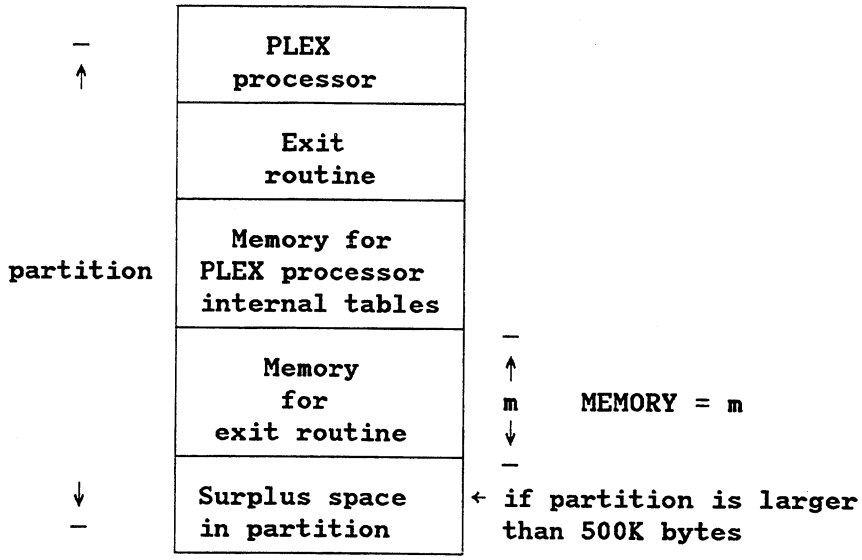
- As an alternative to using a customized exit routine through the EXIT execution option, a standard exit routine named S2KEXIT and developed by systems personnel at the site can be link-edited with the PLEX processor for editing source input from all jobs using the PLEX processor. This exit must have the PROGRAM-ID S2KEXIT (or the CSECT name S2KEXIT if written in Assembler) and must be included as object input during a re-linking of the PLEX processor.

  The PLEX processor automatically checks whether an exit routine named S2KEXIT was link-edited with the PLEX processor. If this routine was link-edited with the PLEX processor, the EXIT execution option should not be specified without approval from the systems personnel at the site, because it overrides use of the standard routine.

- An exit routine can issue up to five diagnostic messages. If more than five messages are issued, PLEX processor diagnostic message 100 is issued.

- An exit routine cannot issue a message for a comment card or a continuation card in the source to the PLEX processor. An attempt to do so causes unpredictable results.

- If the exit routine obtains additional memory using the GETMAIN macro, the amount to be obtained must be specified previously with the MEMORY execution option (see **MEMORY Execution Option** on page B-79). With a GETMAIN, the PLEX processor obtains all available space in its partition (up to 500K bytes). Then it reserves the specified memory space for the exit routine and calculates the amount of memory remaining for its internal tables. The resulting arrangement is shown below.




Sufficient memory must be available in the partition for PLEX processor internal tables for COMMBLOCKs, subschema records, and subschema items described in your PLEX program. The amount of memory required for each of these internal tables is as follows:

**COMMBLOCK table**

**24 bytes for the table control block**
**plus**
**72 bytes per entry (one entry for each COMMBLOCK in the program)**

**Subschema record table**

**24 bytes for the table control block**
**plus**
**52 bytes per entry (one entry for each subschema record in the program)**

**Subschema item table**

**24 bytes for the table control block**
**plus**
**80 bytes per entry (one entry for each subschema item in the program)**

## FORM and NOFORM Execution Options

**Format**

```
|FORM
|NOFORM    (default)
```

If you specify the FORM execution option, the PLEX processor recognizes PLEX commands and directives only if they begin with the characters *PL in columns 2-4 by default or in other columns as set through the SORMGIN execution option (see **SORMGIN Execution Option** on page B-79).

If you use the NOFORM execution option (default), the PLEX processor examines the first word of each line. If it is not a PLEX keyword, the line is assumed to contain only PL/I statements. If the first word is a PLEX keyword, the PLEX processor looks for a PLEX command in that line. With the NOFORM option, a PLEX command can still begin with *PL in columns 2-4, but as just stated, *PL is not required. The NOFORM format is the most commonly used format. **Format of PLEX Commands** on page B-3 discusses the format of PLEX commands under both execution options, as well as restrictions on keywords that can be used in your program.

FORM is primarily intended to provide upward compatibility with earlier SYSTEM 2000 software releases. The use of FORM is not encouraged for new programming efforts.

## GEN and NOGEN Execution Options

**Format**

```
|GEN
|NOGEN    (default)
```

The GEN execution option specifies the printing of the expanded source listing from the PLEX processor on SYSPRINT. NOGEN (default) suppresses the expanded source listing, except that it does not suppress the printing of the expanded COMMBLOCKs in source listings. NOGEN is the default.

## ITEM Execution Option

### Format

```
ITEM = nnnn
```

*nnnn* is the average number of subschema items per subschema record. The default is 5.

The ITEM execution option directs the memory allocation for processing subschema record declarations. The ITEM and SSR options together are used for optimizing memory utilization during the execution of the PLEX processor. See **Memory Utilization during the PLEX Processor Step** on page 22-16 for more information on the use of this execution option in memory utilization.

## LINECNT Execution Option

### Format

```
LINECNT nn
```

*nn* is the number of lines per page. The default is 55.

The LINECNT execution option sets the number of lines per page for SYSPRINT output.

## LOAD and NOLOAD Execution Options

### Format

```
|NOLOAD
|LOAD   (default)
```

The LOAD option (default) specifies that your PLEX program, as expanded by the PLEX processor, will be sent to SYSGO for the compilation job step. NOLOAD specifies that no further processing of your PLEX program will be performed.

## MEMORY Execution Option

### Format

```
MEMORY = bytes
```

*bytes* is the number of bytes for main memory, up to four digits. The default is 4096.

The MEMORY execution option specifies the amount of main memory that should remain available in the region during the PLEX processor job step. If you also specify an exit routine (with the EXIT execution option) that dynamically obtains additional memory, you must give an appropriate value to MEMORY (see **EXIT Execution Option** on page B-73).

## SORMGIN Execution Option

### Format

```
SORMGIN = (a,b,c,d)
```

*a* is not applicable to PL/I.

*b* is not applicable to PL/I.

*c* is the starting column for the scan by the PLEX processor.

*d* is the sequence number starting position.

The SORMGIN execution option designates the margin positions on the source statements. The defaults for PL/I PLEX are as follows:

*c* = 2

*d* = 73

## SOURCE and NOSOURCE Execution Options

### Format

```
|NOSOURCE
|SOURCE   (default)
```

The SOURCE execution option (default) causes the source input, including source PL/I statements, to be listed on SYSPRINT. NOSOURCE suppresses the source listing. NOSOURCE implies NOGEN and TERM, which gives short-form messages.

## SSR Execution Option

### Format

```
SSR = nnnn
```

*nnnn* is the average number of subschema records per COMMBLOCK. The default is 4.

The SSR execution option directs memory allocation for processing subschema record descriptions. This option optimizes memory utilization during the execution of the PLEX processor. See **Memory Utilization during the PLEX Processor Step** on page 22-16 for more information about the use of this execution option in memory utilization.

## SYSSEP Execution Option

### Format

```
SYSSEP = a
```

*a* is a single alphanumeric or special character. The default is the asterisk.

The SYSSEP execution option designates the system separator character for the ternary operators SPAN, SPANS, SPANNING, EQ, and NE, and for item-to-item comparisons in where-clause processing.

## TERM and NOTERM Execution Options

### Format

```
|TERM
|NOTERM    (default)
```

The TERM execution option directs the PLEX processor to issue short-form messages, which are suitable for jobs running in TP environments. The NOTERM option (default) specifies the long form for PLEX processor messages. See **Diagnostic messages** on page B-74 for the short form and long form format of PLEX processor messages.

## TRANS Execution Option

### Format

```
TRANS = load-module
```

*load-module* is the name of a load module for a translate table routine (not link-edited with PLEX processor).

The TRANS execution option designates a table routine that translates all source input. This table is commonly used for translation of character sets, for example, EBCDIC to ASCII. Refer to the IBM *Principles of Operation* manual for a description of the TR table. The routine must be included in JOBLIB, STEPLIB, or the system link library.

An exit routine named S2KTRANS and developed by systems personnel at the site can be link-edited with the PLEX processor for translating source input from all jobs using the PLEX processor. The PLEX processor automatically checks whether an exit routine by the name of S2KTRANS was link-edited with the PLEX processor. If it was, do not specify the TRANS execution option without approval from the systems personnel at the site, because it overrides use of the standard routine.

# SAMPLE PL/I PLEX PROGRAM

A generalized, skeleton PL/I PLEX program is shown in Illus. B.12. PL/I statements can appear anywhere in a program. However, executable PLEX commands can appear only after the START S2K command.

The rest of this section shows a sample program that uses the EMPLOYEE database to produce a listing of employees by department managers.

**Illus. B.12** Skeleton of a PLEX Program

```
COMMBLOCK(s)                          (One COMMBLOCK must be included
                                      for each database to be accessed.)
      :
      :
SSR(s)                                (One or more subschema records can
                                      be included to refer to one or more
                                      schema records.  Each SSR
                                      refers to a one schema record.)
      :
      :
$BLOCKS.                              (The program must have one $BLOCKS
                                      directive after the COMMBLOCK and
                                      SUBSCHEMA declarations.)
      :
      :
START S2K.                            (First PLEX command executed by the
                                      program)
      :
      :
Initialize password(s) in
COMMBLOCK(s) and open database(s)
      :
      :
PLEX command(s)
      :
      :
CLOSE database(s)
      :
      :
PLEX command(s)
      :
      :
CLOSE remaining database(s)
      :
      :
STOP S2K                              (Last PLEX command executed by the
                                      program)
```

## FINDORG Program

The sample program FINDORG uses PLEX retrieval commands and several stacks and Locate Files to produce a roster of all personnel in the company. In this roster, employees are listed under their managers.

The program has one subschema record for level 0 data records. It also has two subschema records that refer to schema record C100, (POSITION WITHIN COMPANY). The C100 subschema record POSITION is used to retrieve the title of an employee's position. The other C100 subschema record, COMPARISON, contains two items used for comparing values in the LOCATE command's where-clause.

The program retrieves employee records according to the hierarchy of the organization. It finds the organizational hierarchy by extracting the initials from each manager's name and looking for those initials in the MANAGER item of the most current POSITION WITHIN COMPANY data record in each logical entry.

A GET1 command retrieves the president's level 0 data record. Then a LOCATE command establishes a Locate File pointing to all employees who report directly to him. That is, Locate File 1 points to the level 0 data records of all employees who report directly to the president.

Next a GET command retrieves a level 0 record for an employee whose address is in this Locate File. The program extracts the initials from the employee name in this record. Then a LOCATE command finds the level 0 records of all personnel who report directly to this employee, and so on, down the company's chain of command.

When an employee who has no subordinates is retrieved (that is, the Locate File is empty), the program logic reverts to the Locate File pointing to this employee's level 0 record. A GET command retrieves the next employee, if there is one, who works for the same manager.

In general, when the program receives return code 04 at the end of one Locate File, it issues a GET command for the previously established Locate File (whose number is one less than that of the latest Locate File). Thus, the program proceeds as far down in the organizational hierarchy as it can before moving across the hierarchy. Eventually, the end of Locate File 1 is reached, which means that all subordinate employees were retrieved.

As the program retrieves each employee's record,

- it extracts the initials from the employee's forename and the last name for use in the next LOCATE command
- it concatenates the forename, last name, and job title into one line for printing by the main program.

Each output line is indented under the line for the employee's immediate manager.

See Illus. B.13 on page B-84 for the FINDORG source statements. Illus. B.14 on page B-89 shows the output from this program.

**Illus. B.13** Source Statements for the FINDORG Program

```
FINDORG:PROCEDURE OPTIONS (MAIN);
 /*********************************************************************/
 /*ABSTRACT:                                                          */
 /*            THIS PROGRAM USES EMPLOYEE DATABASE TO FIND THE        */
 /*            ORGANIZATION OF THE COMPANY.  IT USES MULTIPLE LOCATE  */
 /*            FILES AND STACKS.  IT FINDS THE PRESIDENT, THEN ALL    */
 /*            PEOPLE WHO WORK FOR THE PRESIDENT, THEN ALL THEIR      */
 /*            SUBORDINATES, AND SO ON.  THE NAME AND TITLE OF EACH   */
 /*            EMPLOYEE IS PRINTED.  THE NAME OF EACH EMPLOYEE        */
 /*            APPEARS UNDER HIS SUPERVISOR AND IS INDENTED TO        */
 /*            SHOW THE LEVEL.  IT IS ASSUMED THAT EACH EMPLOYEE      */
 /*            HAS A UNIQUE COMBINATION OF INITIALS.                  */
 /*                                                                   */
 /*NARRATIVE:                                                         */
 /*STEP                              ACTION                           */
 /*  1.   INITIALIZE.                                                 */
 /*  2.   FIND THE PRESIDENT OF THE COMPANY. PRINT THE NAME AND TITLE*/
 /*       AND FIND THE INITIALS.                                      */
 /*  3.LOCATESUB:                                                     */
 /*       LOCATE ALL SUBORDINATES OF THE PERSON WHOSE INITIALS HAVE   */
 /*       JUST BEEN FOUND.                                            */
 /*       IF NONE, CHECK IF NEEDS TO FIND SUBORDINATES OF ANY OTHER   */
 /*       PERSON. IF SO, GO TO STEP4. OTHERWISE, GO TO STEP 5.        */
 /*  4.GETNEXT:                                                       */
 /*       GET THE NEXT SUBORDINATE. IF THERE IS NO SUBORDINATE LEFT   */
 /*       AT THIS LEVEL, THEN REPEAT THIS STEP TO FIND THE NEXT       */
 /*       SUBORDINATE FROM THE PREVIOUS LOCATE. IF THERE IS NO PREVI-*/
 /*       OUS LOCATE THEN GO TO STEP 5. OTHERWISE, FIND THE NAME AND  */
 /*       TITLE OF THIS SUBORDINATE AND PRINT THEM. FIND THE          */
 /*       INITIALS AND GO TO STEP 3 TO LOCATE THAT PERSON'S           */
 /*       SUBORDINATES.                                               */
 /*  5.   TERMINATE THE JOB.                                          */
 /*                                                                   */
 /*********************************************************************/
```

*continued on next page*

**Illus. B.13** *continued*

```
COMMBLOCK OF EMPLOYEE ;
   DCL 1 EMPLOYEE ,
       3 RECORD_NAME,
       3 RETURN_CODE,
       3 FREE ,
       3 LAST_RECORD_FLAG,
       3 PASSWORD,
       3 NO_OF_RECORDS,
       3 LF_POINTER,
       3 LEVEL,
       3 CTIME,
       3 CDATE,
       3 CYCLE,
       3 SEPARATOR,
       3 TERMINATOR,
       3 DAMAGE ;
SUBSCHEMA RECORD ENTRY OF EMPLOYEE ;
   DCL 1 ENTRY ,
       3 LASTNAME CHAR(16) VARYING ,
       EQ C2 ,
       3 FORENAME CHAR(16) VARYING ,
       ;
SUBSCHEMA RECORD POSITION OF EMPLOYEE ;
   DCL 1 POSITION ,
       3 TITLE CHAR (48)  ,
       EQ C101 ,
       ;
SUBSCHEMA RECORD COMPARISON OF EMPLOYEE ;
   DCL 1 COMPARISON ,
       3 MANAGER CHAR (3) ,
       3 END_DATE CHAR (6),
       EQ C106 ,
       ;
                                  /**/
$BLOCKS;

                                  /**/
```

*continued on next page*

**Illus. B.13** *continued*

```
DCL LSNUM BIN FIXED (15);     /* THE SUBSCRIPT OF THE ACTIVE LOCATE  */
                              /* FILE AND STACK                      */
DCL SHORT CHAR(4);            /* INITIALS OF AN EMPLOYEE             */
DCL PRESIDENT CHAR(9) INIT ('PRESIDENT'); /* TITLE OF THE PRESIDENT */
DCL IFLAG CHAR(1) INIT ('C');/* INCOMPLETE FLAG,I=INCOMPLETE,C=COMPL*/
                              /**/
DISPLAYCOMM: PROC(COMMAND); /* DISPLAY COMMBLOCK */
        DCL COMMAND CHAR(20);
        PUT SKIP (3);
        PUT LIST ('COMMBLOCK AFTER', COMMAND);
        PUT SKIP (2);
        PUT LIST (EMPLOYEE);
        PUT SKIP (1);
        END DISPLAYCOMM;
                                 /**/
INITIAL : PROC (LNAME,FNAME) RETURNS (CHAR(4));
        /*THIS FUNCTION EXTRACTS THE INITIALS OF AN EMPLOYEE FROM  */
        /*HIS NAME. THE INPUT PARAMETERS ARE LAST NAME AND FORENAME*/
        /*RESPECTIVELY. THE FORENAME INCLUDES BOTH THE FIRST NAME  */
        /*AND A MIDDLE INITIAL, IF THERE IS ONE.                   */
DCL LNAME CHAR(16) VARYING ;      /* LAST NAME */
DCL FNAME CHAR(16) VARYING ;      /* FORENAME  */
DCL FULLNAME  CHAR(32) VARYING;
DCL ENDCHAR  FIXED BIN (15);
DCL INITS CHAR(4);                /* INITIALS  */
DCL I FIXED BIN (15) ;            /* LOOP INDEX*/
DCL FLAG BIN FIXED (15);          /* POSITION OF INITIAL FOUND      */
        /*   THE ROUTINE:                                          */
    INITS = ' ';
    FULLNAME = FNAME||' '||LNAME;
    SUBSTR(INITS,1,1) = SUBSTR(FNAME,1,1) ; /*FIRST INITIAL FOUND  */
    FLAG = 1;
    ENDCHAR = LENGTH(FULLNAME) - 1;
    DO I = 2 TO ENDCHAR;
       IF SUBSTR(FULLNAME,I,1) = ' '
       THEN
            IF SUBSTR(FULLNAME,I+1,1) = ' '
            THEN DO;
                    FLAG = FLAG + 1;
                    SUBSTR(INITS,FLAG,1) = SUBSTR(FULLNAME,I+1,1);
                 END;
    END;
    RETURN (INITS);
    END INITIAL;
```

*continued on next page*

**Illus. B.13** *continued*

```
/*********************************************************************/
/*                                                                   */
/*             TOP OF MODULE                                         */
/*                                                                   */
/*********************************************************************/
STEP1:   /*INITIALIZE , ASK FOR 15 ADDITIONAL LOCATE FILES  */
    PASSWORD = 'DEMO' ;
    START S2K (15);
    IF S2KRTC = 0 THEN DO;
        PUT SKIP LIST ('S2KRTC = 0 ON START S2K');
        STOP S2K;
        EXIT;
        END;
    OPEN EMPLOYEE ;
        IF RETURN_CODE = 0 THEN
           DO;
             CALL DISPLAYCOMM('OPEN');
             GOTO EXITSTEP;
           END;
                                  /**/
STEP2:   /* FIND THE PRESIDENT  */
    GET1 ENTRY WHERE TITLE EQ PRESIDENT AND END_DATE FAILS;
        IF RETURN_CODE = 0 THEN
           DO;
             CALL DISPLAYCOMM('GET1');
             GOTO EXITSTEP;
           END;
    SHORT = INITIAL  (LASTNAME, FORENAME) ;
    PUT EDIT ( FORENAME ||' ' || LASTNAME ||', ' ||'PRESIDENT')(A) ;
    LSNUM = 1 ;
                                  /**/
STEP3:
LOCATESUB:                          /* LOCATE ALL SUBORDINATES       */
    LOCATE (LSNUM) ENTRY WHERE MANAGER EQ SHORT AND END_DATE FAILS ;
        IF RETURN_CODE = 0 THEN
           DO;
             CALL DISPLAYCOMM('LOCATE');
             GOTO EXITSTEP;
           END;
    IF NO_OF_RECORDS = 0            /* NO SUBORDINATES IN THIS LEVEL*/
    THEN DO;
         IF LSNUM = 1       /* NO SUBORDINATES OF THE PRESIDENT */
         THEN GOTO STEP5 ;          /* ORGANIZATION CHART COMPLETED */
         ELSE LSNUM =LSNUM - 1 ;    /* GET SUBORDINATES IN THE      */
         END ;                      /* PREVIOUS LOCATE              */

                                  /**/
```

*continued on next page*

**Illus. B.13** *continued*

```
   STEP4:
   GETNEXT: /* GET NEXT SUBORDINATE IN THIS LEVEL */
      GET (LSNUM) ENTRY NEXT ;
      IF RETURN_CODE = 4            /* ALL SUBORDINATES FOR THIS LEVEL*/
      THEN DO;                      /* DONE. IF THIS IS LEVEL 1, THE  */
           LSNUM = LSNUM - 1 ;      /* ORGANIZATION CHART IS COMPLETE.*/
           IF LSNUM = 0             /* OTHERWISE, GO BACK TO LEFT OVER*/
           THEN GOTO STEP5 ;        /* FROM PREVIOUS LOCATE           */
           ELSE GOTO GETNEXT ;
           END;
      ELSE IF RETURN_CODE = 0 THEN
             DO;
               CALL DISPLAYCOMM('GET');
               GOTO EXITSTEP;
             END;
   GETD(LSNUM) POSITION NEXT ;
      IF RETURN_CODE= 0 THEN
         DO;
           CALL DISPLAYCOMM('GETD');
           GOTO EXITSTEP;
         END;
      SHORT = INITIAL  (LASTNAME, FORENAME) ;
      PUT EDIT (FORENAME||' '||LASTNAME||', '||TITLE)
               (COLUMN(LSNUM*2),A);
      IF LSNUM  = 15 THEN DO    ;  /* ONLY SIXTEEN LEVELS CAN BE      */
                    IFLAG = 'I';  /* REPORTED                        */
                    GOTO GETNEXT;
                    END;
      LSNUM = LSNUM + 1 ;
      GOTO LOCATESUB ;
                                  /**/
   STEP5:
      PUT SKIP(2) ;
      PUT LIST ('ORGANIZATION CHART COMPLETED.  ');
      IF IFLAG = 'I'
      THEN PUT LIST ('ONLY UPTO 16 LEVELS ARE SHOWN.');
  EXITSTEP: /* CLOSE DATABASE AND STOP */
          CLOSE EMPLOYEE;
      PUT PAGE;
          CALL DISPLAYCOMM('CLOSE');
          STOP S2K;
      END FINDORG ;
```

**Illus. B.14** Output from the FINDORG Program

```
CLIFTON P. WATERHOUSE, PRESIDENT
  YOLANDA SALAZAR, ADMINISTRATIVE ASSISTANT
  HUGH E. BOWMAN, EXECUTIVE VICE-PRESIDENT
    DARRYL NATHANIEL, SECRETARY
    OLAN M. GARRETT, VICE-PRESIDENT MARKETING
      ALTHEA KNIGHT, SECRETARY
      VIRGINA P. BROWN, MANAGER WESTERN REGION MARKETING
        FANNIE LITTLEJOHN, SECRETARY
        MADISON A. SCHOLL, SR SALES REPRESENTATIVE
          RUBEN R. BROOKS, JR SALES REPRESENTATIVE
        ARMANDO JUAREZ, SR SALES REPRESENTATIVE
        DAVID AMEER, SR SALES REPRESENTATIVE
      TRAVIS Z. RICHARDSON, MANAGER EASTERN REGION MARKETING
        MERRILEE D. WAGGONNER, SECRETARY
        ALAN F. GOODSON, SR SALES REPRESENTATIVE
          JERRY LEE SMITH, JR SALES REPRESENTATIVE
        LELAND G. SHROPSHIRE, JR SALES REPRESENTATIVE
        TAI CHAN, SR SALES REPRESENTATIVE
      GWENDOLYN VAN HOTTEN, MANAGER PUBLIC RELATIONS & EDUCATION
        ROMUALDO R. RODRIGUEZ, PR & ADVERTISING SPECALIST
        JANICE L. WILLIAMSON, INSTRUCTOR
        GEORGE J. GIBSON, INSTRUCTOR
    JOEL B. MILLSAP, VICE-PRESIDENT INFORMATION SYSTEMS
      CARRIE ANN FAULKNER, SECRETARY
      JACOB CAHILL, MANAGER SYSTEMS DEVELOPMENT & MAINTENANCE
        MOLLY I. GIBSON, TECHNICAL WRITER
        MICHAEL Y. JONES, SR SYSTEMS ANALYST
          DAVID G. REID, PROGRAMMER
           , PROGRAMMER
          FRANCISCO GARCIA, JR PROGRAMMER/ANALYST
        RICHARD B. REDFOX, SYSTEMS ANALYST
      JESSE L. HERNANDEZ, MANAGER DATA BASE ADMINISTRATION
        LEOPOLD FREEMAN, SR SYSTEMS PROGRAMMER
        SOPHIA FERNANDEZ, STANDARDS & PROCEDURES ANALYST
      IVAN L. POLANSKI, MANAGER SYSTEMS OPERATIONS & RESOURCES
        JANET F. SMITH, SR SYSTEMS PROGRAMMER
          BRADFORD JOHNSON, JR SYSTEMS PROGRAMMER
        PEDRO QUINTERO, OPERATIONS SUPERVISOR
          WILLIAM D. SAVAGE, COMPUTER OPERATOR
          GARY SEATON, COMPUTER OPERATOR
          KENNETH D. REED, COMPUTER LIBRARIAN
    PATRICE R. KNAPP, VICE-PRESIDENT ADMINISTRATION & FINANCE
      FRANK A. CANADY, MANAGER PERSONNEL
        PENNY SCHMIDT, SECRETARY
        STEWART Q. THROCKMORTON, OFFICE SUPERVISOR
          LEONARD R. SLYE, GENERAL MAINTENANCE
          LILLIAN COLLINS, MAIL CLERK
          FREDDIE KAATZ, SUPPLY CLERK
      PATSY MUELLER, SECRETARY
      RITA M. JONES, MANAGER ACCOUNTING
        GARLAND P. SMITH, BOOKKEEPER

ORGANIZATION CHART COMPLETED.
```

# JOB CONTROL LANGUAGE (JCL)

A PLEX job has four distinct steps:

- preprocessing with the PLEX processor
- compiling
- link-editing
- executing.

This section discusses sample JCL for these job steps for both single-user and Multi-User environments. See **PL/I PLEX Processor Execution Options** on page B-70 for execution paramaters that you can use with this JCL.

## Single-User JCL for All Job Steps

In the START S2K command, you can specify that your program will be run in a single-user environment. When the subscript following the keyword USE equals zero or when USE and the subscript are not included in the command, it signifies a single-user environment. (See **The START S2K Command** on page B-43 for information about the START S2K command.)

The execution step requires scratch pads and sort files. These files are dynamically allocated and do not need to be specified in your JCL. See the *SYSTEM 2000 Product Support Manual, Version 12, First Edition* for details on changing the default sizes of pads and sort files.

Illus. B.15 on page B-91 shows an example of the JCL required for all four steps in a single-user environment.

## Multi-User JCL for All Job Steps

The single-user JCL can also be used for a Multi-User environment with the following modifications:

- Delete all DD statements for SYSTEM 2000 work files and all database files from the GO step. (These files are set up when Multi-User is initialized and are shared by all users.)
- Reduce the REGION parameter in the GO statement to 128K.

In the START S2K command, you can specify that your program will be run in a Multi-User environment by setting the subscript following the keyword USE equal to 1. (See **The START S2K Command** on page B-43 for information about the START S2K command.)

**Illus. B.15** Sample JCL for a Single-User PL/I PLEX Program

```
//* -------------- MODIFY JOB CARD AS NEEDED -----------------
//*
//* -----  CHANGE SYSIN TO PL/1 SOURCE PROGRAM  --------------
//PRECOM   EXEC PGM=PRCOMP
//STEPLIB  DD   DSN=S2K.R121.LOAD,DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   DUMMY
//SYSIN    DD   DISP=SHR,DSN=S2K.R121.SOURCE(FINDORG)
//SYSGO    DD  DSN=&&PSOURCE,DISP=(MOD,PASS),UNIT=SYSDA,
//              DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//              SPACE=(CYL,(1,1))
//SYSUT1   DD   UNIT=SYSDA,SPACE=(TRK,(20,20))
//SYSUT2   DD   UNIT=SYSDA,SPACE=(TRK,(20,20))
//* ------------------------------------------------------------
//PL1      EXEC PGM=IEL0AA,
//              PARM=(NODECK,XREF,NEST,STMT),
//              COND=(4,LT,PRECOM)
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSN=&&PSOURCE,DISP=(OLD,DELETE)
//SYSLIN   DD   DISP=(NEW,PASS),SPACE=(80,(900,100)),
//              UNIT=SYSDA,DSN=&&PL1OBJ
//SYSUT3   DD   UNIT=SYSDA,SPACE=(80,(250,250)),
//              DCB=BLKSIZE=80,DSN=&&SYSUT3
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=SYSDA,DCB=BLKSIZE=1024,
//              SPACE=(1024,(60,60),,CONTIG)
//* ----  CHANGE SYSLMOD TO YOUR LOAD MODULE NAME  -----------
//LKED    EXEC PGM=IEWL,
//             PARM='MAP,XREF,LIST,LET,SIZE=(164K,24K),AMODE(24)',
//             COND=((4,LT,PRECOM),(4,LT,PL1))
//SYSLIN   DD   DSN=&&PL1OBJ,DISP=(OLD,DELETE)
//SYSLIB   DD   DSN=SYS1.PLIBASE,DISP=SHR
//         DD   DSN=S2K.R121.LOAD,DISP=SHR
//SYSLMOD  DD   DSN=S2K.R121.LOAD(FINDORG),
//              DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   SPACE=(1024,(400,20)),UNIT=SYSDA
```

*continued on next page*

**Illus. B.15** *continued*

```
//* ------------------------------------------------------------
//GO       EXEC PGM=*.LKED.SYSLMOD,
//              PARM=('ISASIZE=(10K)'),
//              COND=((4,LT,PRECOM),(4,LT,PL1),(4,LT,LKED))
//STEPLIB  DD   DSN=S2K.R121.LOAD,DISP=SHR
//S2KPARMS DD   DSN=S2K.R121.CNTL(NLPARM),DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSUDUMP DD   SYSOUT=A
//S2KSNAP  DD   SYSOUT=A,DCB=BLKSIZE=882
//PL1DUMP  DD   SYSOUT=A
//S2KCOMD  DD   DUMMY
//EMPLOYE1 DD   DSN=S2K.R121.EMPLOYE1,DISP=OLD
//EMPLOYE2 DD   DSN=S2K.R121.EMPLOYE2,DISP=OLD
//EMPLOYE3 DD   DSN=S2K.R121.EMPLOYE3,DISP=OLD
//EMPLOYE4 DD   DSN=S2K.R121.EMPLOYE4,DISP=OLD
//EMPLOYE5 DD   DSN=S2K.R121.EMPLOYE5,DISP=OLD
//EMPLOYE6 DD   DSN=S2K.R121.EMPLOYE6,DISP=OLD
//EMPLOYE7 DD   DSN=S2K.R121.EMPLOYE7,DISP=OLD
//EMPLOYE8 DD   DSN=S2K.R121.EMPLOYE8,DISP=OLD
```

## JCL Variations

The JCL given in the previous section is for all four steps. You can omit one or more steps, depending on the nature of the job. For example, the job might consist of only the preprocessing step or only the preprocessing and compilation steps. The PLEX processor execution options described in **PL/I PLEX Processor Execution Options** on page B-70 also provide for variations in JCL.

The following topics discuss optional DD statements that you can use.

**Additional Locate Files (single-user)** One Locate File, Locate File 0, is available to each PLEX program by default. This file is a SYSTEM 2000 work file with the DDname S2KSYS01. However, for a single-user program that does not use scratch pads but does use additional Locate Files, more DD statements must be provided for the Locate Files. The DDname for Locate File 1 is LOCATE00; Locate File 2 is LOCATE01, and so on.

```
//GO.EMPLOYE1 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE1
//GO.EMPLOYE2 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE1
//GO.EMPLOYE3 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE1          database files
//GO.EMPLOYE4 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE1
//GO.EMPLOYE5 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE1
//GO.EMPLOYE6 DD DISP=OLD,DSN=S2K.DOC.EMPLOYE1
//GO.LOCATE00 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 1
//GO.LOCATE01 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 2
//GO.LOCATE02 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 3
//GO.LOCATE03 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 4
//GO.LOCATE04 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 5
//GO.LOCATE05 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 6
//GO.LOCATE06 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 7
//GO.LOCATE07 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 8
//GO.LOCATE08 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 9
//GO.LOCATE09 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 10
//GO.LOCATE10 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 11
//GO.LOCATE11 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 12
//GO.LOCATE12 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 13
//GO.LOCATE13 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 14
//GO.LOCATE14 DD UNIT=SYSDA,SPACE=(CYL,(1,1))           Locate File 15
```

**Note**: In a Multi-User environment, DD statements for Locate Files are never included in the JCL for each program.

**Additional database files** The Update Log, Rollback Log, Savefile, and Keepfile, as well as database files, will be allocated dynamically by the software if you do not include DD statements for them. The first seven characters of the DDname are the first seven characters of the database name, with Xs replacing embedded (and trailing) blanks. The suffix for the Savefile is S and for the Keepfile is K. Programs that issue KEEP and APPLY commands always use the Keepfile.

**Note**: In a Multi-User environment, DD statements for database files are never included in the JCL for each program.

## Example

```
//GO.EMPLOYE1 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE1
//GO.EMPLOYE2 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE2
//GO.EMPLOYE3 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE3
//GO.EMPLOYE4 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE4
//GO.EMPLOYE5 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE5
//GO.EMPLOYE6 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE6
//GO.EMPLOYE7 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE7          Update Log
//GO.EMPLOYE8 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE8          Rollback Log
//GO.EMPLOYEK DD DSNAME=DATABASE,UNIT=(TAPE9,,DEFER),   Keepfile
//         DISP=(OLD,PASS),VOL=SER=SAS0002
//GO.EMPLOYES DD DSNAME=DATABASE,UNIT=(TAPE9,,DEFER),   Savefile
//         DISP=(OLD,PASS),VOL=SER=SAS001
```

## $COPY Library

When you use the $COPY directive to copy COMMBLOCK declarations or subschema record declarations from a library, you must specify the library in a SYSLIB DD statement in the preprocessing step.

### Example

```
//PRECOM     EXEC     PGM=PRECOMP,PARM='COPY, . . . ', . . .
   .
   .
   .
//SYSLIB     DD       DSN=S2K.COPYLIB,DISP=SHR   $COPY LIBRARY
   .
   .
   .
//PLI        EXEC . . .
```

Appendix C

# FORTRAN PLEX


*FORMAT OF PLEX COMMANDS C-3*
- *FORM and NOFORM Execution Option C-3*
- *Continuation of a Command C-4*
- *Statement Numbers C-5*
- *Comments C-6*
- *One PLEX Command per Line C-6*
- *Reserved Words C-6*

*SYSTEM 2000 ITEM TYPES AND FORTRAN VARIABLE TYPES C-7*
- *Item Types and FORTRAN Variable Types for Items in Where-Clauses C-8*
  - *Program variables as right operands C-9*
  - *Subschema items as right operands C-9*
  - *Considerations C-10*

*COMMBLOCK DECLARATIONS C-11*
- *Format C-12*
- *COMMBLOCK Fields C-12*
- *Examples C-16*
- *Considerations C-20*

*SUBSCHEMA RECORD DECLARATIONS C-22*
- *Format C-23*
- *Examples C-25*
- *Considerations for Subschema Records C-31*
- *Considerations for Items in Subschema Records C-32*

*FORTRAN PLEX PROCESSOR DIRECTIVES C-34*
- *$BLOCKS Directive C-34*
- *$COPY Directive C-37*
- *END PROCEDURE Directive C-37*
- *$NAME Directive C-38*
- *PROGRAM Directive C-38*

*FORTRAN PLEX PROCESSOR EXECUTION OPTIONS C-39*
- *COPY and NOCOPY Execution Options C-39*
- *DECK and NODECK Execution Options C-39*
- *ECHO and NOECHO Execution Options C-40*
- *ERRORS and NOERRORS Execution Options C-40*
- *ERRSYM Execution Option C-40*
- *EXIT Execution Option C-41*
  - *Modifying, replacing, and adding records C-42*
  - *Diagnostic messages C-42*
  - *Notes C-43*
- *FORM and NOFORM Execution Options C-45*
- *GEN and NOGEN Execution Options C-45*
- *ITEM Execution Option C-46*
- *LANG Execution Option C-46*
- *LINECNT Execution Option C-46*
- *LOAD and NOLOAD Execution Options C-47*

*MEMORY Execution Option C-47*
*SORMGIN Execution Option C-48*
*SOURCE and NOSOURCE Execution Options C-48*
*SSR Execution Option C-49*
*SYSSEP Execution Option C-49*
*TERM and NOTERM Execution Options C-49*
*TRANS Execution Option C-50*

*SAMPLE FORTRAN PLEX PROGRAM C-51*
*FINDORG Program C-52*

*JOB CONTROL LANGUAGE (JCL) C-63*
*Single-user JCL for All Job Steps C-63*
*Multi-User JCL for All Job Steps C-63*
*JCL Variations C-66*
*Additional Locate Files (single-user) C-66*
*Additional database files C-67*
*$COPY Library C-67*

This appendix provides FORTRAN-specific information for the PLEX facility (VS FORTRAN Release 3.0). It discusses the general format of PLEX commands, FORTRAN variable types for SYSTEM 2000 item types, COMMBLOCK and SUBSCHEMA declarations, and FORTRAN PLEX processor directives and execution options. A sample program and JCL are also included. You should use the material in the main part of this manual as the basic documentation.

# FORMAT OF PLEX COMMANDS

The PLEX processor scans a program for PLEX statements and commands, turns some statements and commands into FORTRAN comments, and generates FORTRAN statements to carry out the requested processing. As the PLEX processor scans, it expects to find PLEX statements and commands in a prescribed format.

In general, each executable PLEX command consists of an initial keyword, a database identifier or subschema record name, and one or more other specifications or program variable names. A PLEX command can extend over more than one line (see **Continuation of a Command** on page C-4). The keyword, identifiers, and other names in a command must not be divided across adjacent lines of the command. If a command word is so divided, the PLEX processor considers it to be two words.

The placement of a PLEX command on a line depends on the specifications made in the SORMGIN execution option (see **SORMGIN Execution Option** on page C-48), if that option was used. If SORMGIN is not used, the default columns for placement are as follows:

- If the NOFORM execution option is used, a PLEX command can appear anywhere from column 7 to column 72.
- If the FORM execution option is used, a PLEX command line must begin with the characters *PL in columns 1-3.

Columns 73-80 can contain program identification information. These columns are not read by the PLEX processor. Depending on specifications in the SORMGIN execution option, the PLEX processor can generate sequence numbers for source statements. If you use the defaults, the PLEX processor counts the source lines and puts the sequence numbers in columns 73-80 on the source listing.

## FORM and NOFORM Execution Option

Since PLEX commands are intermixed with FORTRAN statements in a program, the PLEX processor must recognize these two kinds of input and process only the PLEX commands. The PLEX processor offers a choice of two processing methods.

With NOFORM (the default), the PLEX processor examines the first syntactic unit of each line. If it is not a PLEX keyword, the line is assumed to be a FORTRAN statement. If the first word is a PLEX keyword, the PLEX processor looks further for other PLEX keywords making up a command. NOFORM is the most commonly used format. (See also **FORTRAN PLEX Processor Execution Options** on page C-39.)

The FORM option is a little more restrictive. With FORM, each line of each PLEX command (including the COMMBLOCK and SUBSCHEMA RECORD declarations) must begin with the characters *PL in columns 1-3, unless other columns are specified in the SORMGIN execution option. Also, each command must end with a period. After the PLEX processor finds the beginning of a PLEX command, it scans the rest of the line (and continuation lines, if there are any) until it finds the period terminating the command.

With FORM, PLEX command lines, including continuation lines, that do not begin with *PL in columns 1-3 are not scanned by the PLEX processor and, therefore, may be passed directly to the FORTRAN compiler. If continuation lines do not begin with *PL, the PLEX processor may flag syntax errors or pass the lines to the compiler, depending on the structure of the

command. The FORM option is supported only for compatibility with previous releases of PLEX software.

**Examples**

```
Col.      Col.
1         6

*PL        GET SSR1 NEXT.                  (acceptable with FORM or NOFORM)


*PL        COMMBLOCK/EMPLOYEE/RECNAM,      (acceptable with FORM or NOFORM)

*PL      - RCODE, FREE, LASTDR, PW, DRNO,

*PL      - LFPTR, LEVEL, CTIME, CDATE,

*PL      - CYCLE, SEPAR, TERMIN, DAMAGE.


           GETD SSR2 S2KCNT              (acceptable with NOFORM only)
```

## Continuation of a Command

All PLEX commands except STOP S2K can be continued on several lines. As with FORTRAN statements, a character other than blank or 0 in column 6 (or some other column specified with *b* in the SORMGIN option) indicates a continuation line. If a character other than blank or 0 does not appear in column 6 of a continuation line, the PLEX processor does not recognize the continuation. The actual continuation can begin anywhere after column 6 on the continuation line.

In addition, if the FORM option is in effect, the characters *PL must appear in columns 1-3 of each continuation line, followed by the continuation character in column 6.

**Examples with NOFORM**

```
Col.
6

    GET1 (12) SSR1

-   FIRST

    GET1 (12)

-      SSR1

-        FIRST
```

**Examples with FORM**

```
Col.    Col.
1       6

*PL      GETD (15) SSR2

*PL     -S2KCNT.


*PL         GETD (15)

*PL     -       SSR2

*PL     -          S2KCNT.
```

## Statement Numbers

If the NOFORM option is in effect, a PLEX command can have a statement number, unless the statement is the end of the range of a DO statement. When a PLEX command has a statement number, the PLEX processor turns the command into a comment and repeats the statement number in the first FORTRAN statement generated for the command. If a PLEX command will be the last statement executed in a DO loop, a CONTINUE statement completing the DO loop should be placed after the command.

### Examples

1. For example, suppose the PLEX processor encounters the following command:

```
Col.
1

 2000  GET1 ENTRY NEXT
```

   It turns the command into a comment and places the specified statement number in the first generated statement:

```
Col.
1

C2000  GET1 ENTRY NEXT

 2000  S2KDU1=4  (first generated statement)
```

2. A CONTINUE statement should follow a PLEX command that is the last statement executed in a DO loop:

```
            DO 100 I=  .  .  .
            .
            .
            GET1 ENTRY NEXT

100         CONTINUE
```

## Comments

You can intersperse comments between PLEX declarations and commands, but not between continuation lines of PLEX declarations and commands.

## One PLEX Command per Line

Each PLEX declaration and command must start on a separate line. Also, PLEX commands and FORTRAN statements cannot be mixed on the same line.

## Reserved Words

You cannot use the following names as procedure names in your PLEX program:

**S2KPL**
**PROC or PROCEDURE**
**SCHEMAS**
**S2KPLC** (*CICS only*)

S2KPL is the name of the routine that is called to process PLEX commands. If PROC, PROCEDURE, or SCHEMAS is used as a subroutine name, the FORTRAN statement END *procedure-name* is interpreted by the PLEX processor as the PLEX command END PROCEDURE or END SCHEMAS (old syntax for $BLOCKS), and this may produce undesirable results.

See **Reserved Words** on page 3-3 for information about reserved words and keyword restrictions for PLEX cmmands.

# SYSTEM 2000 ITEM TYPES AND FORTRAN VARIABLE TYPES

The correspondence between SYSTEM 2000 item types and FORTRAN variable types is summarized in Illus. C.1. This table shows how you must declare schema items in your program and how values must be placed in the fields so they can be interpreted correctly. It also shows how values retrieved from a database are placed in the program's fields.

**Illus. C.1** SYSTEM 2000 Item Types and FORTRAN Variable Types

| Item Type | FORTRAN Variables | Requirements/ Restrictions | From Database | To Database |
|---|---|---|---|---|
| TEXT | INTEGER or CHARACTER | If the value is shorter than its subschema item field, a byte of binary zeros must follow the end of the value in the SSR to avoid storage of trailing blanks for updates and to ensure successful comparisons in a where-clause.<br><br>There is no way of knowing the exact number of trailing blanks with a retrieval. | Stored left-justified and blank-filled or truncated on right as necessary.<br><br>Null = NULL in the first four bytes if it is variable type INTEGER.<br><br>Null = binary zero in the first byte if it is variable type CHARACTER. | Placed in the database left-justified, binary zero-filled, or carried over to the Extended Field Table as necessary.<br><br>User should place the value left-justified and place a byte of binary zeros at the end, if necessary. Null = binary zeros in the first byte. |
| CHARACTER | INTEGER or CHARACTER | | Same as TEXT. | Placement same as TEXT. Null = all blanks in the entire field or binary zeros in the first byte. |
| DATE | INTEGER (2) or CHARACTER | Must be a two-word field for INTEGER and 6 or 8 bytes for CHARACTER. The default is MMDDYYYY. | Placed according to format chosen.<br><br>Null = same as TEXT and CHARACTER. | Value should conform to date format chosen. Null = binary zeros. |
| INTEGER | INTEGER | Abend if the value cannot be represented in 32 bits. | Binary representation is retrieved.<br><br>Null = NULL. | Placed in the database right-justified and zero-filled. Number of digits must not exceed what the schema item picture allows. Null = binary zeros. |
| DECIMAL or MONEY | REAL or DOUBLE | Number of significant digits to the left of the decimal point must not exceed what the schema item picture allows. | Floating point representation is retrieved.<br><br>Null = NULL in the first word. | Decimal point is aligned and zero-fill occurs as necessary. Number of digits left of decimal point cannot exceed the item picture size. If number of digits right of decimal point exceeds the item picture, rounding or truncation occurs. Value cannot exceed maximum allowed by item picture. Null = binary zeros. |

*continued on next page*

**Illus. C.1** *continued*

| Item Type | FORTRAN Variables | Requirements/ Restrictions | From Database | To Database |
|---|---|---|---|---|
| REAL | REAL or DOUBLE | Requires fullword alignment or doubleword alignment.<br><br>Zero means a positive zero. Negative zero is not allowed; causes return code 10. | Internal normalized floating point number is placed in the subschema. External floating point numbers are not allowed in the subschema.<br><br>Null = NULL. | Must be internal normalized floating point number. SYSTEM 2000 software performs no conversions. External floating point numbers are not allowed.<br><br>Null = NULL. |
| DOUBLE | DOUBLE or REAL | Requires doubleword alignment.<br><br>Other rules same as REAL. | Internal normalized floating point number is placed in the subschema. External floating point numbers are not allowed in the subschema.<br><br>Null = NULL in the first word. | Must be internal normalized floating point number. SYSTEM 2000 software performs no conversions. External floating point numbers are not allowed.<br><br>Null = NULL in the first word. |
| UNDEFINED | INTEGER or CHARACTER | Each byte represents one byte of data. Data conversions from hexadecimal to display are not performed. If an UNDEFINED value overflows its picture, return code 10 is issued. | Value placed left-justified in subschema with trailing zero-fill.<br><br>Null = all binary zeros. | All 256 EBCDIC characters are allowed. Placed in database left-justified with trailing zero-fill.<br><br>Null = all binary zeros. |

## Item Types and FORTRAN Variable Types for Items in Where-Clauses

Among other things, the where-clause provides two kinds of comparison of data values:

- two data values in the database
- a value in the database with a value in a program variable, which can belong to a subschema record.

A FORTRAN program variable in working storage or in a subschema record can be variable type INTEGER or CHARACTER for use in where-clauses if the program variable is being compared to type CHAR, TEXT, UNDEFINED, or DATE item values in the database.

In the first kind of comparison, both data values in the database must be from the same group of compatible SYSTEM 2000 item types. There are two such groups: SYSTEM 2000 CHAR, TEXT, and UNDEFINED type items belong to one group; SYSTEM 2000 INTEGER, DECIMAL, MONEY, REAL, DOUBLE, and DATE type items belong to another group. Before the comparison is made, the value of the right operand is reformatted according to the SYSTEM 2000 item type and length of the left operand. Thus, if the left operand is an INTEGER value and the right operand is a DECIMAL value, the DECIMAL or MONEY value is truncated before the comparison. Therefore, the user must place the INTEGER value as the right operand to avoid the truncation of the DECIMAL value.

When a DATE value is the right operand and the left operand is an INTEGER value, the DATE value is converted to an INTEGER value, which is the difference in number of days between the date and October 15, 1582. If the left operand is a DATE value and the right operand is an INTEGER value *n*, the latter is treated as the date which comes *n* days after October 15, 1582.

CHAR, UNDEFINED, and TEXT type items are compared as they appear; that is, extraneous blanks are not suppressed from TEXT and UNDEFINED values for comparison with CHAR values.

In the second kind of comparison, the left operand is always a subschema item. With this kind of comparison, the right operand of a binary operator or one or both right operands of a ternary operator can be either of the following:

- program variables not in subschema records
- subschema items.

Each right operand of a ternary operator can be a variable either within or outside a subschema record.

**Program variables as right operands** When a program variable not in a subschema record serves as a right operand, the value for the left operand is taken from the database; SYSTEM 2000 software uses the data description of the left operand to interpret the value of the right operand.

That is, SYSTEM 2000 software uses the type and length of the left operand, as specified for a subschema item, as the type and length of the program variable(s) in the right operand. If the value of the right operand is incompatible with that of the left operand, return code 17 results. For character data, the left and right operands should be of equal length. If, however, the specified length of a character variable in the right operand or the length of the value within the variable field is shorter than the item length in the left operand, the right operand value must be followed by a byte of zero bits; otherwise, results of the comparison of operands are unpredictable.

**Subschema items as right operands** When a subschema item serves as a right operand, the value for the left operand is taken from the database. SYSTEM 2000 software uses the data type and value of the right operand as they appear in a subschema record. Thus, in setting up the values of the left and right operands for comparison, SYSTEM 2000 software does not adjust the right operand's value to conform to the data type of the left operand. If the right operand contains character data and its value has fewer characters than its picture allows, the right value must be followed by a byte of zero bits; otherwise, results of the comparison of operands are unpredictable.

**Considerations**

- The use of TEXT values requires special care. SYSTEM 2000 software fills TEXT fields with binary zeros on the right because blanks are significant characters in TEXT type items. Therefore, when a TEXT value in a subschema item is shorter than its field length and is used in a where-clause or for an update, the PLEX program must place a byte of binary zeros at the end of the value. Otherwise, the where-clause comparison would fail or the updated value would have extra trailing blanks in the database.

- Two return codes are associated with subschema item types. When a subschema record or any of its items is referenced for the first time, all items in that subschema record are validated for acceptable types. SYSTEM 2000 software issues return code 17 if the subschema item type does not conform to the corresponding schema item type. If all subschema items have valid types, the software opens the validated subschema record.

  If your program places an unacceptable data value in a correctly declared subschema item field, SYSTEM 2000 software issues return code 10 for a LOCATE or GET1 command whose where-clause refers to that subschema item. Return code 10 also pertains to a MODIFY or INSERT command that has an unacceptable value for an item.

  When return codes 10 and 17 are issued, the ordinal position of the item in error within the subschema record is placed in the eighth field of the appropriate COMMBLOCK. Also, for a where-clause that contains an error, the ordinal position (left-to-right) of the erroneous condition is placed in the eighth COMMBLOCK field.

- The binary representation of the characters 'NULL' requires special consideration for retrievals and updates in one specific instance. If an INTEGER item in the database could conceivably have the value -706,423,853 (the binary equivalent of the characters 'NULL') in one data record and could be null in another data record, both items would appear to have identical values when retrieved into an INTEGER subschema item field.

  While -706,423,853 and the characters 'NULL' are identical in binary representation, they are different in floating-point representation: - 706423853 E+09 (for the numeric value) vs. - 172897124 E+26 (for 'NULL'). To take advantage of this difference, the subschema item should in this case be declared with type REAL. The schema item must be defined as DECIMAL, MONEY, REAL, or DOUBLE in the database definition.

# COMMBLOCK DECLARATIONS

The COMMBLOCK declaration sets up a communication area for a database to keep the status flag, return code, and other identification information.

- A COMMBLOCK can be passed as a parameter or can be referenced in a labeled common area.
- A COMMBLOCK can be declared INTERNAL to make it available only to the main program or subprogram that contains it.
- A COMMBLOCK can be given a name that is different from the database name.

A separate COMMBLOCK declaration is required for each database that will be accessed by your program. COMMBLOCK declarations should appear only where FORTRAN specification statements are allowed in a program. A COMMBLOCK declaration resembles a labeled COMMON statement. It consists of two main parts: the name of the COMMBLOCK and the names of the fields in the COMMBLOCK. In the first part, you can specify the INTERNAL option if the COMMBLOCK will be known only to the main program or subprogram that contains it (see Example 3). Or, you can specify the PARM option, which signifies that the COMMBLOCK will be received as a parameter (see Example 4). If neither of these options is specified, the PLEX processor defines a labeled common storage area for the COMMBLOCK.

Also, you give the database name in the first part of the COMMBLOCK declaration. In addition, you can give an alternate name (an alias) to the COMMBLOCK (see Example 2). If you do not specify an alias, the default COMMBLOCK name is the first six characters of the database name.

In the second part of the COMMBLOCK declaration, you specify the COMMBLOCK field names, without size or type. Fourteen fields are required; the fifteenth field is optional.

Depending on whether you specify the INTERNAL or PARM option, the PLEX processor translates the declaration into one or more FORTRAN statements. If neither INTERNAL nor PARM is specified, the PLEX processor generates labeled COMMON statements specifying the name of the COMMBLOCKs. It also generates the variables specified in the COMMBLOCK declaration. A corresponding EQUIVALENCE statement is generated to associate the individual variables with fields in the COMMBLOCK array. Before the field for the first COMMBLOCK variable, the PLEX processor places a 28-byte control block providing internal identification information used in executing PLEX commands.

The PLEX processor also generates initializations for all fields of the COMMBLOCK, as well as the control block, in a BLOCK DATA subprogram. The PLEX processor generates a BLOCK DATA subprogram for each main program or subroutine that includes generated labeled COMMON statements for COMMBLOCKs or subschema records.

If you specify the INTERNAL option, the PLEX processor generates an INTEGER and an EQUIVALENCE statement, as mentioned above, but omits the COMMON statements. A DATA statement for the control block and the COMMBLOCK fields is also generated. If you specify the PARM option, the PLEX processor generates an INTEGER statement, a CHARACTER statement, and associated comments, omitting the COMMON statement.

## Format

```
|[INTERNAL] COMMBLOCK/[name EQ] database-name/var-1,...,var-14[,var-15]
|[PARM]
```

*name* is a valid, unique FORTRAN variable name of up to six characters.

*database-name* is the name of a SYSTEM 2000 database.

*var-n* is a valid FORTRAN variable name of up to six characters, naming a field in the COMMBLOCK. See below for a discussion of each of the COMMBLOCK fields.

## COMMBLOCK Fields

The FORTRAN COMMON block name is one to six characters. The PLEX processor determines the type and size of all COMMBLOCK fields. It generates type CHARACTER variables for the following COMMBLOCK fields:

| COMMBLOCK Field | Sample Name | No. of Bytes | Description |
|---|---|---|---|
| Variable 1 | SBSCNM | 32 | Subschema name |
| Variable 3 | FREE | 4 | Free area for user |
| Variable 5 | PASSWD | 4 | Database password |
| Variable 9 | TIME | 8 | Time of the last update |
| Variable 10 | DATE | 8 | Date of the last update |
| Variable 12 | SEPAR | 4 | System separator |
| Variable 13 | TERMIN | 4 | Entry terminator |

For the other COMMBLOCK fields, the PLEX processor generates type INTEGER variables.

The PLEX processor initializes each field to zero in an INTERNAL COMMBLOCK and in a COMMBLOCK for which a labeled common storage area is defined. For an INTERNAL COMMBLOCK, these initial values are given in a DATA statement in the generated program code. However, for a COMMBLOCK corresponding to a labeled common storage area, a DATA statement in the BLOCK DATA subprogram includes the initial values. One computer word (4 bytes) is reserved for all fields except three, for which extra words are reserved. Each COMMBLOCK field is discussed next.

*var-1* (subschema record name)

If a PLEX command refers to a subschema record, SYSTEM 2000 software places the name of the subschema record in this field. If the command does not reference a subschema record, SYSTEM 2000 software places blanks in this field. When a command is rejected because of an error in a subschema item, SYSTEM 2000 software stores the name or alias of that subschema item in this field. Thus, you can use this field as a debugging aid.

*var-2* (return code)

The values returned to your PLEX program in this field are very important. This field contains the return code from SYSTEM 2000 software after the execution of each command referring to that COMMBLOCK. Every PLEX program should check the value in this field after every PLEX command and take any action necessary. Failure to check the return code immediately can multiply errors and cause problems that would be difficult to solve. See *SYSTEM 2000 Messages and Codes, Version 12, First Edition* for the meaning of each return code.

**Note**: The variable S2KRTC in the S2KDUM control block contains the return code after every command, regardless of whether the command refers to a database. Since some commands, such as START S2K, do not refer to a database, their return codes are placed only in S2KRTC, not in any COMMBLOCK.

*var-3* (for the user)

This field is not used by SYSTEM 2000 software. You can store information about the database in this field.

*var-4* (last data record indicator)

SYSTEM 2000 software sets this field to one when a GET command retrieves the data record for the last address in a Locate File. Otherwise, this field is set to zero. (This field is also set to zero for GETP.)

You can use this field after a LOCATE command in order to control the retrieval of data records from a located subset of data records. For example, suppose LAST is the name of the fourth COMMBLOCK field.

```
          :
          :
     LOCATE SSR WHERE . . .
     (Check return code)
10 IF (LAST.NE.0) GO TO 20
     GET SSR . . .
     (Check return code)
          :
          :
   GO TO 10
          :
          :
20 CONTINUE
```

*var-5* (password)

A valid password (master, secondary, or DBA) must be present in the fifth COMMBLOCK field before your program issues an OPEN command for the database and before it refers again to the database after working with another database. The password in the COMMBLOCK is not referred to again until the database is opened again (either explicitly by an OPEN command or implicitly by a command referencing the database after your program used another database).

Until a CLOSE command is issued for the database, SYSTEM 2000 software automatically keeps track of the access authorities allowed for the secondary passwords. SYSTEM 2000 software also keeps track of the commands available to the DBA password. You can move the password into this field with an assignment or input statement before opening the database.

*var-6* (number of records selected by a LOCATE command)

SYSTEM 2000 software sets this field after each successful LOCATE command or GET1 command with a where-clause for the database. For a LOCATE command, the value in this field indicates the number of data records that were selected. For a GET1 command with a where-clause, this value is the number of qualified records.

This field can be used after a LOCATE command in order to control the retrieval of data records from a selected subset of records.

For example, suppose the name of the sixth COMMBLOCK field is NUMREC in the following example:

```
      .
      .
      .
   LOCATE SSR WHERE . . .
   (Check return code)
      .
      .
      .
DO 50 I=1, NUMREC
      .
      .
      .
   GET SSR
   (Check return code)
      .
      .
      .
50 CONTINUE
```

*var-7* (Locate File pointer: position of retrieved data record in Locate File)

This field contains the ordinal of the data record address in the Locate File when SYSTEM 2000 software retrieves a data record. (See illustration in **COMMBLOCK Fields** on page 5-3.) You can use this field for debugging when an error return code occurs.

*var-8* (level)

During the normal execution of a command, SYSTEM 2000 software uses this field to store the hierarchical level of the data record accessed. However, under error conditions this field contains information helpful for debugging, such as the ordinal of a rejected item in its subschema record, the ordinal of a component in a where-clause responsible for the rejection of the where-clause, or the level of a data record removed from the database. (See illustration in **COMMBLOCK Fields** on page 5-3.)

*var-9* (cycle time)

This field contains the time of day for the last update to the database. The time format is HH:MM:SS, that is, hours, minutes, seconds. SYSTEM 2000 enters the time in this field whenever the database is referenced.

*var-10* (cycle date)

This field contains the date of the last update to the database. The format is MM/DD/YY (month/day/year). SYSTEM 2000 software enters the date in this field whenever the database is referenced.

*var-11* (cycle number)

SYSTEM 2000 software places the cycle number of the database in this field whenever the database is referenced. SYSTEM 2000 software maintains a cycle number for each database internally in the database files. The cycle number indicates how many times the data in the database were updated. The internal cycle number is moved into this field of the COMMBLOCK so that the user can associate the cycle number with the updates to the database.

*var-12* (system separator)

When the database is opened, SYSTEM 2000 software places the system separator (right-justified and binary zero-filled) in this field. The default system separator is an asterisk.

**Note**: Your PLEX program must use an asterisk as the system separator with ternary operators in a where-clause unless you specify a different character with the SYSSEP execution option (see **SYSSEP Execution Option** on page C-49).

*var-13* (entry terminator)

When the database is opened, SYSTEM 2000 software places the entry terminator word in this field. The default entry terminator word is END, left-justified and blank-filled. The entry terminator word is not used in PLEX.

*var-14* (database damage flag)

When the database is opened, SYSTEM 2000 software moves the internal damage flag for the database into this field. A zero value indicates an undamaged database, and a value of one indicates that the database is damaged. Your program should check this field after the OPEN command is given. Updates are not permitted for a damaged database; however, data retrievals are allowed.

*var-15* (parent - optional field)

The COMMBLOCK can include this optional fifteenth field. When any retrieval command is executed, SYSTEM 2000 software uses this field to store the value of the Hierarchical Table pointer for the parent of the data record retrieved. This field can be used to distinguish between similar data records under different parents. The value of this field, being an internal address, is not meaningful to the user. However, you can save it after a retrieval and then compare it with the value of the parent in the next retrieval. (See illustration in **COMMBLOCK Fields** on page 5-3.)

## Examples

These examples show source COMMBLOCK statements, followed by the expanded code produced by the PLEX processor for input to the VS FORTRAN Release 3.0 compiler. Where appropriate, the examples include portions of the BLOCK DATA subprogram, also generated by the PLEX processor.

### Example 1

This example shows a COMMBLOCK for the EMPLOYEE database. Note that no type or subscript is specified for any COMMBLOCK field.

```
 COMMBLOCK/EMPLOYEE/RECNAM,CODE,FREE,LASTDR,PW,DRNO,
-LFPNTR,LEVEL,TIME,DATE,CYCLE,SEPAR,TERMIN,DAMAGE
```

Here is the VS FORTRAN Release 3.0 expanded source. The PLEX processor makes a comment out of the entire COMMBLOCK statement and generates one COMMON block. COMMBLOCK fields 1, 3, 5, 9, 10, 12, and 13 are variable type CHARACTER.

```
C     COMMBLOCK/EMPLOYEE/RECNAM,CODE,FREE,LASTDR,PW,DRNO,
C    -LFPNTR,LEVEL,TIME,DATE,CYCLE,SEPAR,TERMIN,DAMAGE
      COMMON/EMPLOY/EMPLOY
      INTEGER EMPLOY(30),CODE,LASTDR,DRNO,LFPNTR,LEVEL,CYCLE, DAMAGE
      CHARACTER RECNAM*32,FREE*4,PW*4,TIME*8,DATE*8,SEPAR*4,TERMIN*4
      EQUIVALENCE (EMPLOY(8),RECNAM),(EMPLOY(16),CODE),(EMPLOY(17),
     -FREE),(EMPLOY(18),LASTDR),(EMPLOY(19),PW),(EMPLOY(20),DRNO),
     -EMPLOY(21),LFPNTR),(EMPLOY(22),LEVEL),(EMPLOY(23),TIME),
     -EMPLOY(25),DATE),(EMPLOY(27),CYCLE),(EMPLOY(28),SEPAR),(EMPLOY(29
     -),TERMIN),(EMPLOY(30),DAMAGE)
```

The corresponding portion of the BLOCK DATA subprogram initializes the COMMBLOCK control block (the first 28 bytes) as well as all COMMBLOCK fields.

```
BLOCK DATA
COMMON/EMPLOY/EMPLOY
INTEGER EMPLOY(30)
DATA EMPLOY/ZC5D4D7D3,ZD6E8C5C5,2*Z40404040,0,120,24*0/
END
```

### Example 2

The variation in the COMMBLOCK statement format used here is acceptable to the PLEX processor.

A COMMBLOCK for the EMPLOYEE database can be given the name EMP, for example, instead of EMPLOY. Also, it can be declared with the optional fifteenth field that stores the address of the parent of the most recently retrieved record.

```
 COMMBLOCK/EMP EQ EMPLOYEE/
-    SSRNAM,
-    RCODE,
-    FREE,
-    LASTDR,
-    PASSWD,
-    RECNO,
-    LFPTR,
-    LEVEL,
-    TIME,
-    DATE,
-    CYCLE,
-    SEPTOR,
-    TERMTR,
-    DAMAGE,
-    PARENT
```

The PLEX processor turns the COMMBLOCK declaration into a comment and generates one COMMON block. COMMBLOCK fields 1, 3, 5, 9, 10, 12, and 13 are variable type CHARACTER. In addition, the PLEX processor generates INTEGER, CHARACTER, and EQUIVALENCE statements for the COMMBLOCK fields, including the fifteenth COMMBLOCK field.

```
C      COMMBLOCK/EMP EQ EMPLOYEE/
C     -    SSRNAM,
C     -    RCODE,
C     -    FREE,
C     -    LASTDR,
C     -    PASSWD,
C     -    RECNO,
C     -    LFPTR,
C     -    LEVEL,
C     -    TIME,
C     -    DATE,
C     -    CYCLE,
C     -    SEPTOR,
C     -    TERMTR,
C     -    DAMAGE,
C     -    PARENT
```

```
      COMMON/EMP/EMP
      INTEGER EMP(31),RCODE,LASTDR,RECNO,LFPTR,LEVEL,CYCLE,DAMAGE,PARENT
      CHARACTER SSRNAM*32,FREE*4,PASSWD*4,TIME*8,DATE*8,SEPTOR*4,
     -TERMTR*4
      EQUIVALENCE (EMP(8),SSRNAM),(EMP(16),RCODE),(EMP(17),FREE),(EMP(18
     -),LASTDR),(EMP(19),PASSWD),(EMP(20),RECNO),(EMP(21),LFPTR),(EMP(22
     -),LEVEL),(EMP(23),TIME),(EMP(25),DATE),(EMP(27),CYCLE),(EMP(28),
     -SEPTOR),(EMP(29),TERMTR),(EMP(30),DAMAGE),(EMP(31),PARENT)
```

In the corresponding portion of the BLOCK DATA subprogram, the PLEX processor generates initializations for the control block and all COMMBLOCK fields.

```
BLOCK DATA
COMMON/EMP/EMP
INTEGER EMP(31)
DATA EMP/ZC5D4D7D3,ZD6E8C5C5,2*Z40404040,0,124,25*0/
```

**Example 3**

This sample COMMBLOCK is for a COMPANY database. It is declared with the INTERNAL option.

```
      INTERNAL COMMBLOCK/COMPANY/ RECNAM,CODE,FREE,LASTDR,PW,DRNO,
     -LFPTR,LEVEL,CTIME,CDATE,CYCLE,SEPTOR,TERMTR,DAMAGE,PARENT
```

In the expansion from the PLEX processor, no COMMON statements are generated, because this COMMBLOCK will be known only to the subroutine that contains it.

The generated INTEGER statement names the array COMPAN, the first six characters of the database name. The second generated array is type CHARACTER for COMMBLOCK fields 1, 3, 5, 9, 10, 12, and 13, and the array name is COMPA1. Notice that the DATA statement is included here in the generated code, rather than in a BLOCK DATA subprogram, because no labeled COMMON block is established for this INTERNAL COMMBLOCK.

```
C     INTERNAL COMMBLOCK/COMPANY/ RECNAM,CODE,FREE,LASTDR,PW,DRNO,
C    -LFPTR,LEVEL,CTIME,CDATE,CYCLE,SEPTOR,TERMTR,DAMAGE,PARENT
      INTEGER COMPAN(31),CODE,LASTDR,DRNO,LFPTR,LEVEL,CYCLE,DAMAGE,
     -PARENT
      CHARACTER RECNAM*32,FREE*4,PW*4,CTIME*8,CDATE*8,SEPTOR*4,TERMTR*4
      EQUIVALENCE (COMPAN(8),RECNAM),(COMPAN(16),CODE),(COMPAN(17),
     -FREE),(COMPAN(18),LASTDR),(COMPAN(19),PW),(COMPAN(20),DRNO),
     -(COMPAN(21),LFPTR),(COMPAN(22),LEVEL),(COMPAN(23),CTIME),
     -(COMPAN(25),CDATE),(COMPAN(27),CYCLE),(COMPAN(28),SEPTOR),
     -(COMPAN(29),TERMTR),(COMPAN(30),DAMAGE),(COMPAN(31),PARENT)
      DATA COMPAN/ZC3D6D4D7,ZC1D5E840,2*Z40404040,0,124,25*0/
```

### Example 4

Here is a declaration of a COMMBLOCK named INVNTR for an INVENTORY database. It includes the PARM option. This option is used for a COMMBLOCK that appears as a parameter in a called subroutine. In this case, the calling routine is passing an INTERNAL COMMBLOCK as a parameter.

```
 PARM COMMBLOCK/ INVNTR EQ INVENTORY/ RECNAM,CODE,FREE,LASTDR,
-                                    PW,DRNO,LFPTR,LEVEL,TIME,
-                                    DATE,CYCLE,SEPTOR,TERMIN,
-                                    DAMAGE
```

The PLEX processor generates an INTEGER statement for an array named INVNTR. The PLEX processor generates an INTEGER array and a CHARACTER array. The EQUIVALENCE statement is a comment.

```
C     PARM COMMBLOCK/ INVNTR EQ INVENTORY/ RECNAM,CODE,FREE,LASTDR,
C    -                                     PW,DRNO,LFPTR,LEVEL,TIME,
C    -                                     DATE,CYCLE,SEPTOR,TERMIN,
C    -                                     DAMAGE
      INTEGER INVNTR(30)
      ,CODE,LASTDR,DRNO,LFPTR,LEVEL,CYCLE,DAMAGE
      CHARACTER RECNAM*32,FREE*4,PW*4,TIME*8,DATE*8,SEPTOR*4,TERMIN*4
C     EQUIVALENCE (INVNTR(8),RECNAM),(INVNTR(16),CODE),(INVNTR(17),
C    -FREE),(INVNTR(18),LASTDR),(INVNTR(19),PW),(INVNTR(20),DRNO),
C    -(INVNTR(21),LFPTR),(INVNTR(22),LEVEL),(INVNTR(23),TIME),(INVNTR(25
C    -),DATE),(INVNTR(27),CYCLE),(INVNTR(28),SEPTOR),(INVNTR(29),
C    -TERMIN),(INVNTR(30),DAMAGE)
```

## Considerations

- The COMMBLOCK contains important debugging information for checking program logic. It should be printed in an understandable format when an unexpected return code is issued.

- If an INTERNAL COMMBLOCK will be received as a parameter in a subroutine, the PARM option must be included in the COMMBLOCK declaration in that subroutine. The calling subroutine, which contains the INTERNAL COMMBLOCK, must pass the name of the COMMBLOCK.

- If you do not specify the INTERNAL or the PARM option for the COMMBLOCK, the PLEX processor generates a labeled COMMON statement for the COMMBLOCK. In this situation, a COMMBLOCK and its fields can be referenced by a main program and one or more subroutines, just as any other labeled common storage area can be referenced.

- Every main program and subroutine with one or more PLEX commands must contain the $BLOCKS directive. In addition, each main program and subroutine accessing a database must reference the same COMMBLOCK and subschema record(s) declared for the database, by passing or receiving them as parameters or by referencing the same area in common storage.

- The name of each COMMBLOCK field must be unique in the main program or subroutine. If COMMBLOCKs include fields with identical names, the FORTRAN compiler detects the error and issues an error message.

- A COMMBLOCK must contain 14 fields. If it does not, the PLEX processor gives an error message. You can also include the optional fifteenth field.

- If the name of a COMMBLOCK is not unique within the subprogram, the PLEX processor issues an error message.

- When a Locate File is currently in use by your program, the fourth, sixth, and seventh fields of the COMMBLOCK are set for that Locate File. When your program uses one Locate File, then switches to a second one, those COMMBLOCK fields are set with the appropriate information for the second Locate File. If your program resumes retrievals with the first Locate File, the information for that Locate File is brought back into the COMMBLOCK. For example, your program can give GET commands to retrieve records for the first five addresses in the first Locate File, then work with another Locate File for a while, and later resume retrievals at the point where it stopped earlier in the first Locate File.

- SYSTEM 2000 software sets the sixth COMMBLOCK field when a PLEX program issues a GET1 command with a where-clause. The value of this field is the number of data records qualified by the where-clause.

- You can change the database name associated with a COMMBLOCK during program execution, which is useful for processing two or more databases in which the components to be accessed have identical definitions.

The name of the database associated with the COMMBLOCK is kept in the first 16 bytes of the COMMBLOCK control block, which the PLEX processor generates and places before the first COMMBLOCK field. The PLEX processor associates the COMMBLOCK with a database by initializing this 16-byte field to the database name (specified after EQ, if EQ is used) given in the COMMBLOCK statement.

Your FORTRAN PLEX program can change this name at execution time by assigning a different value to the first four words (16 bytes) of the COMMBLOCK control block. Each time your PLEX program assigns the name of another database to that 16-byte field, it must then re-issue the OPEN command for the COMMBLOCK, regardless of whether the database was opened. The new database name in the COMMBLOCK, together with the OPEN command for that database, associates each SSR for that COMMBLOCK with the alternate database. Subschema item declarations must be compatible with component definitions in each database with which an SSR is associated. If they are not, return code 17 results when the SSR is referenced.

To change the database name, use a procedure similar to the one shown below, where EMPLOY(I) contains the database name in the FORTRAN COMMON block.

```
      INTEGER ALTNAM(4)
      DATA ALTNAM/4HNEWD,4HBNAM,2*4H      /

      DO 10 I=1,4
   10 EMPLOY(I)=ALTNAM(I)
```

- You can also use another method for processing two or more databases in which the components to be accessed have identical definitions. This method involves one or more subroutines. Place the SUBSCHEMA RECORD declarations and a PARM COMMBLOCK declaration(s) in the called subroutine. The calling program includes a separate INTERNAL COMMBLOCK for each database that will be accessed.

  The calling program decides which database will be accessed and passes the appropriate COMMBLOCK(s) to the subroutine, which then issues one or more commands referring to the COMMBLOCK(s) or the SSRs. In this case, when the subroutine is invoked, SYSTEM 2000 software automatically takes care of opening the database and opening (validating) the SSRs as necessary.

  With this technique, the called program does not need to include an OPEN command for the database that will be accessed unless the damage flag must be checked before update commands are issued.

# SUBSCHEMA RECORD DECLARATIONS

A SUBSCHEMA RECORD declaration associates FORTRAN variables with a database schema record (SR) and the items in the schema record. The subschema record represents a schema record, and the items in the subschema record represent the corresponding items in the schema record. The SUBSCHEMA RECORD declaration also allocates a storage area for items in the corresponding data records. Update and retrieval commands use subschema records as input and output areas, respectively, for data records in a database. A database can be accessed only through a subschema record.

- A subschema record can be passed as a parameter.
- A subschema record can be declared INTERNAL, to make it available only to the main program or a subprogram that contains it.
- The name of a subschema record can be different from the name of the corresponding schema record.
- The names of the items of a subschema record can be different from the names of the corresponding schema.
- Items can appear in any order in a subschema record, regardless of their order in a schema record.
- Six different display formats are available for items of type DATE.

At least one subschema record (SSR) should be declared for each schema record representing data records that will be accessed by your program. SSR declarations should appear only where FORTRAN specification statements are allowed in a program. A subschema record declaration resembles a labeled COMMON statement. As such, it consists of two main parts: the name of the subschema record and database and the names of the variables in the subschema record.

In the first part, you can specify the INTERNAL option (see Example 3) if the SSR will be known only to the main program or a subprogram that contains it. Or you can specify the PARM option (see Example 4), indicating that the SSR will be received as a parameter. If neither of these options is specified, the PLEX processor defines a labeled common storage area for the SSR. Also, you specify the name of the subschema record in the first part of the subschema record declaration, along with either the COMMBLOCK name or the database name.

In the second part of the SSR declaration, you specify the names, types, and sizes of subschema items. For each subschema record, the items must correspond to items in a single schema record. The name (alias) of a subschema item can be the component name or C-number (the letter C followed by the component number) of a schema item, or it can be an alternate identifier that is acceptable to FORTRAN (see Examples 2-4). A component name can contain blanks and, therefore, might not be an acceptable FORTRAN identifier, but each C-number is. The use of an alternate identifier is especially helpful when the component name is not acceptable in FORTRAN and yet an identifier is required that is more meaningful than the C-number. If the alias is not the component name or C-number, the component name or C-number must appear, preceded by the keyword EQ, in the item declaration.

Each subschema item can be declared with FORTRAN variable type INTEGER, REAL, or DOUBLE. Also, FORTRAN type CHARACTER can be used for schema items of type CHAR, TEXT, UNDEFINED, or DATE. The variable type should be compatible with the item type of the corresponding database item. See **SYSTEM 2000 Item Types and FORTRAN Variable Types** on page C-7 for a discussion of item types and FORTRAN variable types.

A subschema item for a type DATE item can be declared INTEGER(2) or type CHARACTER with six or eight bytes. Also, item type DATE can be declared with any of six date formats with a FORM statement. The FORM statement must appear following the EQ statement, if there is one, in the item declaration. The date format defaults to MMDDYYYY.

Depending on the option specified in the SSR declaration, the PLEX processor translates the declaration into one or more FORTRAN statements. If you do not specify INTERNAL or PARM, the PLEX processor generates a labeled COMMON statement with the name of the SSR. It also generates an INTEGER statement that dimensions an array for the SSR. Additional type and DIMENSION statements are generated according to the specifications for SSR items. If no REAL, DOUBLE, INTEGER, or CHARACTER specification is included, the item type is assigned according to FORTRAN naming conventions or through a user-coded IMPLICIT statement. The PLEX processor generates EQUIVALENCE statements to associate the individual variables with elements in the SSR array.

In addition, the PLEX processor inserts corresponding initializations in a BLOCK DATA subprogram. These initializations provide internal identification information used in executing PLEX commands. The PLEX processor generates a BLOCK DATA subprogram for each main program or subroutine that includes labeled COMMON statements for COMMBLOCKs and subschema records.

If you specify the INTERNAL option, the PLEX processor generates INTEGER, CHARACTER, and EQUIVALENCE statements, as mentioned above, but omits COMMON statement(s). DATA statements for variable initializations are also generated. If you specify the PARM option, the PLEX processor generates only INTEGER and CHARACTER statements and associated comments, and omits COMMON statements.

## Format

```
|[INTERNAL] SUBSCHEMA [RECORD]/name OF database-identifier/[item-list]
|[PARM]
```

| | |
|---|---|
| *name* | is a valid, unique FORTRAN variable name of up to six characters. |
| *database-identifier* | is the name of a SYSTEM 2000 database or the name of the COMMBLOCK. |
| *item-list* | *item-declaration* [, *item-list* ] |

Abbreviations: SUBSCHEMA = SSR

*item-declaration* — *alias* [( *type* )] [ *EQ-specification*] [*FORM-specification*]

*alias* — is a valid FORTRAN variable name of up to six characters, naming a subschema item.

*type* —
**|CHARACTER** (*size*)
**|INTEGER** [(*size*)]
**|REAL**
**|DOUBLE**
|*size*

Type CHARACTER can be used with schema item types CHAR, TEXT, UNDEFINED, or DATE.

*size* — is specified for INTEGER and CHARACTER variables: number of INTEGER words or CHARACTER bytes that will be reserved for the variable. The default is 1. If you specify *size* without INTEGER, your program must ensure that the variable is type INTEGER through a type statement or through the predefined FORTRAN naming convention for integer names.

**EQ**-*specification* — **EQ** |*component-name*
|*C-number*

If you omit the EQ specification, the alias is assumed to be a component name or C-number.

*component-name* — is the name of an item in the database schema.

*C-number* — is the letter C followed by a component number. The number can have up to four digits, including leading zeros. For example, SYSTEM 2000 software interprets C0001 as C1.

**FORM**-*specification* — FORM *date-format*

*date-format* — is MMDDYYYY, MMDDYY, DDMMYYYY, DDMMYY, YYYYMMDD, or YYMMDD. The default date format is MMDDYYYY.

**INTERNAL**
**PARM** — If neither option is specified, a labeled common storage block is established for the subschema record.

---

Abbreviations:
CHARACTER = C
INTEGER = I
REAL = R
DOUBLE = D

## Examples

Examples of SUBSCHEMA RECORD declarations for the EMPLOYEE database are presented here. Each example shows how your program can declare the subschema record, then how the PLEX processor translates the declaration for input to the FORTRAN compiler. Where appropriate, the examples include portions of the BLOCK DATA subprogram also generated by the PLEX processor.

### Example 1

In this example, subschema record DIRECT includes three items; all three aliases are C-numbers. Because no type is specified for the items, they are all type INTEGER because of the IMPLICIT INTEGER statement. One word is reserved for C1, five for C2, and four for C3.

```
IMPLICIT INTEGER(A-W)
    .
    .
    .
SUBSCHEMA RECORD/DIRECT OF EMPLOYEE/ C1,C2(5),C3(4)
```

The statements below show the same subschema record declaration, augmented by the PLEX processor. The declaration line in the program was translated into a comment. A COMMON statement, with the label DIRECT, was given the type INTEGER. In addition, a DIMENSION statement was generated for C2 and C3, with an accompanying EQUIVALENCE statement. The FORTRAN compiler will generate the INTEGER statement needed for labels C1, C2, and C3.

```
C  SUBSCHEMA RECORD/DIRECT OF EMPLOYEE/ C1,C2(5),C3(4)
   COMMON/DIRECT/DIRECT
   INTEGER DIRECT(66)
   DIMENSION C2(5),C3(4)
   EQUIVALENCE(DIRECT(57),C1),(DIRECT(58),C2(1)),(DIRECT(63),C3(1))
```

Here is the corresponding portion of the BLOCK DATA subprogram, which initializes the control elements of array DIRECT that were generated by the PLEX processor.

```
 COMMON/DIRECT/DIRECT
 INTEGER DIRECT(66)
 DATA DIRECT(1),DIRECT(2),DIRECT(3),DIRECT(4),DIRECT(5),DIRECT(6),
-DIRECT(7),DIRECT(8),DIRECT(9),DIRECT(10),DIRECT(11),DIRECT(12),
-DIRECT(13),DIRECT(14)/4HDIRE,4HCT  ,6*4H     ,40,2*0,3,0,134217728/
 DATA DIRECT(15),DIRECT(16),DIRECT(17),DIRECT(18),DIRECT(19),
-DIRECT(20),DIRECT(21),DIRECT(22),DIRECT(23),DIRECT(24),DIRECT(25),
-DIRECT(26),DIRECT(27),DIRECT(28)/4HC1  ,7*4H     ,168,4*0,524292/
 DATA DIRECT(29),DIRECT(30),DIRECT(31),DIRECT(32),DIRECT(33),
-DIRECT(34),DIRECT(35),DIRECT(36),DIRECT(37),DIRECT(38),DIRECT(39),
-DIRECT(40),DIRECT(41),DIRECT(42)/4HC2  ,7*4H     ,116,4*0,524308/
 DATA DIRECT(43),DIRECT(44),DIRECT(45),DIRECT(46),DIRECT(47),
-DIRECT(48),DIRECT(49),DIRECT(50),DIRECT(51),DIRECT(52),DIRECT(53),
-DIRECT(54),DIRECT(55),DIRECT(56)/4HC3  ,7*4H     ,80,4*0,524304/
```

**Example 2**

In this sample declaration of subschema record TIMTOT, each item is explicitly given a type. In addition, each item is equated with a C-number or component name.

- LASTNM, FORENM, and HIRE are INTEGER.
- VAC and SICKLV are REAL.
- OVRTIM is DOUBLE (DOUBLE PRECISION).

Subschema item HIRE corresponds to the DATE item HIRE DATE. It is declared as INTEGER, and two words must be reserved. The date format is YYMMDD.

```
 SUBSCHEMA/TIMTOT  OF EMPLOYEE/
-    LASTNM(I(6)) EQ C1,
-    FORENM(I(6)) EQ C2,
-    HIRE(I(2)) EQ HIRE DATE FORM YYMMDD,
-    VAC(R) EQ ACCRUED VACATION,
-    SICKLV(R) EQ ACCRUED SICK LEAVE,
-    OVRTIM(D) EQ OVERTIME HOURS
```

In the expansion from the PLEX processor, all lines of the program's declaration were turned into comments. Appropriate COMMON, type, DIMENSION, and EQUIVALENCE statements were substituted.

```
C     SUBSCHEMA/TIMTOT OF EMPLOYEE/
C    -    LASTNM(I(6)) EQ C1,
C    -    FORENM(I(6)) EQ C2,
C    -    HIRE(I(2)) EQ HIRE DATE FORM YYMMDD,
C    -    VAC(R) EQ ACCRUED VACATION,
C    -    SICKLV(R) EQ ACCRUED SICK LEAVE,
C    -    OVRTIM(D) EQ OVERTIME HOURS
      COMMON/TIMTOT/TIMTOT
      INTEGER TIMTOT(116)
      INTEGER LASTNM,FORENM,HIRE
      REAL VAC,SICKLV
      DOUBLE PRECISION OVRTIM
      DIMENSION LASTNM(6),FOREMN(6),HIRE(2)
      EQUIVALENCE(TIMTOT(99),LASTNM(1)),(TIMTOT(105),FORENM(1)),
     -(TIMTOT(111),HIRE(1)),(TIMTOT(113),VAC),(TIMTOT(114),SICKLV),
     -(TIMTOT(115),OVRTIM)
```

**Example 2** (continued)

The corresponding portion of the BLOCK DATA subprogram includes initializations for additional control elements of array TIMTOT that were generated by the PLEX processor.

```
 COMMON/TIMTOT/TIMTOT
 INTEGER TIMTOT(116)
 DATA TIMTOT(1),TIMTOT(2),TIMTOT(3),TIMTOT(4),TIMTOT(5),TIMTOT(6),
-TIMTOT(7),TIMTOT(8),TIMTOT(9),TIMTOT(10),TIMTOT(11),TIMTOT(12),
-TIMTOT(13),TIMTOT(14)/4HTIMT,4HOT  ,6*4H    ,72,2*0,6,0,134217728/
 DATA TIMTOT(15),TIMTOT(16),TIMTOT(17),TIMTOT(18),TIMTOT(19),
-TIMTOT(20),TIMTOT(21),TIMTOT(22),TIMTOT(23),TIMTOT(24),TIMTOT(25),
-TIMTOT(26),TIMTOT(27),TIMTOT(28)/4HC1  ,7*4H    ,336,4*0,524312/
 DATA TIMTOT(29),TIMTOT(30),TIMTOT(31),TIMTOT(32),TIMTOT(33),
-TIMTOT(34),TIMTOT(35),TIMTOT(36),TIMTOT(37),TIMTOT(38),TIMTOT(39),
-TIMTOT(40),TIMTOT(41),TIMTOT(42)/4HC2  ,7*4H    ,304,4*0,524312/
 DATA TIMTOT(43),TIMTOT(44),TIMTOT(45),TIMTOT(46),TIMTOT(47),
-TIMTOT(48),TIMTOT(49),TIMTOT(50),TIMTOT(51),TIMTOT(52),TIMTOT(53),
-TIMTOT(54),TIMTOT(55),TIMTOT(56)/4HHIRE,4H DAT,4HE   ,5*4H    ,
-272,4*0,1048584/
 DATA TIMTOT(57),TIMTOT(58),TIMTOT(59),TIMTOT(60),TIMTOT(61),
-TIMTOT(62),TIMTOT(63),TIMTOT(64),TIMTOT(65),TIMTOT(66),TIMTOT(67),
-TIMTOT(68),TIMTOT(69),TIMTOT(70)/4HACCR,4HUED ,4HVACA,4HTION,
-4*4H    ,224,4*0,524292/
 DATA TIMTOT(71),TIMTOT(72),TIMTOT(73),TIMTOT(74),TIMTOT(75),
-TIMTOT(76),TIMTOT(77),TIMTOT(78),TIMTOT(79),TIMTOT(80),TIMTOT(81),
-TIMTOT(82),TIMTOT(83),TIMTOT(84)/4HACCR,4HUED ,4HSICK,4H LEA,
-4HVE  ,3*4H    ,172,4*0,524292/
 DATA TIMTOT(85),TIMTOT(86),TIMTOT(87),TIMTOT(88),TIMTOT(89),
-TIMTOT(90),TIMTOT(91),TIMTOT(92),TIMTOT(93),TIMTOT(94),TIMTOT(95),
-TIMTOT(96),TIMTOT(97),TIMTOT(98)/4HOVER,4HTIME,4H HOU,4HRS  ,
-4*4H    ,120,4*0,524296/
```

**Example 3**

This sample LISTNG subschema record includes the INTERNAL option. All items are type INTEGER. One word is reserved for EMPNUM, five for LNAME, three for OFFICE, and five for FORNAM.

```
 IMPLICIT INTEGER(A-W) . . .
 INTERNAL SSR/LISTNG OF EMPLOYEE/ EMPNUM EQ EMPLOYEE NUMBER,
-LNAME(5) EQ LAST NAME, OFFICE(3) EQ OFFICE-EXTENSION,
-FORNAM(5) EQ FORENAME
```

In the expansion by the PLEX processor, no COMMON statement is generated since this SSR will be known only to the subroutine that contains it.

In addition, DATA statements are included in the generated code, rather than in a BLOCK DATA subprogram, since no labeled common block was defined for the SSR.

```
   INTERNAL SSR/LISTNG OF EMPLOYEE/ EMPNUM EQ EMPLOYEE  NUMBER
C -LNAME(5) EQ LAST NAME, OFFICE(3) EQ OFFICE-EXTENSION,
C -FORNAM(5) EQ FORENAME
   INTEGER LISTNG(84)
   DIMENSION LNAME(5),OFFICE(3),FORNAM(5)
   EQUIVALENCE(LISTNG(71),EMPNUM),(LISTNG(72),LNAME(1)),(LISTNG(77),
  -OFFICE(1)),(LISTNG(80),FORNAM(1))
   DATA LISTNG(1),LISTNG(2),LISTNG(3),LISTNG(4),LISTNG(5),LISTNG(6),
  -LISTNG(7),LISTNG(8),LISTNG(9),LISTNG(10),LISTNG(11),LISTNG(12),
  -LISTNG(13),LISTNG(14)/4HLIST,4HNG  ,6*4H    ,56,2*0,4,0,134217728/
   DATA LISTNG(15),LISTNG(16),LISTNG(17),LISTNG(18),LISTNG(19),
  -LISTNG(20),LISTNG(21),LISTNG(22),LISTNG(23),LISTNG(24),LISTNG(25),
  -LISTNG(26),LISTNG(27),LISTNG(28)/4HEMPL,4HOYEE,4H NUM,4HBER ,
  -4*4H    ,224,4*0,524292/
   DATA LISTNG(29),LISTNG(30),LISTNG(31),LISTNG(32),LISTNG(33),
  -LISTNG(34),LISTNG(35),LISTNG(36),LISTNG(37),LISTNG(38),LISTNG(39),
  -LISTNG(40),LISTNG(41),LISTNG(42)/4HLAST,4H NAM,4HE   ,5*4H    ,
  -172,4*0,524308/
   DATA LISTNG(43),LISTNG(44),LISTNG(45),LISTNG(46),LISTNG(47),
  -LISTNG(48),LISTNG(49),LISTNG(50),LISTNG(51),LISTNG(52),LISTNG(53),
  -LISTNG(54),LISTNG(55),LISTNG(56)/4HOFFI,4HCE-E,4HXTEN,4HSION,
  -4*4H    ,136,4*0,524300/
   DATA LISTNG(57),LISTNG(58),LISTNG(59),LISTNG(60),LISTNG(61),
  -LISTNG(62),LISTNG(63),LISTNG(64),LISTNG(65),LISTNG(66),LISTNG(67),
  -LISTNG(68),LISTNG(69),LISTNG(70)/4HFORE,4HNAME,6*4H    ,92,4*0,
  -524308/
```

### Example 4

Here is a sample declaration of the LISTNG subschema record with the PARM option. This option is used for an SSR that appears as a parameter in a called subroutine. In this case, the calling routine is passing an INTERNAL SSR as a parameter.

```
 PARM SSR      /LISTNG OF EMPLOYEE/ EMPNUM EQ EMPLOYEE  NUMBER,
-LNAME(5) EQ LAST NAME, OFFICE(3) EQ OFFICE-EXTENSION,
-FORNAM(5) EQ FORENAME
```

As in Example 3, the PLEX processor generates an INTEGER statement for an array named LISTNG. But in this case, comments are included noting the fields of the array that correspond to the subschema items.

```
C   PARM SSR      /LISTNG OF EMPLOYEE/ EMPNUM EQ EMPLOYEE NUMBER,
C  -LNAME(5) EQ LAST NAME, OFFICE(3) EQ OFFICE-EXTENSION,
C  -FORNAM(5) EQ FORENAME
    INTEGER LISTNG(84)
C   ,(LISTNG(71),EMPNUM),(LISTNG(72),LNAME(1)),(LISTNG(77),OFFICE(1)),
C  -(LISTNG(80),FORNAM(1))
```

### Example 5

A subschema record can appear without items, regardless of whether the corresponding schema record has items. The name of such a subschema record must be either the component name or the C-number of the schema record. In this example, the subschema record name is the C-number C400.

```
SUBSCHEMA RECORD/C400 OF EMPLOYEE/
```

In the expansion by the PLEX processor, a COMMON statement and an INTEGER statement were generated.

```
C    SUBSCHEMA RECORD/C400 OF EMPLOYEE/
     COMMON/C400/C400
     INTEGER C400(14)
```

The corresponding portion of the BLOCK DATA subprogram includes initialized control fields of array C400 that were generated by the PLEX processor.

```
 COMMON/C400/C400
 INTEGER C400(14)
 DATA C400(1),C400(2),C400(3),C400(4),C400(5),C400(6),C400(7),
-C400(8),C400(9),C400(10),C400(11),C400(12),C400(13),
-C400(14)/4HC400,7*4H     ,5*0,134217728/
```

**Example 6**

This example shows the use of type CHARACTER for subschema items. The input SUBSCHEMA RECORD declaration is shown followed by the expanded source generated by the PLEX processor.

Input SSR declaration:

```
 SSR/ ENTRY OF EMPLOYEE/
-     FNAME(C)          EQ C3,
-     LNAME(C(16))      EQ C2,
-     STREET(I(8))      EQ C14,
-     CITYST(I(5))      EQ C15,
-     ZIP(I(2))         EQ C16
```

Expanded source:

```
C     SSR/ ENTRY OF EMPLOYEE/
C    -     FNAME(C)          EQ C3,
C    -     LNAME(C(16))      EQ C2,
C    -     STREET(I(8))      EQ C14,
C    -     CITYST(I(5))      EQ C15,
C    -     ZIP(I(2))         EQ C16
      COMMON/ENTRY/ENTRY
      INTEGER ENTRY(104)
      INTEGER STREET,CITYST,ZIP
      DIMENSION STREET(8),CITYST(5),ZIP(2)
      CHARACTER C3*1,LNAME*16
      EQUIVALENCE(ENTRY(85),C3),(ENTRY(86),LNAME),(ENTRY(90),
     -STREET(1)),(ENTRY(98),CITYST(1)),(ENTRY(103),ZIP(1))
```

## Considerations for Subschema Records

- A subschema record declaration can appear wherever FORTRAN specification statements are allowed in a program, as long as the COMMBLOCK associated with the subschema record is declared before the subschema record.

- The SSR name can be different from the schema record name if the SSR has at least one item in it. SYSTEM 2000 software associates an SSR containing items with the corresponding schema record by associating item names, rather than by matching record names.

- You can declare a subschema record without any items in it. (See Example 5.) The syntax of such a subschema record is as follows:

  ```
  |[INTERNAL] SUBSCHEMA [RECORD] / name OF database-identifier/
  |[PARM]
  ```

  An SSR without items can be used when there is no item in the schema record or when retrieval of the data values corresponding to the items is not required. Such a subschema record can be used in a where-clause or for address retrieval before an update. Because the SSR has no items, its name must be either the component name or C-number of the corresponding schema record. In this case, SYSTEM 2000 software associates the two records by matching their names. If the name of such an SSR is not the component name or C-number of the corresponding schema record, any command referring to the SSR results in return code 13. Also, the EQ option can be used to make the schema record name match the SSR name.

- Your program can have more than one SSR corresponding to the same schema record. The only requirement for these SSRs is that they have unique names (not necessarily the name of the corresponding schema record). If the name of an SSR is not unique within your program, an error message occurs.

- If the main program or subroutine in which the SSR is declared does not include the corresponding COMMBLOCK, the PLEX processor gives an error message for the subschema record declaration. This consideration applies to subschema records declared with or without the INTERNAL or PARM option.

- If you do not specify the INTERNAL or the PARM option for an SSR, the PLEX processor generates a labeled COMMON statement for the SSR. This means that an SSR and its fields can be referenced by a main program and one or more subroutines, just as any labeled common storage areas can be referenced.

- Every main program and subroutine with one or more PLEX commands must contain the $BLOCKS directive. In addition, each main program and subroutine accessing a database must reference the same COMMBLOCK and subschema record(s) declared for the database, by passing or receiving them as parameters or by referencing the same area in common storage.

- Subschema records can be placed in the source library and copied into your program during PLEX processing using the $COPY directive. (See **$COPY Directive** on page C-37.)

---

Abbreviations: SUBSCHEMA = SSR

## Considerations for Items in Subschema Records

- If the alias of a subschema item is different from the corresponding schema item, you must specify the EQ option to establish the correspondence between that alias and the component name or C-number of the schema item. In this situation, if you do not use the EQ option, return code 13 results from each command referring to this subschema record.

- The FORTRAN type of a subschema item must be compatible with the item type of the corresponding schema item. The PLEX processor does not check the validity of the FORTRAN types declared. If the declarations are not valid in FORTRAN, errors occur during compilation. Incompatible item types result in run-time errors.

- When a subschema record is first referenced at execution time, all the items of the subschema record are validated. This is called "the opening of a subschema record." If the FORTRAN type of a subschema item is not compatible with the item type, picture, or size of the corresponding item in the database, return code 17 is placed in the COMMBLOCK for that subschema record. The ordinal of that item within its subschema record is placed in the eighth field of the COMMBLOCK.

- The names of items used in a PLEX command must appear exactly as they do in the subschema item declarations. The PLEX processor issues an error message if a command includes an alias that does not appear in a subschema record declaration.

  For C-numbers declared as aliases for subschema items, SYSTEM 2000 software allows some flexibility that FORTRAN and, therefore, the PLEX processor do not. For example, if you declare the alias C010 for a subschema item corresponding to C10 database item, your PLEX program does not need to include the number C10 with EQ in the item declaration. SYSTEM 2000 software understands the implicit equivalence at execution time. When a PLEX command includes a C-number such as C010, SYSTEM 2000 software knows that it refers to component C10 in the database. If the number C010 is declared as the alias, this number must be used in a PLEX command referring to that item. In this case, if C10 is used instead, the PLEX processor issues an error message, since the alias is actually C010 in the subschema item declaration.

  A C-number declared as an alias can have up to four digits, besides C. Thus, the component whose C-number is C1 can be declared in a subschema record with the alias of C1, C01, C001, or C0001.

- If the subschema record refers twice to the same database item by specifying the same alias, component name, or C-number twice, the PLEX processor issues an error message. This error can occur in the following three situations:

  - The same alias is declared twice in the same SSR.

  - The SSR includes an EQ option specifying a component name or C-number also declared as the alias for another item in the same SSR.

  - Two EQ specifications in the SSR specify the same component name or C-number.

- Return code 13 results from a command for a subschema record referring to the same database item if the component name is specified in one item declaration and the associated C-number is specified in another item declaration. The same error results if different forms of the same C-number appear in different item declarations in the same SSR. This error is not detected in preprocessing, because the PLEX processor

does not access the database definition and therefore does not detect such a double reference. This error can occur within a subschema record in the following four situations:

- The component name and C-number are both used as item aliases.
- The component name or C-number is used as an alias, while its counterpart is named with the EQ option in another item declaration.
- The component name and C-number are named with the EQ option in two different item declarations.
- The C-number C1, for example, is specified in one item declaration, while a variant such as C001 is specified in another item declaration.

- Return code 13 results from a command for a subschema record referring (by C-number or component name) to an item not in the database schema.

- A subschema item declared with type DOUBLE will be given the FORTRAN type DOUBLE PRECISION and will be aligned on a double-word boundary. To ensure that DOUBLE PRECISION items can be stored at double-word boundaries, the labeled common storage block for the SSR must start at a double-word boundary. This is taken care of by the PLEX processor if the SSR is not declared with the INTERNAL or PARM option. However, if you declare the SSR INTERNAL or PARM, your program must ensure that the labeled common storage block for the SSR starts on a double-word boundary.

- A subschema item corresponding to a DATE type item in a database can be declared INTEGER(2) or type CHARACTER with six or eight bytes. If not, return code 17 results from a command referring to the subschema record, and the alias of the unacceptable item appears in the first field of the COMMBLOCK.

  If such a subschema item is correctly declared and a six-character date format is specified with the FORM option, a six-digit value is retrieved. Similarly, if an eight-character date format is specified, an eight-digit value is retrieved. For a where-clause or an update, if the length of the date format and the number of digits placed in the field do not correspond, return code 17 results.

- If both EQ and FORM are specified in a subschema item declaration for a DATE type database item, the EQ specification must precede the FORM specification.

- When more than one word will be used for an INTEGER variable in the SSR, you must specify the size, so that the PLEX processor can generate an appropriate DIMENSION statement for the variable. If *size* is not specified for such a variable, the size is one word. Similarly, the default size for variable type CHARACTER is one byte. If more than one byte will be used, you must specify the number of bytes, for example, C(15).

  **Note**: The PLEX processor ignores any separate, user-coded DIMENSION statement for the variable.

- Up to 1000 items can be declared in an SSR, provided that all of these belong to the same schema record. If your program declares more than 1000 items in an SSR, the PLEX processor issues an error message.

# FORTRAN PLEX PROCESSOR DIRECTIVES

The PLEX processor directives are coded like PLEX commands in the source program. These instructions serve as signals to the PLEX processor. This section discusses each of the PLEX processor directives.

## $BLOCKS Directive

### Format

```
$BLOCKS[.]
```

Each FORTRAN PLEX program must include the $BLOCKS directive before any executable statements. In response to $BLOCKS, the PLEX processor generates declarations of control blocks for retrieval or update commands that contain item lists or that require special information to be passed to SYSTEM 2000 software during execution. Following those control blocks, the PLEX processor generates the declaration of S2KDUM. It generates initial values for S2KDUM and the other control blocks in the BLOCK DATA subprogram. (See Illus. C.2 below and Illus. C.3 on page C-35.)

**Illus. C.2** Sample S2KDUM Generated by the PLEX Processor for $BLOCKS

```
$BLOCKS
 INTEGER S2F1(9)
 DATA S2F1/0,4*0,2147483647,3*-1/
 INTEGER S2R1(2)
 DATA S2R1/2,11/
 INTEGER S2R2(3)
 DATA S2R2/3,2*1/
 COMMON/S2KDUM/S2KDUM(38)
 INTEGER S2KDUM,S2KDU1,S2KRTC,S2KCNT,S2KDU2,S2KDU3,S2KDU4,S2KDU5,
-S2KDU6,S2KDU7,S2KDU8,S2KDU9,S2KSRT,S2KLKH(10)
 EQUIVALENCE(S2KDUM(1),S2KDU1),(S2KDUM(2),S2KRTC),(S2KDUM(3),
-S2KCNT),(S2KDUM(4),S2KDU2),(S2KDUM(5),S2KDU3),(S2KDUM(7),S2KDU4),
-(S2KDUM(17),S2KDU5),(S2KDUM(18),S2KDU6),(S2KDUM(19),S2KDU7),
-(S2KDUM(23),S2KDU8),(S2KDUM(24),S2KDU9),(S2KDUM(25),S2KSRT),
-(S2KDUM(29),S2KLKH(1))
```

**Illus. C.3** Description of S2KDUM Control Block Items

| Offset HEX | Offset DEC | Name | Length/ Alignment | Type/ Value | Field Description |
|---|---|---|---|---|---|
| 0 | 0 | S2KDU1 | 4 | Binary | SYSTEM 2000 operation code; set by the expanded PLEX code at execution time. |
| 4 | 4 | S2KRTC | 4 | Binary | SYSTEM 2000 return code; set by a PLEX command execution. |
| 8 | 8 | S2KCOUNT | 4 | Binary | Value of S2KCOUNT; set by the PLEX program. |
| C | 12 | S2KDU2 | 4 | Binary | Stack or Locate File number; set by the expanded code at run time. |
| 10 | 16 | S2KDU3 | 4 | Binary | Maximum number for Locate File; set at run time by the expanded START S2K code. Also, the number of the target stack or Locate File referred to in a COPYS or COPYL command; set at run time. |
| 14 | 20 | FILLER | 4 | Binary<br>1<br>2<br>3<br>4<br>5 | PLEX program type; set by PLEX processor.<br>FORTRAN<br>COBOL<br>PL/I - optimizing or checkout<br>PL/I (F)<br>ASSEMBLER |
| 18 | 24 | S2KDU4 | 4 | Binary<br>0<br>1 | SYSTEM 2000 Multi-User indicator; set at run time by the expanded START S2K code.<br>Single-user<br>Multi-User |
| 1C | 28 | FILLER | 4 | Alpha-numeric | If S2KDU4 is neither fullword binary zero or one, then S2KDU4 concatenated with this field contains the name of a special module to be used (left-justified and blank-filled). |
| 20 | 32 | FILLER | 8 | Alpha-numeric | Name of the program (left-justified and blank-filled); set by the PLEX processor as specified in the $NAME directive. |
| 28 | 40 | FILLER | 4 | Alpha-numeric | Release of the SYSTEM 2000 PLEX processor used; set by the PLEX processor. If changes are made to the Program Service Processor, this field identifies the PLEX processor release that is used. |
| 2C | 44 | FILLER | 8 | Alpha-numeric | Date of the PLEX processor run (YYDDD, left-justified and blank-filled); set by the PLEX processor. |
| 34 | 52 | FILLER | 8 | Alpha-numeric | Time of PLEX processor run (HHMMSSS, left-justified and blank-filled); set by the PLEX processor. |
| 3C | 60 | FILLER | 4 | Alpha-numeric | SYSTEM 2000 release number; set at execution time by SYSTEM 2000 software. |

*continued on next page*

**Illus. C.3** *continued*

| Offset HEX | Offset DEC | Name | Length/ Alignment | Type/ Value | Field Description |
|---|---|---|---|---|---|
| 40 | 64 | S2KDU5 | 4 | Binary | Reserved for internal use. |
| 44 | 68 | S2KDU6 | 4 | Binary | Reserved for internal use. |
| 48 | 72 | S2KDU7 | 4 | Alpha | Reserved for internal use. |
| 4C | 76 | FILLER | 4 | Binary | Address of run-time interface (S2KPLR). |
| 50 | 80 | FILLER | 4 | Binary | Address of SYSTEM 2000 software (S2KPLI, MUPLINT). |
| 54 | 84 | FILLER | 4 | Binary | Address of working storage; set by run-time interface. This address may be used by installation run-time exit routines S2KEX1 or S2KEX2. |
| 58 | 88 | S2KDU8 | 4 | Binary<br>0<br>1 | Snap request indicator; set by expanded code.<br>Off<br>On: SYSTEM 2000 software will provide snapshot data. |
| 5 | 92 | S2KDU9 | 4 | Binary<br>0<br>1 | Timing request indicator; set by the expanded code.<br>Off<br>On |
| 60 | 96 | S2KSRT | 4 | Binary | No-sort request indicator; set by the PLEX program.<br>0 = Sort normally.<br>1 = Do not sort. |
| 64 | 100 | | 4 | Binary | Reserved for internal use. |
| 68 | 104 | | 4 | Binary | Reserved for internal use. |
| 6C | 108 | | 1 | Bit<br>.... ...0<br>.... ...1 | Switches used at run time.<br>Link history was not modified.<br>Link history was modified. |
| 6D | 109 | | 3 | | Reserved for internal use. |
| 70 | 112 | S2KLKH | 40 | Binary | Ten full words representing LINK0 thru LINK9 return codes; set by linked retrievals. Initialized by the PLEX processor to ten words of binary minus one. |

## $COPY Directive

### Format

```
$COPY  member [.]
```

*member* is the name of a member of the SYSLIB copy library.

The $COPY directive instructs the PLEX processor to copy source statements from a member of a prepared library into the PLEX program where it appears. The $COPY command specifies the member of the SYSLIB that provides the statements, and the placement of $COPY indicates where the copied statements go. Sample JCL required for using a $COPY directive appears in **$COPY Library** on page C-67.

You must specify the COPY execution option in the JCL for a job that contains a program that issues the $COPY directive. Only entire commands, COMMBLOCKs, or subschema records can be copied. For example, you cannot copy a where-clause or a portion of a subschema record by itself.

The PLEX processor supports up to ten levels of $COPY nesting. If you include a $COPY directive in a member that is copied, the PLEX processor obtains the additional commands from the indicated member. If you concatenate $COPY files, the first file must have the largest block size.

Special Zap #124 is available to require that all subschema record declarations be copied from the $COPY library. Contact your SYSTEM 2000 Software Representative.

## END PROCEDURE Directive

### Format

```
*PL  END PROCEDURE[.]
```

(Notice that the END PROCEDURE directive has no initial dollar sign.)

If you use the FORM execution option, issue END PROCEDURE to separate two programs (such as a main program and a subroutine) that are submitted together but that will be processed individually by the PLEX processor. This directive causes the PLEX processor to reinitialize its internal settings. If END PROCEDURE is not used in this case, the PLEX processor issues error messages if the $BLOCKS directive appears twice or if two COMMBLOCKs, subschema records, or subschema items have the same name. If only one program is submitted to the PLEX processor, END PROCEDURE can be omitted.

If you use the NOFORM execution option, omit END PROCEDURE between two programs, since the END statement of the first program serves the same function for the PLEX processor. NOFORM is the default.

## $NAME Directive

### Format

```
$NAME name [.]
```

*name* is a string of up to eight characters.

The $NAME directive instructs the PLEX processor to initialize a field in the S2KDUM control block to the specified name. Also, this name is printed at the top of each page of output from the PLEX processor. The PROGRAM directive is the functional equivalent of $NAME.

## PROGRAM Directive

### Format

```
PROGRAM name[.]
```

*name* is a string of up to eight characters.

The PROGRAM directive instructs the PLEX processor to initialize a field in the S2KDUM control block to the specified name. Also, this name is printed at the top of each page of output from the PLEX processor. The $NAME directive is the functional equivalent of the PROGRAM directive.

# FORTRAN PLEX PROCESSOR EXECUTION OPTIONS

You set the PLEX processor execution options through the PARM parameter on the EXEC statement. For example, if the name of the job step for the PLEX processor is PRECOM, the parameter list for the job step appears as follows:

```
//PRECOM  EXEC   PGM=PRCOMF,PARM= 'parameter-list'
```

Illus. C.7 on page C-64 shows a complete example of JCL for a single-user PLEX program.

## COPY and NOCOPY Execution Options

### Format

```
|COPY
|NOCOPY    (default)
```

If you issue a $COPY directive in a PLEX program (see **$COPY Directive** on page C-37), you must specify the COPY execution option. If this execution option is not specified or if the NOCOPY execution option is specified, the PLEX processor issues an error message if it encounters a $COPY directive in the program.

The COPY execution option does not force the use of $COPY. However, Special Zap #124 is available that forces all subschema record declarations to be copied from the $COPY library. Contact your SYSTEM 2000 Software Representative.

## DECK and NODECK Execution Options

### Format

```
|DECK
|NODECK    (default)
```

The DECK execution option (default) instructs the PLEX processor to punch the expanded source on SYSPUNCH.

## ECHO and NOECHO Execution Options

**Format**

```
|NOECHO
|ECHO    (default)
```

The ECHO execution option (default) specifies that source PLEX commands and PLEX processor directives should be included as comments in the input (on SYSPUNCH and SYSGO) to the FORTRAN compiler. NOECHO specifies that source commands and directives should be omitted from the input to the FORTRAN compiler.

## ERRORS and NOERRORS Execution Options

**Format**

```
|NOERRORS
|ERRORS    (default)
```

The ERRORS execution option causes diagnostic messages to be printed together on SYSPRINT following other output from the PLEX processor. NOERRORS suppresses printing of those diagnostic messages at the end of the PLEX processor output, but not within the body of the source listing. Either option allows other messages to appear at the end of PLEX processor output on SYSPRINT. These other messages give the total number of diagnostics issued for the PLEX program, the highest severity level of those diagnostics, and a memory utilization report.

## ERRSYM Execution Option

**Format**

```
ERRSYM = character
```

*character* is a single alphanumeric or special character. The default is the period.

The ERRSYM execution option designates the character that fills blank columns preceding the cursor 'C' printed above certain PLEX processor error messages such as message 406: DATA IDENTIFIER NOT FOUND.

## EXIT Execution Option

### Format

```
EXIT = load-module
```

*load-module* is the name of a load module (not link-edited with PLEX processor).

The EXIT execution option names a user-written exit routine for the PLEX processor that edits source input (including statements input by a $COPY directive) before processing by the PLEX processor. The routine must reside in JOBLIB, STEPLIB, or the system link library.

As the PLEX processor receives each input record, it calls the specified exit routine and passes the following three parameters:

- The address of the current source record as it stands before processing by the PLEX processor. (This is also the address used if you want to add records.)
- The address of a fullword value indicating the current level of $COPY nesting. A value of binary zero means that the current record was not generated for a $COPY directive (see **$COPY Directive** on page C-37). A value of binary 1 signifies that a $COPY directive in the original source code caused the record to be read from a member of the copy library. Higher numbers indicate the current value of nested $COPY logic, up to a maximum of 10.
- The address of a fullword value of the action code, which will be set by the exit routine to indicate the disposition of the current record processed by the exit routine. The action code is initialized to zero by the PLEX processor before each call to the exit routine. The exit routine then sets the action code before passing control back to the PLEX processor, as appropriate:

  0 means the PLEX processor should accept the current record as passed back (and possibly modified) by the exit routine.

  1 means the PLEX processor should delete the current record.

  2 means the PLEX processor should accept the current record as passed back (and possibly modified) by the exit routine and should allow another record to be added on the next call to the exit routine.

  >2 is the address of the diagnostic message that the PLEX processor will write to output after the current record.

**Modifying, replacing, and adding records** The exit routine can modify or replace the current record in the source input by receiving it in a call from the PLEX processor, making the appropriate substitution in the field containing the received record, then sending it back to the PLEX processor. If the action code from the exit routine is not 1 (delete record), the PLEX processor accepts what it receives as the current record.

If the action code is 2, the PLEX processor accepts and processes the current record from the exit routine, then immediately makes another call to the exit routine, without taking another record from the source input. The exit routine then supplies a record to be added to the PLEX program, sets the action code, and returns control to the PLEX processor. If the exit routine sets the action code to 2 again this time, the PLEX processor accepts the new record and again calls the exit routine. In this way, one or more records can be added to the PLEX program through multiple calls to the exit routine.

To discontinue this sequence for adding a group of records, the exit routine must set the action code to 0 before it passes the last new record. Action code 0 instructs the PLEX processor to accept the last new record and to read another record from the source input before calling the exit routine again.

**Diagnostic messages** The format for the standard PLEX processor diagnostics is also used for printing diagnostic messages issued by an exit routine:

| col. 1 | col. 10 | col. 17 |
|---|---|---|
| **PLX** *5 digits* | **PLX** *3 digits* | *message-text* |
| **statement sequence number** | **message number** | |

The PLEX processor sets the statement sequence number automatically. The exit routine determines the message number and message text and passes both to the PLEX processor for printing. The message text must be in two forms: the long form, occupying 16 words, followed by the short form, occupying five words. The PLEX processor prints the long form of the message when the NOTERM (default) execution option is in effect, and the short form when the TERM execution option was specified. (See **TERM and NOTERM Execution Options** on page C-49.)

In addition, the exit routine must determine both the severity level of the diagnostic and the column in which the cursor for the diagnostic message should appear.

The field containing the specifications discussed above for a diagnostic message must be arranged as follows:

| Word(s) | Contents |
|---|---|
| 1 | is the fullword value of the column number in which the error in the current record was detected; determines positioning of the PLEX processor error cursor, which is printed on a line between the current record and the diagnostic message text. If the column number is less than 4, the cursor is not printed. |
| 2 | is the fullword value of the severity level of the error. |
| 3 | is a message number of up to four digits in EBCDIC character representation, left-justified and blank-filled. If the message number is four characters long, no space appears between the message number and the message text when the diagnostic message is displayed. |
| 4-19 | is the long-form message text (64 characters); used if the NOTERM (default) execution option is in effect. |
| 20-24 | is the short-form message text (20 characters); used if the TERM execution option is in effect. |

**Notes**

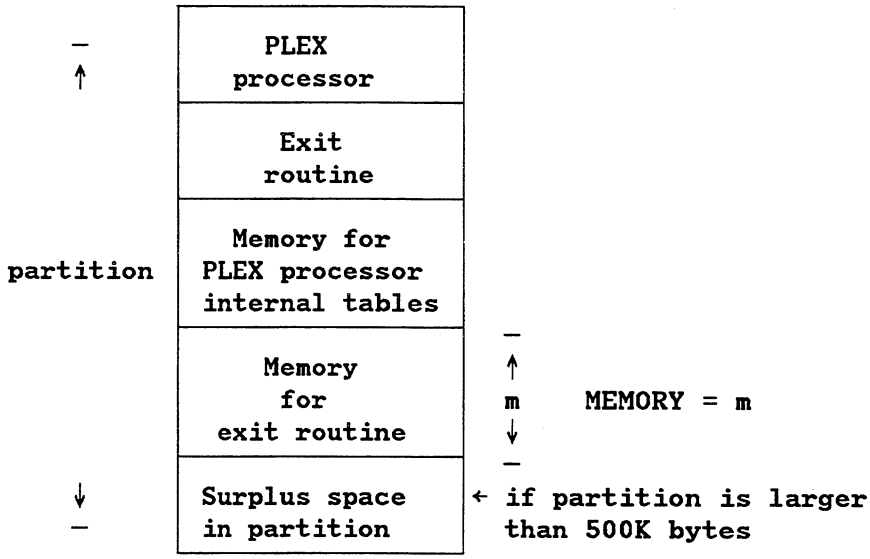
- As an alternative to using a customized exit routine through the EXIT execution option, a standard exit subroutine named S2KEXIT and developed by systems personnel at the site can be link-edited with the PLEX processor for editing source input from all jobs using the PLEX processor. This exit must be named SUBROUTINE S2KEXIT (or have the CSECT name S2KEXIT if written in Assembler) and must be included as object input during a re-linking of the PLEX processor.

  The PLEX processor automatically checks whether an exit routine named S2KEXIT was link-edited with the PLEX processor. If this routine was link-edited with the PLEX processor, the EXIT execution option should not be specified without approval from the systems personnel at the site, because it overrides use of the standard routine.

- An exit routine can issue up to five diagnostic messages. If more than five messages are issued, PLEX processor diagnostic message 100 is issued.

- An exit routine cannot issue a message for a comment card or a continuation card in the source to the PLEX processor. An attempt to do so causes unpredictable results.

- If the exit routine obtains additional memory with the GETMAIN macro, the amount to be obtained must be specified previously with the MEMORY execution option (see **MEMORY Execution Option** on page C-47). With a GETMAIN, the PLEX processor obtains all available space in its partition (up to 500K bytes). Then it reserves the specified memory space for the exit routine and calculates the amount of memory remaining for its internal tables. The resulting arrangement is as shown below.




Sufficient memory must be available in the partition for PLEX processor internal tables for COMMBLOCKs, SSRs, and subschema items declared in the PLEX program. The amount of memory required for each of these internal tables is as follows:

**COMMBLOCK table**

**24 bytes for the table control block**
**plus**
**72 bytes per entry (one entry for each COMMBLOCK in the program)**

**Subschema record table**

**24 bytes for the table control block**
**plus**
**52 bytes per entry (one entry for each subschema in the program)**

**Subschema item table**

**24 bytes for the table control block**
**plus**
**80 bytes per entry (one entry for each subschema item in the program)**

## FORM and NOFORM Execution Options

**Format**

```
|FORM
|NOFORM    (default)
```

If you specify the FORM execution option, the PLEX processor recognizes PLEX commands and directives only if they begin with the characters *PL in columns 1-3 by default or in other columns as set through the SORMGIN execution option (see **SORMGIN Execution Option** on page C-48). If you use the NOFORM execution option (default), PLEX commands can be free-form and can begin anywhere after column 6. **Format of PLEX Commands** on page C-3 discusses the format of PLEX commands under both execution options, as well as restrictions on keywords that can be used in the program.

FORM is primarily intended to provide upward compatibility with earlier SYSTEM 2000 software releases. The use of FORM is not encouraged for new programming efforts.

## GEN and NOGEN Execution Options

**Format**

```
|GEN
|NOGEN    (default)
```

The GEN execution option specifies the printing of the expanded source listing from the PLEX processor on SYSPRINT. NOGEN (default) suppresses the expanded source listing, except that it does not suppress the printing of the expanded COMMBLOCKs in source listings. NOGEN is the default.

## ITEM Execution Option

### Format

```
ITEM = nnnn
```

*nnnn* is the average number of subschema items per subschema record. The default is 5.

The ITEM execution option directs the memory allocation for processing subschema record declarations. The ITEM and SSR options together are used for optimizing memory utilization during the execution of the PLEX processor. See **Memory Utilization during the PLEX Processor Step** on page 22-16 for more information on the use of this execution option in memory utilization.

## LANG Execution Option

### Format

```
LANG = |FTN3
       |FTN2
       |FTN   (default)
```

The LANG execution option designates which FORTRAN compiler will be used. FTN (default) means VS FORTRAN Release 1.0 or Release 1.1. FTN2 means VS FORTRAN Release 2.0. FTN3 means VS FORTRAN Release 3.0.

## LINECNT Execution Option

### Format

```
LINECNT = nn
```

*nn* is the number of lines per page. The default is 55.

The LINECNT execution option sets the number of lines per page for SYPRINT output.

## LOAD and NOLOAD Execution Options

### Format

```
|NOLOAD
|LOAD    (default)
```

The LOAD execution option (default) specifies that the PLEX program, as expanded by the PLEX processor, will be sent to SYSGO for the compilation job step. NOLOAD specifies that no further processing of the PLEX program will be performed.

## MEMORY Execution Option

### Format

```
MEMORY = bytes
```

*bytes* is the number of bytes for main memory, up to four digits. The default is 4096.

The MEMORY execution option specifies the amount of main memory that should remain available in the region during the PLEX processor job step. If an exit routine is also specified (with the EXIT option) that dynamically obtains additional main memory, an appropriate value must be supplied for MEMORY. (See **EXIT Execution Option** on page C-41.)

## SORMGIN Execution Option

### Format

```
SORMGIN = (a,b,c,d)
```

- *a* is the column for the comment flag (C) generated by the PLEX processor.
- *b* is the continuation column for the code generated by the PLEX processor.
- *c* is the starting column for the scan by the PLEX processor.
- *d* is the sequence number starting position. If *d* is less than or equal to 73, the PLEX processor generates a sequence number for each statement in the source input, but not for statements generated for PLEX commands. If *d* is greater than 73, no sequence numbers are generated.

The SORMGIN execution option designates the margin positions on the source statements. The defaults for FORTRAN PLEX are as follows:

- *a* = 1
- *b* = 6
- *c* = 7
- *d* = 73

If you use the SORMGIN execution option, you must specify *a* through *d* explicitly. The specified values must be such that $1 < a < b < c < d < 80$. Otherwise, the PLEX processor issues an error message.

## SOURCE and NOSOURCE Execution Options

### Format

```
|NOSOURCE
|SOURCE    (default)
```

The SOURCE execution option (default) causes the source input, including source FORTRAN statements, to be listed on SYSPRINT. NOSOURCE suppresses the source listing. NOSOURCE implies the TERM option, which gives short-form messages.

## SSR Execution Option

### Format

```
SSR = nnnn
```

*nnnn* is the average number of subschema records per COMMBLOCK. The default is 4.

The SSR execution option directs memory allocation for processing subschema record declarations. This option optimizes memory utilization during the execution of the PLEX processor.

## SYSSEP Execution Option

### Format

```
SYSSEP = a
```

*a* is a single alphanumeric or special character. The default is the asterisk.

The SYSSEP execution option designates the system separator character for the ternary operators SPAN, SPANS, SPANNING, EQ, and NE, and for item-to-item comparisons in where-clause processing.

## TERM and NOTERM Execution Options

### Format

```
|TERM
|NOTERM    (default)
```

The TERM execution option directs the PLEX processor to issue abbreviated, short-form messages, which are suitable for jobs running in TP environments. The NOTERM execution option (default) specifies the long form for PLEX processor messages. See **Diagnostic messages** on page 19-14 for the short form and long form for all PLEX processor messages.

## TRANS Execution Option

### Format

---

**TRANS** = *load-module*

---

*load-module* is the name of a load module for a translate table routine (not link-edited with PLEX processor).

The TRANS execution option designates a table routine that translates all source input. This table is commonly used for translation of character sets, for example, EBCDIC to ASCII. Refer to the IBM *Principles of Operation* manual for a description of the TR table. The routine must be included in JOBLIB, STEPLIB, or the system link library.

An exit routine named S2KTRANS and developed by systems personnel at the site can be link-edited with the PLEX processor for translating source input from all jobs using the PLEX processor. The PLEX processor automatically checks whether an exit routine named S2KTRANS was link-edited with the PLEX processor. If it was, do not specify the TRANS execution option without approval from the systems personnel at the site, because it overrides use of the standard routine.

# SAMPLE FORTRAN PLEX PROGRAM

A generalized, skeleton FORTRAN PLEX program is shown in Illus. C.4. FORTRAN statements can appear anywhere in a PLEX program. However, executable PLEX commands can appear only after the START S2K command.

The rest of this section shows a sample program that uses the EMPLOYEE database to produce a listing of employees by department managers.

**Illus. C.4** Skeleton of a PLEX Program

```
COMMBLOCK(s)                          (One COMMBLOCK must be included
                                      for each database to be accessed.)
      :
      :
SSR(s)                                (One or more subschema records can
                                      be included to refer to one or more
                                      schema records.  Each SSR
                                      refers to one schema record.)
      :
      :
$BLOCKS.                              (The program must have one $BLOCKS
                                      directive after the COMMBLOCK and
                                      SUBSCHEMA declarations.)
      :
      :
START S2K.                            (First PLEX command executed by the
                                      program)
      :
      :
Initialize password(s) in
COMMBLOCK(s) and open database(s)
      :
      :
PLEX command(s)
      :
      :
CLOSE database(s)
      :
      :
PLEX command(s)
      :
      :
CLOSE remaining database(s)
      :
      :
STOP S2K                              (Last PLEX command executed by the
                                      program)
```

## FINDORG Program

The sample program FINDORG uses PLEX retrieval commands and several stacks and Locate Files to produce a roster of all personnel in the company. In this roster, employees are listed under their managers.

The program has one subschema record for level 0 data records. It also has two subschema records that refer to schema record C100, (POSITION WITHIN COMPANY). The C100 subschema record POS is INTERNAL. It is used to retrieve the title of an employee's position. The other C100 subschema record, COMPAR, contains two items used for comparing values in the LOCATE command's where-clause.

The program retrieves employee records according to the hierarchy of the organization. It finds the organizational hierarchy by extracting the initials from each manager's name and looking for those initials in the MANAGER item of the most current POSITION WITHIN COMPANY data record in each logical entry.

A GET1 command retrieves the president's level 0 data record. Then a LOCATE command establishes a Locate File pointing to all employees who report directly to him. That is, Locate File 1 points to the level 0 data records of all employees who report directly to the president.

Next a GET command retrieves a level 0 record for an employee whose address is in Locate File 1. The program extracts the initials from the employee name in this record. Then a LOCATE command finds the level 0 records of all personnel who report directly to this employee, and so on, down the company's chain of command.

When an employee who has no subordinates is retrieved (that is, the Locate File is empty), the program logic reverts to the Locate File pointing to this employee's level 0 record. A GET command retrieves the next employee, if there is one, who works for the same manager.

In general, when the program receives return code 04 at the end of one Locate File, it issues a GET command for the previously established Locate File (whose number is one less than that of the latest Locate File). Thus, the program proceeds as far down in the organizational hierarchy as it can before moving across the hierarchy. Eventually, the end of Locate File 1 is reached, which means that all subordinate employees were retrieved.

As the program retrieves each employee's record, it calls a subroutine named SCAN that

- extracts the initials from the employee's forename and the last name for use in the next LOCATE command
- concatenates the forename, last name, and job title into one line for printing by the main program.

Each output line is indented under the line for the employee's immediate manager.

See Illus. C.5 on page C-53 for the FINDORG source statements. Illus. C.6 on page C-62 shows the output from this program.

**Illus. C.5** Source Statements for the FINDORG Program

```
C*********************************************************************
C                          EXAMPLE PROGRAM FINDORG                    *
C ABSTRACT:                                                           *
C    THIS PROGRAM USES THE EMPLOYEE DATABASE TO FIND THE              *
C    ORGANIZATION OF THE COMPANY.  IT USES MULTIPLE LOCATE FILES      *
C    AND STACKS.  IT STARTS WITH THE PRESIDENT, FINDS ALL PEOPLE      *
C    WHO WORK FOR THE PRESIDENT, AND FOR EACH OF THEM FINDS ALL       *
C    SUBORDINATES, AND SO ON.  THE NAME AND TITLE OF EACH EMPLOYEE    *
C    IS PRINTED.  THE NAME OF EACH EMPLOYEE APPEARS UNDER HIS         *
C    SUPERVISOR AND IS INDENTED TO SHOW THE APPROPRIATE LEVEL OF THE  *
C    ORGANIZATION.  IT IS ASSUMED THAT EACH EMPLOYEE HAS A UNIQUE     *
C    COMBINATION OF INITIALS.                                         *
C                                                                     *
C NARRATIVE:                                                          *
C    STEP                      ACTION                               *
C    1.   INITIALIZE, GET 15 ADDITIONAL LOCATE FILES, OPEN DATABASE.  *
C    2.   RETRIEVE THE PRESIDENT TO BEGIN THE ORGANIZATION LIST.      *
C         (GET1 OR GET COMMAND)                                       *
C    3.   LOCATE SUBORDINATES OF THE EMPLOYEE JUST RETRIEVED (BY A    *
C         (GET1 OR GET COMMAND)                                       *
C    4.   RETRIEVE THE NEXT SUBORDINATE AT THE CURRENT LEVEL OF THE   *
C         ORGANIZATION.                                               *
C    5.   FINISH ORGANIZATION CHART, CLOSE DATABASE, STOP.            *
C*********************************************************************
      PROGRAM FNDORG
      IMPLICIT INTEGER (A-W)
C
C*****
C  THE PROGRAM VARIABLES:
C
C   CFNAME    - HOLDS VALUE OF FNAME IN SSR ENTRY FOR USE BY
C               SUBROUTINE SCAN
C   CLNAME    - HOLDS VALUE OF LNAME IN SSR ENTRY FOR USE BY
C               SUBROUTINE SCAN
C   CTITLE    -  HOLDS VALUE OF TITLE IN SSR POS FOR USE BY
C               SUBROUTINE SCAN
C   DEMO      -  HOLDS VALUE OF PASSWORD (MOVED TO PASSWD FIELD OF
C               COMMBLOCK)
C   FMT       -  HOLDS VARIABLE FORMAT FOR WRITE STATEMENT FOR OUTLIN
C   FMTBLK    -  HOLDS 2-BYTE VALUE ('H ') MOVED TO FMT AFTER TOP LINE
C               WRITTEN (GIVES SINGLE-SPACING)
C   FNULL     -  HOLDS STRING 'NULL' TO COMPARE TO LNAME FOR PRINTING
C               LINE FOR NULL DATA ENTRY, USING VARIABLE FORMAT VACFMT
C   GET,GETD,GET1,LOCATE,OPEN  -  ALL HOLD NAMES OF COMMANDS; SENT AS
C               PARAMETERS TO SUBROUTINE ERRORS IF NON-ZERO RETURN CODES
C   IFLAG     -  USED IN STEPS 4 AND 5 TO DECIDE IF THE ORGANIZATION HAS
C               MORE THAN THE 16 HIERARCHICAL LEVELS THE PROGRAM CAN
C               SHOW
```

*continued on next page*

**Illus. C.5** *continued*

```
C   INCOMP  -  'INCOMPLETE ORGANIZATION' VALUE FOR IFLAG
C   INITLS   - EMPLOYEE'S INITIALS DERIVED FROM NAME BY SUBROUTINE
C              SCAN; USED FOR LOCATE WHERE-CLAUSE COMPARISON
C   LFNUM    - THE SUBSCRIPT OF THE CURRENT LOCATE FILE AND STACK
C   OUTLIN   - OUTPUT LINE: EMPLOYEE FORENAME, LAST NAME, JOB TITLE
C              CONCATENATED BY SUBROUTINE SCAN
C   PRES     - HOLDS PRESIDENT'S JOB TITLE FOR GET1 WHERE-CLAUSE
C   TABS     - HOLDS 2-BYTE VALUES SUBSTITUTED FOR 'T' VALUE IN FMT,
C              TO CAUSE APPROPRIATE INDENTATION FOR OUTLIN
C*****
      DIMENSION CFNAME(4), CLNAME(4), OUTLIN(20)
      CHARACTER*2 FMT(7), FMTBLK, TABS(15), VACFMT(23)
      CHARACTER*4 DEMO, GET, GETD, GET1, OPEN, LOCATE, IFLAG, INCOMP
      CHARACTER*48 CTITLE
      COMMON /TFSCAN/ CFNAME, CLNAME, CTITLE, INITLS, OUTLIN
C
      COMMBLOCK/ EMPLOYEE/ SSRNAM, RCODE, FREE, LASTDR, PASSWD,
     -                     RECNUM, LFPTR, LEVEL, CTIME, CDATE,
     -                     CYCLE, SEPTOR, TERMTR, DAMAGE
C
C  SSR WITH EMPLOYEE'S FORENAME AND LAST NAME
      INTERNAL SUBSCHEMA RECORD/ ENTRY OF EMPLOYEE/
     -                  FNAME(4)   EQ C3,
     -                  LNAME(4)   EQ C2
C
C  SSR WITH EMPLOYEE'S JOB TITLE
      INTERNAL SUBSCHEMA RECORD/ POS OF EMPLOYEE/
     -                  TITLE(C(48))  EQ C101
C
C  SSR WITH TWO ITEMS FOR WHERE-CLAUSE COMPARISONS:
C   MGR      - INITIALS OF EMPLOYEE'S MANAGER
C   ENDDAT   - ONLY CURRENT EMPLOYEES ARE TO BE LISTED (THEY HAVE
C              NO END DATE FOR THEIR CURRENT POSITIONS)
      SUBSCHEMA RECORD/ COMPAR OF EMPLOYEE/
     -                  MGR        EQ C103,
     -                  ENDDAT(2)  EQ C106
C
      $BLOCKS
```

*continued on next page*

**Illus. C.5** *continued*

```
      DATA DEMO /'DEMO'/
C           FMT HOLDS THE VARIABLE FORMAT (1H1,T02,20A4)
      DATA FMT /'(1', 'H1', ',T', '02', ',2', '0A', '4)'/
      DATA FMTBLK /'H '/
      DATA FNULL /ZD5E4D3D3/
      DATA GET  /'GET '/
      DATA GETD /'GETD'/
      DATA GET1 /'GET1'/
      DATA IFLAG /'C   '/
      DATA INCOMP /'INCM'/
      DATA LOCATE /'LOC '/
      DATA OPEN /'OPEN'/
      DATA TITLE /'PRESIDENT'/
      DATA TABS /'04', '06', '08', '10', '12', '14', '16', '18',
     -          '20', '22', '24', '26', '28', '30', '32'/
C           VACFMT HOLDS THE VARIABLE FORMAT FOR A NULL ENTRY:
C               (1H ,T02, 25H*** VACANT POSITION ***, , 12A4)
      DATA VACFMT /'(1', 'H ', ',T', '02', ', ', '25', 'H*',
     -             '**', ' V', 'AC', 'AN', 'T ', 'PO', 'SI',
     -             'TI', 'ON', ' *', '**', ', ', ', ', ', '12',
     -             'A4', ') '/
C
C*****************************************************************************
C                                                                             *
C                              TOP OF MODULE                                  *
C                                                                             *
C*****************************************************************************
C
C***
C STEP 1: REQUEST 15 EXTRA LOCATE FILES, OPEN DATABASE
C***
      START S2K(15)
          IF (S2KRTC .EQ. 0) GO TO 100
               WRITE (6,50) S2KRTC
   50          FORMAT ('1','S2KRTC = ', I4, ' ON START S2K')
               STOP S2K
               STOP
  100 CONTINUE
      PASSWD = DEMO
      OPEN EMPLOYEE
          IF (RCODE .EQ. 0) GO TO 150
               CALL ERRORS  (OPEN)
               GO TO 999
  150 CONTINUE
```

*continued on next page*

**Illus. C.5** *continued*

```
C
C***
C STEP 2: RETRIEVE THE PRESIDENT TO BEGIN THE ORGANIZATION LIST
C***
      GET1 ENTRY WHERE TITLE AND ENDDAT FAILS
          IF (RCODE .EQ. 0) GO TO 200
               CALL ERRORS (GET1)
               GO TO 999
  200 CONTINUE
          DO 210 I=1,4
          CFNAME(I) = FNAME(I)
          CLNAME(I) = LNAME(I)
  210 CONTINUE
          CTITLE = TITLE
  220 CONTINUE
          CALL SCAN
          WRITE (6,FMT) (OUTLIN(I), I=1,20)
          FMT(2) = FMTBLK
          LFNUM = 1
  225 CONTINUE
C
C***
C STEP 3: LOCATE SUBORDINATES OF THE EMPLOYEE JUST RETRIEVED (BY A
C         GET1 OR GET COMMAND)
C***
      LOCATE (LFNUM) ENTRY WHERE MGR EQ INITLS AND ENDDAT FAILS
          IF (RCODE .EQ. 0) GO TO 250
               CALL ERRORS (LOCATE)
               GO TO 999
  250 CONTINUE
          IF (RECNUM .GT. 0) GO TO 300
          IF (LFNUM .EQ. 1) GO TO 600
          LFNUM = LFNUM -1
  300 CONTINUE
```

*continued on next page*

**Illus. C.5** *continued*

```
C
C***
C STEP 4: RETRIEVE THE NEXT SUBORDINATE AT THE CURRENT ORGANIZATION
C         LEVEL
C***
      GET (LFNUM) ENTRY NEXT
          IF (RCODE .NE. 4) GO TO 350
               LFNUM = LFNUM - 1
               IF (LFNUM .EQ. 0) GO TO 600
               GO TO 300
  350     CONTINUE
          IF (RCODE .EQ. 0) GO TO 400
               CALL ERRORS (GET)
               GO TO 999
  400 CONTINUE
      GETD (LFNUM) POS NEXT
          IF (RCODE .EQ. 0) GO TO 450
               CALL ERRORS (GETD)
               GO TO 999
  450 CONTINUE
      IF (LNAME(1) .NE. FNULL) GO TO 455
      VACFMT(4) = TABS(LFNUM)
      WRITE (6,VACFMT) (TITLE)
      GO TO 480
  455 CONTINUE
      DO 460 I=1,4
          CFNAME(I) = FNAME(I)
          CLNAME(I) = LNAME(I)
  460 CONTINUE
          CTITLE = TITLE
  470 CONTINUE
      CALL SCAN
      FMT(4) = TABS(LFNUM)
      WRITE (6,FMT) (OUTLIN(I),I=1,20)
  480 CONTINUE
      IF (LFNUM .LT. 15) GO TO 500
          IFLAG = INCOMP
          GO TO 300
  500 CONTINUE
      LFNUM = LFNUM + 1
      GO TO 225
  600 CONTINUE
```

*continued on next page*

**Illus. C.5** *continued*

```
C
C***
C STEP 5: FINISH ORGANIZATION LIST, CLOSE DATABASE, STOP
C***
      WRITE (6,610)
  610 FORMAT ('0','ORGANIZATION LIST COMPLETED')
      IF (IFLAG .NE. INCOMP) GO TO 999
          WRITE (6,620)
  620     FORMAT ('0','ONLY 16 LEVELS OF COMPANY SHOWN')
C
  999 CONTINUE
      CLOSE EMPLOYEE
      STOP S2K
      STOP
      END
      SUBROUTINE SCAN
C ***** SUBROUTINE USES EMPLOYEE FORENAME, LAST NAME, AND JOB
C       TITLE; EXTRACTS INITIALS FROM FORENAME AND LAST NAME;
C       CONCATENATES FULL NAME WITH JOB TITLE; AND RETURNS INITIALS
C       AND CONCATENATED OUTPUT LINE.
C       FORENAME (FNAME), LAST NAME (LNAME), JOB TITLE (TITLE),
C       AND OUTPUT LINE (OUTLIN) ARE EACH EQUIVALENCED WITH
C       LOGICAL*1 ARRAYS FOR CHARACTER SCANNING AND CONCATENATING.
C
      IMPLICIT INTEGER (A-W)
      DIMENSION FNAME(4), LNAME(4), TITLE(12), OUTLIN(20)
      LOGICAL*1 BLKLOG(4), CHRLOG(4), BLKCOM(4), FNMARR(16), LNMARR(16),
     -          OUTARR(80), INITS(4), COMMA(4), TTLLOG(48)
      COMMON /TFSCAN/ FNAME, LNAME, TITLE, INITLS, OUTLIN
      EQUIVALENCE (BLKFLD,BLKLOG(1)), (BLANK,BLKCOM(1)),
     -            (CHAR,CHRLOG(1)), (FNAME(1),FNMARR(1)),
     -            (LNAME(1),LNMARR(1)), (INITLS,INITS(1)),
     -            (OUTLIN(1),OUTARR(1)), (FCOMMA,COMMA(1)),
     -            (TITLE(1),TTLLOG(1))
      DATA BLKFLD/4H    /
      DATA BLANK/0/
      DATA CHAR /0/
      DATA FCOMMA/4H,   /
C
C ***** SET INITLS TO BLANK AND BYTE TO BLANK FOR FORENAME AND LAST
C ***** NAME CHARACTER COMPARISONS.
      INITLS = BLKFLD
      BLKCOM(4) = BLKLOG(1)
```

*continued on next page*

**Illus. C.5** *continued*

```
C ***** FNAME SCAN -- WORK ON FNAME:
C  -- TAKE FIRST INITIAL, PUT IN FIRST BYTE OF INITS(INITLS) AND IN
C     FIRST BYTE OF OUTARR (OUTLIN).
C  -- SCAN FOR BLANK:
C        - IF FOUND, CHECK NEXT CHARACTER
C             - IF BLANK, GO TO LNAME SCAN
C             - IF NON-BLANK, TAKE INITIAL; RESUME FNAME SCAN
C        - IF NOT FOUND, GO TO LNAME SCAN
C  -- PUT CHARACTERS IN OUTARR (OUTLIN) AS SCAN PROCEEDS
C
      OUTARR(1) = FNMARR(1)
      INITS(1) = FNMARR(1)
      INIPOS = 1
      DO 250 FNMPTR=2,15
          CHRLOG(4) = FNMARR(FNMPTR)
          IF (CHAR .NE. BLANK) GO TO 200
          CHRLOG(4) = FNMARR(FNMPTR + 1)
          IF (CHAR .EQ. BLANK) GO TO 300
          INIPOS = INIPOS + 1
          INITS(INIPOS) = FNMARR(FNMPTR + 1)
  200     OUTARR(FNMPTR) = FNMARR(FNMPTR)
  250 CONTINUE
      CHRLOG(4) = FNMARR(16)
      IF (CHAR .EQ. BLANK) GO TO 300
      OUTARR(16) = CHRLOG(4)
      FNMPTR = 17
  300 CONTINUE
C ***** LNAME SCAN -- WORK ON LNAME:
C  -- PUT BLANK AFTER FNAME IN OUTARR (OUTLIN)
C  -- TAKE FIRST INITIAL OF LNAME
C  -- SCAN FOR BLANK:
C        - IF FOUND, CHECK NEXT CHARACTER
C             - IF BLANK, STOP LNAME SCAN
C             - IF NON-BLANK, TAKE INITIAL; RESUME LNAME SCAN
C        - IF NOT FOUND, END LNAME SCAN
C  -- PUT CHARACTERS IN OUTARR (OUTLIN) AS SCAN PROCEEDS
C
      BLKPOS = FNMPTR
      OUTARR(BLKPOS) = BLKLOG(1)
      INIPOS = INIPOS + 1
      INITS(INIPOS) = LNMARR(1)
      CHRPOS = BLKPOS + 1
      OUTARR(CHRPOS) = LNMARR(1)
```

*continued on next page*

**Illus. C.5** *continued*

```
      DO 400 LNMPTR=2,15
          CHRLOG(4) = LNMARR(LNMPTR)
          IF (CHAR .NE. BLANK) GO TO 350
          CHRLOG(4) = LNMARR(LNMPTR + 1)
          IF (CHAR .EQ. BLANK) GO TO 450
          INIPOS = INIPOS + 1
          INITS (INIPOS) = LNMARR(LNMPTR + 1)
  350     CHRPOS = CHRPOS + 1
          OUTARR(CHRPOS) = LNMARR(LNMPTR)
  400 CONTINUE
      CHRLOG(4) = LNMARR(16)
      IF (CHAR .EQ. BLANK) GO TO 450
      OUTARR(FNMPTR + 16) = CHRLOG(4)
      LNMPTR = 17
  450 CONTINUE
C ***** JOB TITLE: AFTER LNAME IN OUTARR (OUTLIN), PUT BLANK, COMMA,
C       AND JOB TITLE
      COMPOS = FNMPTR + LNMPTR
      OUTARR(COMPOS) = COMMA(1)
      BLKPOS = COMPOS + 1
      OUTARR(BLKPOS) = BLKLOG(1)
      TTLSTR = BLKPOS + 1
      TTLEND = TTLSTR + 47
      TTLPOS = 0
      DO 500 I=TTLSTR,TTLEND
          TTLPOS = TTLPOS + 1
          OUTARR(I) = TTLLOG(TTLPOS)
  500 CONTINUE
      RETURN
      END
```

*continued on next page*

**Illus. C.5** *continued*

```
      SUBROUTINE ERRORS(COMMND)
C ***** SUBROUTINE CALLED FOR NON-ZERO RETURN CODE: GIVES NAME OF
C       COMMAND AND DISPLAYS COMMBLOCK.
C
      CHARACTER*4 COMMND
      COMMBLOCK/ EMPLOYEE/ SSRNAM, RCODE, FREE, LASTDR, PASSWD,
     -                     RECNUM, LFPTR, LEVEL, CTIME, CDATE,
     -                     CYCLE, SEPTOR, TERMTR, DAMAGE
      $BLOCKS
      WRITE (6,100) COMMND, SSRNAM, RCODE, LASTDR,
     -        PASSWD, RECNUM, LFPTR, LEVEL, CTIME,
     -        CDATE, CYCLE, DAMAGE
  100 FORMAT ('1', 'COMMBLOCK AFTER ', A4, ' COMMAND:'//
     -        T5, 'SSR/ITEM NAME:', T30, 8A4,/
     -        T5, 'RETURN CODE:', T30, I4,/
     -        T5, 'LAST RECORD FLAG:', T30, I4,/
     -        T5, 'PASSWORD:', T30, A4,/
     -        T5, 'NO. LOCATED RECORDS:', T30, I4,/
     -        T5, 'LOC. FILE POINTER:', T30, I4,/
     -        T5, 'LEVEL/ORDINAL:', T30, I4,/
     -        T5, 'CYCLE TIME:', T30, 2A4,/
     -        T5, 'CYCLE DATE:', T30, 2A4,/
     -        T5, 'CYCLE NUMBER:', T30, I4,/
     -        T5, 'DAMAGE FLAG:', T30, I4)
      CLOSE EMPLOYEE
      STOP S2K
      STOP
      END
```

**Illus. C.6** Output from the FINDORG Program

```
CLIFTON P. WATERHOUSE, PRESIDENT
  YOLANDA SALAZAR, ADMINISTRATIVE ASSISTANT
  HUGH E. BOWMAN, EXECUTIVE VICE-PRESIDENT
    DARRYL NATHANIEL, SECRETARY
    OLAN M. GARRETT, VICE-PRESIDENT MARKETING
      ALTHEA KNIGHT, SECRETARY
      VIRGINA P. BROWN, MANAGER WESTERN REGION MARKETING
        FANNIE LITTLEJOHN, SECRETARY
        MADISON A. SCHOLL, SR SALES REPRESENTATIVE
          RUBEN R. BROOKS, JR SALES REPRESENTATIVE
        ARMANDO JUAREZ, SR SALES REPRESENTATIVE
        DAVID AMEER, SR SALES REPRESENTATIVE
      TRAVIS Z. RICHARDSON, MANAGER EASTERN REGION MARKETING
        MERRILEE D. WAGGONNER, SECRETARY
        ALAN F. GOODSON, SR SALES REPRESENTATIVE
          JERRY LEE SMITH, JR SALES REPRESENTATIVE
        LELAND G. SHROPSHIRE, JR SALES REPRESENTATIVE
        TAI CHAN, SR SALES REPRESENTATIVE
      GWENDOLYN VAN HOTTEN, MANAGER PUBLIC RELATIONS & EDUCATION
        ROMUALDO R. RODRIGUEZ, PR & ADVERTISING SPECALIST
        JANICE L. WILLIAMSON, INSTRUCTOR
        GEORGE J. GIBSON, INSTRUCTOR
    JOEL B. MILLSAP, VICE-PRESIDENT INFORMATION SYSTEMS
      CARRIE ANN FAULKNER, SECRETARY
      JACOB CAHILL, MANAGER SYSTEMS DEVELOPMENT & MAINTENANCE
        MOLLY I. GIBSON, TECHNICAL WRITER
        MICHAEL Y. JONES, SR SYSTEMS ANALYST
          DAVID G. REID, PROGRAMMER
          ***VACANT POSITION***, PROGRAMMER
          FRANCISCO GARCIA, JR PROGRAMMER/ANALYST
        RICHARD B. REDFOX, SYSTEMS ANALYST
      JESSE L. HERNANDEZ, MANAGER DATA BASE ADMINISTRATION
        LEOPOLD FREEMAN, SR SYSTEMS PROGRAMMER
        SOPHIA FERNANDEZ, STANDARDS & PROCEDURES ANALYST
      IVAN L. POLANSKI, MANAGER SYSTEMS OPERATIONS & RESOURCES
        JANET F. SMITH, SR SYSTEMS PROGRAMMER
          BRADFORD JOHNSON, JR SYSTEMS PROGRAMMER
        PEDRO QUINTERO, OPERATIONS SUPERVISOR
          WILLIAM D. SAVAGE, COMPUTER OPERATOR
          GARY SEATON, COMPUTER OPERATOR
          KENNETH D. REED, COMPUTER LIBRARIAN
    PATRICE R. KNAPP, VICE-PRESIDENT ADMINISTRATION & FINANCE
      FRANK A. CANADY, MANAGER PERSONNEL
        PENNY SCHMIDT, SECRETARY
        STEWART Q. THROCKMORTON, OFFICE SUPERVISOR
          LEONARD R. SLYE, GENERAL MAINTENANCE
          LILLIAN COLLINS, MAIL CLERK
          FREDDIE KAATZ, SUPPLY CLERK
      PATSY MUELLER, SECRETARY
      RITA M. JONES, MANAGER ACCOUNTING
        GARLAND P. SMITH, BOOKKEEPER

ORGANIZATION LIST COMPLETED
```

# JOB CONTROL LANGUAGE (JCL)

A PLEX job has the following four distinct steps:

- preprocessing with the PLEX processor
- compiling
- link-editing
- executing.

This section discusses sample JCL for these job steps for both single-user and Multi-User environments. **FORTRAN PLEX Processor Execution Options** on page C-39 describes the execution parameters that you can use with this JCL.

## Single-user JCL for All Job Steps

In the START S2K command, you can specify that a program will be run in a single-user environment. When the subscript following the keyword USE is equal to zero or when USE and the subscript are not included in the command, it signifies a single-user environment. (See **START S2K** on page 6-1 for information about the START S2K command.)

The execution step requires scratch pads and sort files. These files are dynamically allocated and do not need to be specified in your JCL. See the *SYSTEM 2000 Product Support Manual, Version 12, First Edition* for information on pad and sort files and changing default sizes.

Illus. C.7 on page C-64 shows sample JCL required for all four steps in a single-user environment.

## Multi-User JCL for All Job Steps

The single-user JCL can also be used for a Multi-User environment with the following modifications:

- Delete all DD statements for SYSTEM 2000 work files and all database files from the GO step. (These files are set up when Multi-User is initialized and are shared by all users.)
- Reduce the REGION parameter in the GO statement to 128K.

In the START S2K command, you can specify that a program will be run in a Multi-User environment, by setting the subscript following the keyword USE equal to 1. (See **START S2K** on page 6-1 for information about the START S2K command.)

**Illus. C.7** Sample JCL for a Single-User FORTRAN PLEX Program

```
//*
//* -------------- MODIFY JOB CARD AS NEEDED -----------------
//*
//* -----  CHANGE SYSIN TO FORTRAN SOURCE PROGRAM  -----------
//PRECOM   EXEC PGM=PRCOMF,PARM='LANG=FTN3'
//STEPLIB  DD   DSN=S2K.R121.LOAD,DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSN=S2K.R121.SOURCE(FINDORG),
//              DISP=SHR
//SYSGO    DD   DSN=&&FSOURCE,DISP=(MOD,PASS),UNIT=SYSDA,
//              DCB=(BLKSIZE=800,LRECL=80,RECFM=FB),
//              SPACE=(CYL,(1,1))
//SYSUT1   DD   UNIT=SYSDA,SPACE=(TRK,(20,20))
//SYSLIB   DD   DSN=S2K.R121.SOURCE,DISP=SHR
//* ------------------------------------------------------------
//FORT     EXEC PGM=FORTVS,
//              PARM='NOLIST,NODECK,MAP,LOAD,XREF',
//              COND=(4,LT,PRECOM)
//* ---------------  STEPLIB MAY BE REQUIRED -----------------
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSN=&&FSOURCE,DISP=(OLD,DELETE)
//SYSTERM  DD   SYSOUT=A
//SYSLIN   DD   DSN=&&FORTOBJ,DISP=(NEW,PASS),
//              SPACE=(3200,(25,6)),UNIT=SYSDA
//SYSUT1   DD   UNIT=SYSDA,SPACE=(3465,(3,3)),
//              DCB=BLKSIZE=3465
//SYSUT2   DD   UNIT=SYSDA,SPACE=(5048,(10,10))
//* -------  CHANGE SYSLMOD TO YOUR LOAD MODULE NAME   -------
//LKED     EXEC PGM=IEWL,
//              PARM='MAP,XREF,LIST,LET,SIZE=(164K,24K),AMODE(24)',
//              COND=((4,LT,PRECOM),(8,LT,FORT))
//SYSLIN   DD   DSN=&&FORTOBJ,DISP=(OLD,DELETE)
//SYSLIB   DD   DSN=SYS1.VFORTLIB,DISP=SHR
//         DD   DSN=S2K.R121.LOAD,DISP=SHR
//SYSLMOD  DD   DSN=S2K.R121.LOAD(FINDORG),DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   SPACE=(1024,(400,20)),UNIT=SYSDA
```

*continued on next page*

**Illus. C.7** *continued*

```
//* ------------------------------------------------------------
//GO       EXEC PGM=*.LKED.SYSLMOD,
//              COND=((8,LT,FORT),(8,LT,LKED))
//STEPLIB  DD   DSN=S2K.R121.LOAD,DISP=SHR
//S2KPARMS DD   DSN=S2K.R121.CNTL(NLPARM),DISP=SHR
//STEPLIB  DD   DSN=S2K.R121.LOAD,DISP=SHR
//S2KMSG   DD   SYSOUT=A,DCB=(BLKSIZE=1320,LRECL=132)
//SYSOUT   DD   SYSOUT=A
//SYSUDUMP DD   SYSOUT=A
//S2KCOMD  DD   DUMMY
//* ---------------  DATA BASE FILES  -----------------------
EMPLOYE1   DD   DSN=S2K.R121.EMPLOYE1,DISP=OLD
EMPLOYE2   DD   DSN=S2K.R121.EMPLOYE2,DISP=OLD
EMPLOYE3   DD   DSN=S2K.R121.EMPLOYE3,DISP=OLD
EMPLOYE4   DD   DSN=S2K.R121.EMPLOYE4,DISP=OLD
EMPLOYE5   DD   DSN=S2K.R121.EMPLOYE5,DISP=OLD
EMPLOYE6   DD   DSN=S2K.R121.EMPLOYE6,DISP=OLD
EMPLOYE7   DD   DSN=S2K.R121.EMPLOYE7,DISP=OLD
EMPLOYE8   DD   DSN=S2K.R121.EMPLOYE8,DISP=OLD
//* ----------  FORTRAN FILES AS REQUIRED  -------------------
//FT05F001 DD   DDNAME=SYSIN
//FT06F001 DD   SYSOUT=A,
//             DCB=(BLKSIZE=1320,LRECL=132,RECFM=FBA)
```

## JCL Variations

The JCL given in the previous section is for all four steps. You can omit one or more steps, depending on the nature of the job. For example, the job might consist of only the preprocessing step or only the preprocessing and compilation steps. The PLEX processor execution options described in **FORTRAN PLEX Processor Execution Options** on page C-39 also provide for variations in JCL.

The following topics discuss optional DD statements that you can use.

**Additional Locate Files (single-user)** One Locate File, Locate File 0, is available to a program by default. This file is a SYSTEM 2000 work file with the DDname S2KSYS01. However, for a single-user job that does not use scratch pads but does use additional Locate Files, more DD statements must be provided for the Locate Files. The DDname for Locate File 1 is LOCATE00; Locate File 2 is LOCATE01, and so on.

```
//GO.EMPLOYE1 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE1
//GO.EMPLOYE2 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE2
//GO.EMPLOYE3 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE3        database files
//GO.EMPLOYE4 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE4
//GO.EMPLOYE5 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE5
//GO.EMPLOYE6 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE6
//GO.LOCATE00 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 1
//GO.LOCATE01 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 2
//GO.LOCATE02 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 3
//GO.LOCATE03 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 4
//GO.LOCATE04 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 5
//GO.LOCATE05 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 6
//GO.LOCATE06 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 7
//GO.LOCATE07 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 8
//GO.LOCATE08 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 9
//GO.LOCATE09 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 10
//GO.LOCATE10 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 11
//GO.LOCATE11 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 12
//GO.LOCATE12 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 13
//GO.LOCATE13 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 14
//GO.LOCATE14 DD UNIT=SYSDA,SPACE=(CYL,(1,1))         Locate File 15
```

**Note**: In a Multi-User environment, DD statements for Locate Files are never included in the JCL for each program.

**Additional database files** The Update Log, Rollback Log, Savefile, and Keepfile, as well as database files, will be allocated dynamically by the software if you do not include DD statements for them. The first seven characters of the DDname are the first seven characters of the database name, with Xs replacing embedded (and trailing) blanks. The suffix for the Savefile is S and for the Keepfile is K. Programs that issue KEEP and APPLY commands always use the Keepfile.

**Note**: In a Multi-User environment, DD statements for database files are never included in the JCL for each program.

**Example**

```
//GO.EMPLOYE1 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE1
//GO.EMPLOYE2 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE2
//GO.EMPLOYE3 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE3
//GO.EMPLOYE4 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE4
//GO.EMPLOYE5 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE5
//GO.EMPLOYE6 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE6
//GO.EMPLOYE7 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE7          Update Log
//GO.EMPLOYE8 DD DISP=OLD,DSN=SAS.DOC.EMPLOYE8          Rollback Log
//GO.EMPLOYEK DD DSNAME=DATABASE,UNIT=(TAPE9,,DEFER),   Keepfile
//            DISP=(OLD,PASS),VOL=SER=SAS002
//GO.EMPLOYES DD DSNAME=DATABASE,UNIT=(TAPE9,,DEFER),   Savefile
//            DISP=(OLD,PASS),VOL=SER=SAS001
```

## $COPY Library

When you use the $COPY directive to copy COMMBLOCK declarations or subschema record declarations from a library, you must specify the library in a SYSLIB DD statement in the preprocessing step.

**Example**

```
//PRECOM    EXEC      PGM=PRECOMC,PARM='COPY, . . . ', . . .
   .
   .
   .
//SYSLIB    DD        DSN=S2K.COPYLIB,DISP=SHR    $COPY LIBRARY
   .
   .
   .
//FORT      EXEC. . .
```

Appendix D

# Assembler PLEX

*FORMAT OF PLEX COMMANDS D-2*

*SYSTEM 2000 ITEM TYPES AND ASSEMBLER DATA FORMS D-3*

*COMMBLOCK DECLARATIONS D-7*
- *Format D-7*
- *Example D-7*

*SUBSCHEMA RECORD DECLARATIONS D-9*
- *Format D-9*
- *Example D-11*
- *Considerations D-12*

*INCLUSION LIST D-13*
- *Format D-13*
- *Item List D-15*
- *Designation Bit String D-18*
- *Designation Vector D-21*

*USING 0 OPTION D-24*

*PLEX PROCESSOR DIRECTIVES D-25*
- *$ALLOCATE Directive D-25*
- *$S2KDUM Directive D-27*

*PLEX PROCESSOR EXECUTION OPTIONS D-32*
- *CICS Execution Option D-32*
- *Other PLEX Processor Execution Options D-32*

*SAMPLE ASSEMBLER PLEX PROGRAMS AND JCL D-32*

This appendix provides Assembler-specific information for the PLEX facility. It includes discussions of the general format of PLEX commands, COMMBLOCK and SUBSCHEMA declarations, Assembler PLEX processor directives and execution options, and a reference to a sample Assembler PLEX program. You should use the material in the main part of this manual as the basic documentation.

The Assembler PLEX feature supports all capabilities and functional characteristics described in the main text of this manual. The Assembler PLEX processor accepts all standard PLEX commands and generates Assembler code that is acceptable to standard MVS and CMS Assemblers. Reentrant coding techniques and the type of coding required by the SYSTEM 2000 TP interfaces are supported. The PLEX processor also recognizes the END Assembler directive indicating the end of a program so that multiple programs can be processed in a job.

# FORMAT OF PLEX COMMANDS

The format of PLEX commands for Assembler language is similar to the formats for other languages. That is, each PLEX command can be coded and recognized by keyword or, optionally, by a fixed format.

The FORM/NOFORM execution option determines the format. The default is NOFORM, which means the PLEX processor looks for keywords. FORM means that for Assembler PLEX source code, a hyphen in column 1 causes the line to be processed by the PLEX processor (see **PLEX Processor Execution Options** on page D-32). If you do not specify FORM and the PLEX processor finds "errors" in the syntax of the START, OPEN, LOAD, RESTORE, SAVE, or CLOSE commands, the PLEX processor passes the command on to the Assembler without issuing an error diagnostic. These commands are used in Assembler programs with non-SYSTEM 2000 meaning. If the Assembler does not accept the syntax associated with these verbs, it issues an appropriate diagnostic.

The following rules pertain to coding PLEX commands:

- Each PLEX command must begin on a separate line. The command can start in any column except column 1 when the FORM option is specified. The lack of a continuation symbol in column 72 terminates the command.

- You can continue a PLEX command beyond one line. Continuation is indicated by a non-blank symbol in column 72, which specifies that the following line also needs to be scanned by the PLEX processor. With the NOFORM option, the continuation line can begin in any column. If you specify the FORM option, the continuation line cannot begin in column 1, since that option requires the hyphen in column 1. Subschema items do not need the hyphen.

- A statement containing a PLEX command cannot contain a comment. Comments can precede or follow the PLEX commands on separate lines.

- A statement containing a PLEX command cannot contain a statement label. The only exception is that items in a subschema are declared with Assembler language format. That is, subschema item names begin in column 1 (see **SUBSCHEMA RECORD Declarations** on page D-9).

- For some commands, the PLEX processor handles item lists, designation bit strings, and designation vectors as described in **Inclusion List** on page D-13.

# SYSTEM 2000 ITEM TYPES AND ASSEMBLER DATA FORMS

Illus. D.1 shows SYSTEM 2000 item types and Assembler data forms.

**Illus. D.1** Correspondence Between SYSTEM 2000 Item Types and Assembler Data Forms

| Item type | Assembler Data Forms | Requirements/Restrictions | From Database | To Database |
|---|---|---|---|---|
| **TEXT** | **C**<br>**CL***n* | If the value is shorter than its subschema item field, a byte of binary zeros must follow the end of the value in the SSR to avoid storage of trailing blanks for updates and to insure successful comparisons in a where-clause.<br><br>There is no way of knowing the exact number of trailing blanks with a retrieval. | Stored left-justified, blank-filled or truncated on right as necessary.<br><br>Null = binary zeros. | Placed in the database left-justified, binary zero-filled or overflows to the Extended Field Table as necessary.<br><br>You should place the value left-justified and place a byte of zeros at the end, if necessary.<br><br>Null = binary zeros in the first byte. |
| **CHAR** | **C**<br>**CL***n* | | Same as in TEXT. | Placement same as in TEXT.<br><br>Null = binary zeros in the first byte or all blanks. |
| **DATE** | | Must be 6 or 8 bytes long. | Placed according chosen format.<br><br>Null = binary zeros. | Value should conform to chosen data format.<br><br>Null = binary zeros. |
| **INTEGER** | **P**<br>**PL***n* | Subschema item field must match schema item field in total length in bytes. Number of digits to the right of the decimal point must be zero. | Stored right-justified with zero-fill in the leftmost byte if necessary. Truncation is not allowed.<br><br>Null = binary zeros. | Placed in the database right-justified with zero-fill in the leftmost byte if necessary. Truncation is not allowed.<br><br>Null = binary zeros. |
| **INTEGER** | **H**<br>**F** | Abend if the value cannot be represented in 32 bits. | Binary representation is placed. Truncation can occur on the left.<br><br>Null = NULL or NL. | Same as above.<br><br>Null = NULL or NL. |

*continued on next page*

**Illus.** **D.1** *continued*

| Item type | Assembler Data Forms | Requirements/Restrictions | From Database | To Database |
|---|---|---|---|---|
| **INTEGER** | **Z**<br>**ZL***n* | Only numeric picture, no edit characters. | Character representation is placed with aligned decimal point and zero-fill. High-order digits are truncated if necessary.<br><br>Null = binary zeros. | Placed in the database right-justified and zero-filled. The least significant digits might be rounded or truncated; the most significant digits might be truncated if necessary.<br><br>Null = binary zeros. |
| **INTEGER** | **E**<br>**D** | Number of significant digits must not exceed the schema item picture. | Floating point representation is retrieved.<br><br>Null = NULL in the first word. | Placed in the database right-justified and zero-filled. Truncation is not allowed.<br><br>Null = NULL in the first word. |
| **DECIMAL** or **MONEY** | **P**<br>**PL***n* | Subschema item field must match the schema item field in total length in bytes and in the number of digits to the right of the decimal point. | Stored decimal point aligned with zero-fill in the leftmost byte if necessary. Truncation is not allowed.<br><br>Null = binary zeros. | Placed in the database decimal-point aligned with zero-fill in the leftmost byte if necessary. Truncation of the most significant digits is not allowed.<br><br>Null = binary zeros. |
| **DECIMAL** or **MONEY** | **H**<br>**F** | Decimal point, if present, should be at the right end in the schema item picture.<br><br>Abend if binary representation exceeds 32 bits. | Binary representation is placed right-justified. Truncation of the most significant bits can occur.<br><br>Null = NULL or NL. | Same as above. |
| **DECIMAL** or **MONEY** | **Z**<br>**ZL***n* | Only numeric pictures without edit characters are allowed. | Character representation is placed decimal-point-aligned and zero-filled. Truncation can occur on either side as necessary.<br><br>Null = binary zeros. | Placed in the database decimal-point-aligned, zero-filled if necessary. Truncation can occur on either side.<br><br>Null = binary zeros. |

*continued on next page*

**Illus.** **D.1** *continued*

| Item type | Assembler Data Forms | Requirements/Restrictions | From Database | To Database |
|---|---|---|---|---|
| **DECIMAL** or **MONEY** | **D** or **E** | Number of significant digits to the left of the decimal point must not exceed the schema item picture. | Floating point representation is retrieved.<br><br>Null = NULL in the first word. | Decimal point is aligned and zero-fill occurs as necessary. Number of digits to left of decimal point must not exceed the schema item picture. If the number of digits to the right of the decimal point exceeds the corresponding number in schema item picture, rounding or truncation occurs. Value must not exceed the maximum allowed by schema item picture.<br><br>Null = NULL in the first word. |
| **REAL** or **DOUBLE** | **D** or **E** | Requires fullword alignment or doubleword alignment. The OPT039 and OPT040 execution parameters determine how the system handles padding and truncation (see the *SYSTEM 2000 Product Support Manual, Version 12, First Edition*).<br><br>Zero means a positive zero. Negative zero is not allowed; causes return code 10. | Internal normalized floating point number is placed in the subschema. External floating point numbers are not allowed in the subschema.<br><br>Null = NULL in first word. | Must be internal normalized floating point number. The software performs no conversions. External floating point numbers not allowed.<br><br>Null = NULL in first word. |

*continued on next page*

**Illus.** **D.1** *continued*

| Item type | Assembler Data Forms | Requirements/Restrictions | From Database | To Database |
|---|---|---|---|---|
| **UNDEFINED** | **C**<br>**CL***n* | Each byte represents one byte of data. Data conversions from hexadecimal to display are not performed. If an UNDEFINED value overflows its picture, return code 10 is issued. | Value placed left-justified in subschema with trailing zero-fill.<br><br>Null = all binary zeros. | All 256 EBCDIC characters are allowed. Placed in data base left-justified with trailing zero-fill.<br><br>Null = all binary zeros. |

# COMMBLOCK DECLARATIONS

## Format

```
[DSECT] COMMBLOCK / [alias EQ] database / var-1,...,var-14 [,var-15]
```

The COMMBLOCK statement defines the status communication area. Because the COMMBLOCK syntax depends on the storage allocation method of the programming language, the format varies among the languages. The DSECT option is a special storage classification for Assembler language. If you omit the DSECT option, the PLEX processor generates one COMMBLOCK area using DC and initial values. You can use this form for a COMMBLOCK that is part of a larger DSECT as long as you provide the proper initial values in the actual area referenced by this definition. This would be the case if you pass the DSECT as a parameter and the calling program has already initialized it.

If you specify DSECT, the PLEX processor generates two COMMBLOCKs. One COMMBLOCK is generated in the line named *alias*. You must provide addressability. The second COMMBLOCK is generated at the location of the $BLOCKS directive. It consists of only the prefix needed by SYSTEM 2000 software. Before using this COMMBLOCK, you must move the initial values from the second COMMBLOCK to the DSECT COMMBLOCK. You can accomplish this most easily with the $ALLOCATE command (see **$ALLOCATE Directive** on page D-25).

**Note**: The COMMBLOCK alias name cannot exceed seven characters, because the PLEX processor appends a suffix (0) to the alias name. If you do not specify an alias, the database name is truncated to seven characters.

## Example

The following example shows a COMMBLOCK statement for the LIBRARY database. Notice the continuation character (non-blank in column 72). The PLEX processor removes these continuation characters, because they are not acceptable to the Assembler.

```
                                                          Col. 72
COMMBLOCK/STACKS EQ LIBRARY/SCHEME,RTNCD,FREE,LASTDS,             X
   PASSWD,DATASN,DATASP,LEVEL,CTIME,CDATE,CYCLE,                  X
   SEPSYM,ENTRYT,DAMAGE,PARENT
```

The PLEX processor generates the following output:

```
*      COMMBLOCK/STACKS EQ LIBRARY/SCHEME,RTNCD,FREE,LASTDS,
*           PASSWD,DATASN,DATASP,LEVEL,CTIME,CDATE,CYCLE,
*           SEPSYM,ENTRYT,DAMAGE,PARENT
                DS    0D
  STACKS0       DS    0CL124
                DC    CL16'LIBRARY'
                DC    F'0,124,0'
  STACKS        DS    0CL96
  SCHEME        DC    CL32'    '
  RTNCD         DC    F'0'
  FREE          DC    F'0'
  LASTDS        DC    F'0'
  PASSWD        DC    CL4'     '
  DATASN        DC    F'0'
  DATASP        DC    F'0'
  LEVEL         DC    F'0'
  CTIME         DC    CL8'     '
  CDATE         DC    CL8'     '
  CYCLE         DC    F'0'
  SEPSYM        DC    F'0'
  ENTRYT        DC    CL4'     '
  DAMAGE        DC    F'0'
  PARENT        DC    F'0'
```

# SUBSCHEMA RECORD DECLARATIONS

A subschema describes the subset of database schema items accessible to the program. The subschema is composed of subschema records that define the storage area for values from data records conforming to a schema record.

## Format

```
[DSECT] SUBSCHEMA [RECORD]/subschema OF database/
   [subschema DS             0CLXX]

                             |C
                             |CLn
                             |X
                             |XLn
                             |H
                             |F
        name        |DC      |P      ['initial-value']
                    |DS      |PLn
                             |E
                             |D
                             |Z
                             |ZLn

   [DECIMAL [DIGITS TO THE RIGHT] m]

   [EQ component]

   [FORM date-form]
                    .
                    .
                    .
$END
```

*subschema* is the name of the subschema record.

*database* is the name of the COMMBLOCK or alias for this subschema record.

*name* is the name of a schema item that the program intends to use. If the optional EQ *component* is not used, the name must be the SYSTEM 2000 component name or C-number.

*initial-value* passed to the Assembler and ignored by the PLEX processor.

**EQ** *component* is a SYSTEM 2000 component name or C-number (if different from the name used by the program).

Abbreviation: SUBSCHEMA RECORD = SSR

*date-form* is MMDDYYYY, MMDDYY, YYMMDD, YYYYMMDD, DDMMYYYY, or DDMMYY. The default is MMDDYYYY.

*m* is the number of digits that appear to the right of the decimal point for type DECIMAL and MONEY components. The default is zero digits to the right. This is valid only for P and Z type variables.

C and CL*n* is display code, where *n* has a value of 0 to 250. Zero indicates a minor structure, and a component is not represented. The length must be specified if it is greater than one.

X and XL*n* is hexadecimal code, where *n* has a value of 1 to 256.

H(halfword) and F(fullword) is numeric binary. INTEGER, DECIMAL, MONEY, REAL, and DOUBLE are the only schema item types that can have this form.

P and PL*n* is packed decimal, where *n* can be 1 through 8, indicating one to eight bytes. INTEGER, DECIMAL, MONEY, REAL, and DOUBLE are the only schema item types that can have this form. This is also the most efficient form for these types of items since they are stored in packed decimal format in the database and no editing is required for retrieval or update. You must specify the length if it is greater than one. The total size and digits to the right must match the schema item definition exactly, or return code 17 occurs. Database definitions with the total number of digits equal to an even number are increased by one. For example,

```
1* NUM1 (DECIMAL 9(5).9(4))
2* NUM2 (DECIMAL 9(4).9(4))
```

corresponds to:

```
NUM1        DC          PL5
            DECIMAL 4
NUM2        DC          PL5
            DECIMAL 4
```

E and D is floating point binary. E is single precision (one word), and D is double precision (two words). INTEGER, DECIMAL, MONEY, REAL, and DOUBLE are the only types of schema items that can have this form.

Z and ZL*n* is zoned decimal. Z is numeric display code. INTEGER, MONEY, DECIMAL, REAL, and DOUBLE are the only types of schema items that can have this form. You must specify the length *n* if it is greater than one. The total size and digits to the right must match the schema item definition exactly, or return code 17 occurs.

The use and meaning of items declared in each subschema record are discussed in the main text of this manual.

The DSECT and the default are the same as for the COMMBLOCK (see **COMMBLOCK Declarations** on page D-7). The second line containing *subschema* can be used to assign a label to the area containing the I/O area. The PLEX processor generates a prefix to this area, which contains SYSTEM 2000 control information. The name of this prefix is *name* concatenated with zero (0).

All data forms are supported. Subschema records are aligned on a doubleword boundary. The PLEX processor checks field alignment, which you must ensure.

## Example

Here is a sample SUBSCHEMA statement for Assembler PLEX. Notice that you can use either the DS or the DC specification of storage.

```
          SUBSCHEMA/BOOKS OF LIBRARY/

BOOKS     DS        0CL20

C2        DS        CL5

QUANITY   DS        PL2

DATES     DC        CL8'03151947'

C5        DC        CL5'ABCDE'

$END
```

The PLEX processor generates the following output:

```
*          SUBSCHEMA/BOOKS OF LIBRARY/
                   DS    0D
BOOKS0             DS    0CL300
                   DC    CL32'BOOKS',F'20,0,0,4,0,134217728'
                   DC    CL32'C2',F'224,0,0,0,0,65541'
                   DC    CL32'QUANITY',F'173,0,0,0,0,458752'
                   DC    CL32'DATES',F'119,0,0,0,0,851968'
                   DC    CL32'C5',F'71,0,0,0,0,65541'
BOOKS              DS    0CL20
C2                 DS    CL5
QUANITY            DS    PL2
DATES              DC    CL8'03151947'
C5                 DC    CL5'ABCDE'
*    $END
```

## Considerations

- $END must appear at the end of each subschema record.
- Subschema record names cannot exceed seven characters, because the PLEX processor appends a suffix (0) to the subschema record name.

# INCLUSION LIST

This section discusses the inclusion list for Assembler PLEX and supplements material in the main text of this manual.

An inclusion list is used to indicate the subschema items that will be used in a retrieval or update operation.

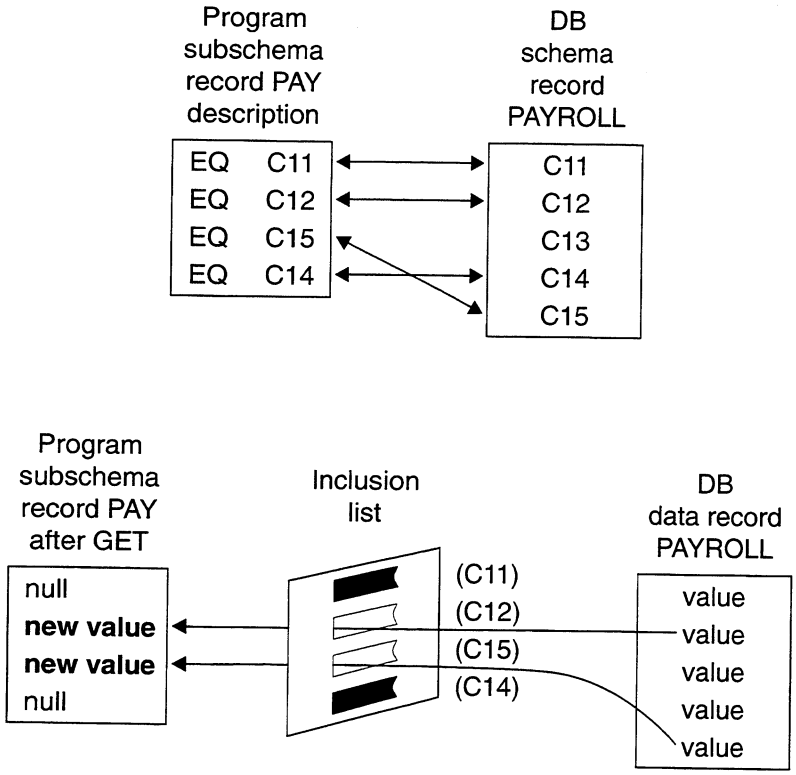
## Format

```
|item-list

|USING |designation-bit-string
       |designation-vector
```

An Assembler program can use an inclusion list with PLEX retrieval commands, such as GET1, GET, GETA, and GETD, and also with the PLEX update commands MODIFY, REMOVE, and INSERT. If you do not specify an inclusion list for a PLEX command, all items in the subschema record are affected by the command. However, if you specify an inclusion list, it indicates the items of the subschema record that will be used in the operation. Illus. D.2 on page D-14 shows the process of retrieving values with an inclusion list.

**Illus. D.2** Retrieval with an Inclusion List




In this illustration, the subschema record description consists of four of the five items in the corresponding data record. Thus, four values can be retrieved by the program. The inclusion list includes only two of those four values and passes them to the subschema record. The other two item values are screened out.

You can specify any of the three kinds of inclusion lists:

- an *item list*, which is a series of aliases of subschema items that will be affected.
- a *designation bit string*, which is a string of 0 and 1 bits corresponding to the items in the subschema record. A 1 means that the item will be affected, and a 0 means that the item will not be affected.
- a *designation vector*, which is a table of one-word indicators (containing 1 or 0), corresponding to all items in the subschema record. A 1 means that the item will be affected, and a 0 means that the item will not be affected. In each indicator, only the low-order bit is checked to see if it is 0 or 1.

An item list can only be static; that is, you code it in the PLEX command, and the PLEX processor parses it. If you want to generate an inclusion list dynamically (based, for instance, on a varying set of data), you can specify a designation bit string or vector.

The following topics discuss each of the three kinds of inclusion lists.

## Item List

**item-list**

An item list is part of the PLEX command. It is a series of aliases (separated by commas) of subschema items to be included in the command's operation. If an item list includes an item that does not belong to the specified subschema record, the PLEX processor issues an error message.

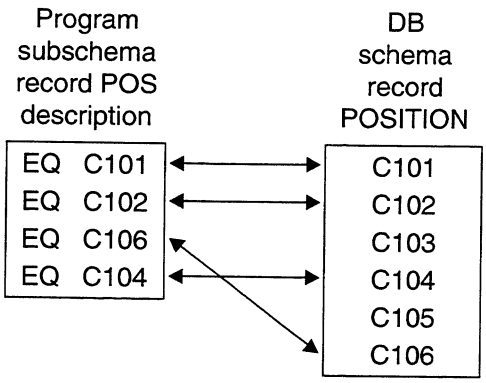
Illus. D.3 on page D-16 shows the items of the schema record called POSITION and the corresponding items of the subschema record called POS.

Four items are included in the subschema record, and, therefore, four values would ordinarily be retrieved from the corresponding data record. In this case, however, the GET1 command includes an item list.

```
GET1 POS C102, C106 NEXT;
```

According to the item list, the values for subschema items C102 and C106 will be brought into the subschema record area, and the other two values will not, as shown in Illus. D.4 on page D-16.

**Illus. D.3** Correspondence Between the Schema Record and the Subschema Record Description



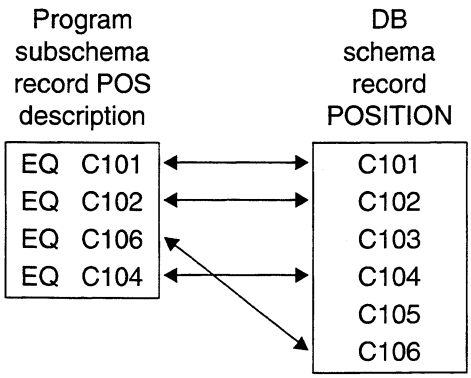

**Illus. D.4** Processing GET1 with an Item List

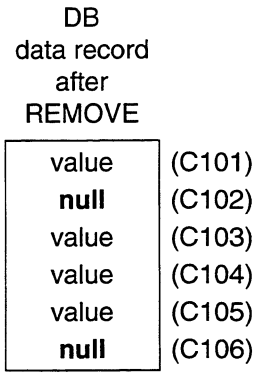
| | Program subschema record POS after GET1 |
|---|---|
| (C101) | null |
| (C102) | **new value** |
| (C106) | **new value** |
| (C104) | null |

Consider a REMOVE command with an item list. Only those items in the corresponding data record become null, as shown in Illus. D.5.

**Illus. D.5** Processing REMOVE with an Item List

**REMOVE POS C102, C106;**



## Designation Bit String

---

**USING** *designation-bit-string*

---

You specify a designation bit string in the command, following the keyword USING. Use one bit of the string for each item in the subschema record, plus two prefix bits to indicate to SYSTEM 2000 software that you are using a designation bit string.

When the command is executed, the first two bits must equal 1 and 0, respectively. The rest of the bit string must equal 0s and 1s, with the third bit corresponding to the first item in the subschema record, the fourth bit to the second item, and so on. A 0 in a bit position means that the corresponding item is will not participate in the command, and a 1 means that it will.

For example, suppose the description of SSR EDUC contains the following items.

**SSR EDUC**

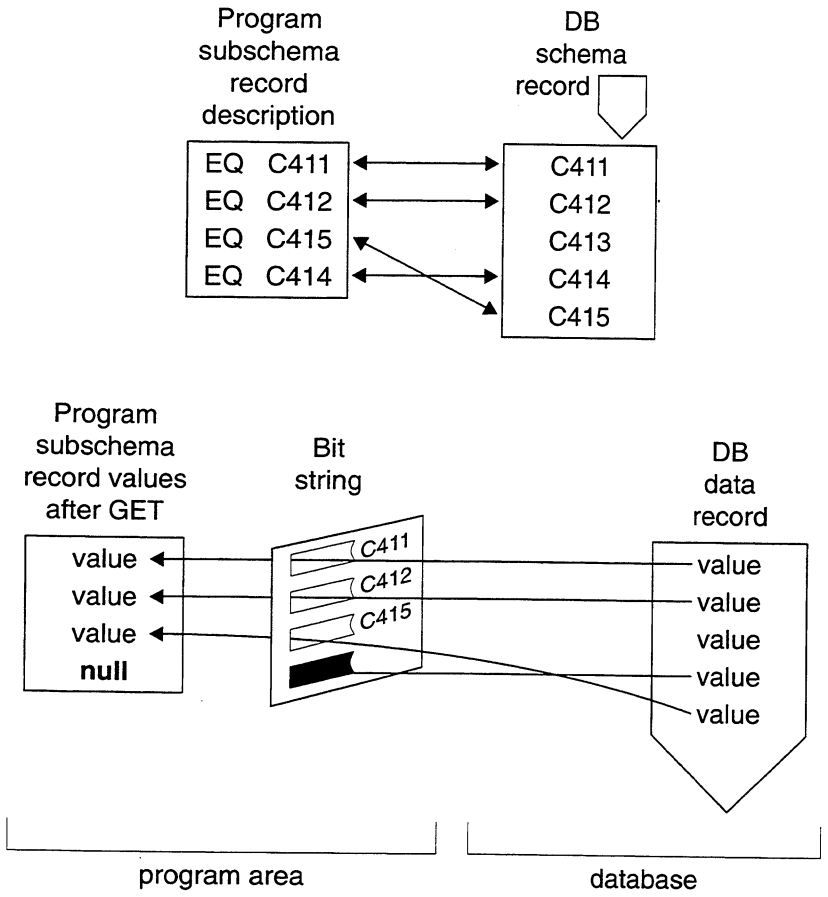
| |
|---|
| **C411** |
| **C412** |
| **C415** |
| **C414** |

You can include a designation bit string that specifies inclusion of three of the four subschema items.

```
        .
        .
        .
BITSTR DC B'10111000'
        .
        .
        .
GETD EDUC USING BITSTR NEXT
```

According to this sample designation bit string, the values for subschema items C411, C412, and C415 will be brought into the subschema record area, but the value for C414 will not, as shown in Illus. D.6.

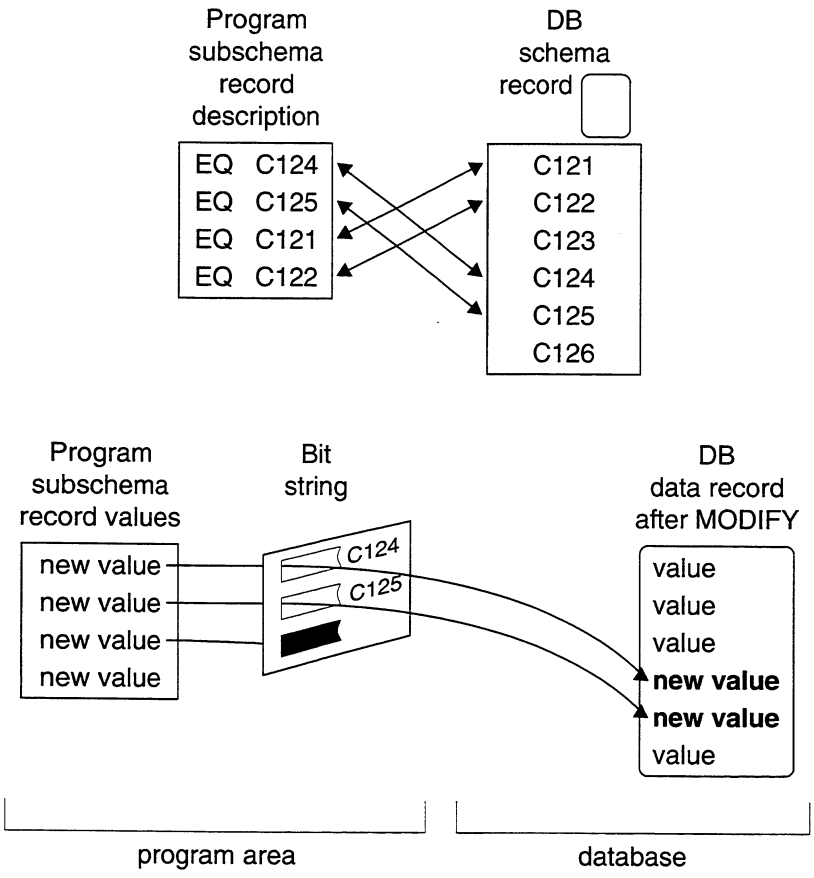
**Illus. D.6** Processing GETD with a Designation Bit String



Similarly, for a MODIFY command, you can declare a designation bit string and then use it to change two values in a data record, as shown in Illus. D.7.

**Illus. D.7** Processing MODIFY with a Designation Bit String

```
        .
        .
        .
BITSTR DC B'10110000'
        .
        .
        .
MODIFY PAYROLL USING BITSTR
```



## Designation Vector

---

USING *designation-vector*

---

Like a designation bit string, you specify a designation vector in a PLEX command, following the keyword USING. It is an array of one-word indicators, one indicator for each item in the specified subschema record. Each word contains zero (0) or one (1). In each indicator, only the low-order bit is checked to see whether it is zero or one. An item is included in the operation of the command if its indicator in the table is 1. Conversely, an item is excluded from the operation if its indicator is 0. Your program must ensure fullword alignment for the designation vector.

For example, suppose subschema record PAYROLL contains the following items:

**SSR PAYROLL**

| |
|---|
| C121 |
| C124 |
| C125 |
| C126 |
| C122 |
| C123 |

You can include a word table in the program, specifying that values for only four of the six items are to be retrieved.

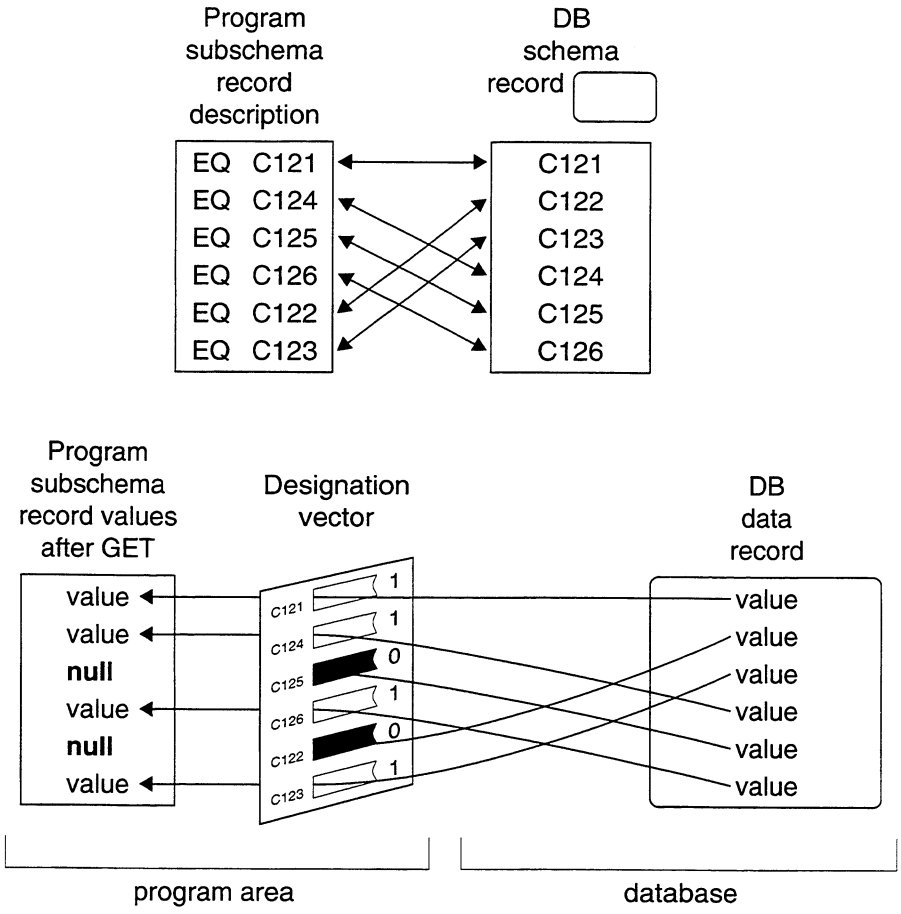
```
        .
        .
        .
VECTOR DC F'1,1,0,1,0,1'
        .
        .
        .
```

Then a retrieval command can refer to that designation vector.

```
GETA PAYROLL USING VECTOR NEXT;
```

According to this sample designation vector, the values for subschema items C121, C124, C126, and C123 will be brought into the subschema record area, but the values for C125 and C122 will not, as shown in Illus. D.8.

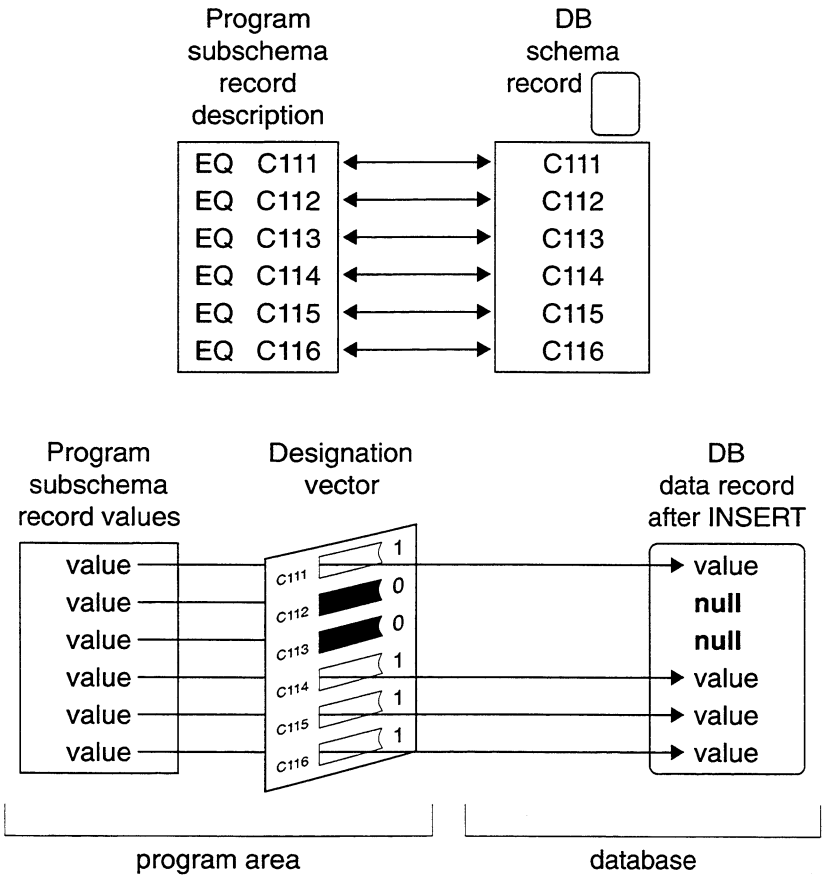
**Illus. D.8** Retrieval Processing with a Designation Vector



Similarly, for an INSERT command, you can declare a designation vector and use it to insert a new data record, as shown in Illus. D.9.

```
        .
        .
        .
VECTOR DC F'1,0,0,1,1,1'
        .
        .
        .
INSERT          USING WORD_ITEM;
```

**Illus. D.9** Processing INSERT with a Designation Vector



# USING 0 OPTION

This section discusses the USING 0 option for Assembler PLEX and supplements material in the main text of this manual.

The USING 0 option pertains to retrieval commands. It indicates that only address retrieval is to be performed, without data retrieval.

You can specify the USING 0 option in a GET1, GET, GETA, GETD, or GETP command. If you do not specify the USING 0 option in a command, two kinds of retrieval take place: retrieval of data from a data record and retrieval of the address of the data record. If you specify the USING 0 option, only the address is retrieved.

Preceding an update operation, you might want to establish a stack entry only to indicate where an update is to be performed. It might not be necessary in this case to retrieve any data from the record for that stack entry. You can specify the USING 0 option to prevent unnecessary data retrieval.

Another way in which you can retrieve an address without data retrieval is by specifying a subschema record that has no items. However, the USING 0 option is more flexible, since you can use a subschema record that has items for retrieval and update commands and then use it in other commands (with the USING 0 option) for address retrieval only.

Since the USING 0 option pertains only to the five retrieval commands listed above, its use in an update command causes the PLEX processor to issue an error message.

# PLEX PROCESSOR DIRECTIVES

## $ALLOCATE Directive

The $ALLOCATE directive instructs the PLEX processor to initialize the S2KDUM control block (see **$S2KDUM Directive** on page D-27), the COMMBLOCKs, and the subschema records that were specified as DSECT. You must set the initial values for these items before you use them. The instructions that are generated by $ALLOCATE must be executed at run time. You cannot simply include a $ALLOCATE statement in the program source statements; it must be executed before the DSECT COMMBLOCKs and DSECT schemas are referenced. CICS programs require $ALLOCATE.

### Format

```
$ALLOCATE  [blocks]
```

*blocks* is a list of S2KDUM, COMMBLOCKs, or subschemas.

$ALLOCATE without any additional parameters initializes S2KDUM and all COMMBLOCKs and subschema records that were declared with DSECT if they were not allocated before. You can allocate each item, where appropriate, by specifying a list of blocks in the $ALLOCATE directive.

The $ALLOCATE directive should be used because the PLEX processor determines the size of each of these items and generates enough MVC instructions to accomplish the correct movement and to initialize values. You must provide addressability to all items before using $ALLOCATE.

**Note:** CICS users can substitute a CICS GETMAIN or the MVS GETMAIN or place $S2KDUM COMMBLOCKs and subschema records in a DSECT. DSECT syntax must be used for reentrant programs.

**Example**

The following example illustrates the use of DSECT and $ALLOCATE:

```
Col             Col             Col
1               10              16

EXAMPLE         CSECT
                USING BLOCK,R2
BLOCK           DSECT
                $S2KDUM DSECT
                DSECT COMMBLOCK/ . . .
                DSECT SUBSCHEMA/ . . .
BLOCKEND        EQU   *

EXAMPLE         CSECT
                $BLOCKS
*                               GETMAIN STORAGE FOR ALL
                                VARIABLES
                LA              R2,BLOCKEND-BLOCK
                GETMAIN         R,LV=(R2)
                LR              R2,R1
*                               ALLOCATE ALL VARIABLES
                $ALLOCATE
                START S2K
                END
```

## $S2KDUM Directive

The $S2KDUM directive controls the placement and attributes of the S2KDUM control block.

### Format

```
$S2KDUM [DSECT]
```

When you specify the $S2KDUM directive, the PLEX processor generates the S2KDUM control block inline. If you specify the DSECT option, the PLEX processor generates a second block named S2KDUMV at the $BLOCKS location. The $ALLOCATE directive initializes the first control block with S2KDUMV when you use DSECT. If you do not use the DSECT option, the PLEX processor generates one S2KDUM control block at the $BLOCKS location. For descriptions of the various fields in the S2KDUM control block, see Illus. D.10 on page D-30.

The next page shows a sample S2KDUM table.

**Example**

The following example illustrates an S2KDUM table generated by the PLEX processor:

```
Col         Col       Col
1           10        16

S2KDUM      DS        0CL152

S2KDU1      DC        F'0'

S2KRTC      DC        F'0'

S2KCNT      DC        F'0'

S2KDU2      DC        F'0'

S2KDU3      DC        F'0'

            DC        F'5'

S2KDU4      DC        F'0'

            DC        F'0'

            DC        CL8'PROGRAM'

            DC        C'0201'

            DC        CL8'77030'

            DC        CL8'1432041'

            DC        F'0'

S2KDU5      DC        F'0'

S2KDU6      DC        F'-1'

S2KDU7      DC        CL4'    '

            DC        3F'0'

S2KDU8      DC        F'0'

S2KDU9      DC        F'0'

            DC        4F'0'

S2KLKH      DC        10F'-1'
```

The parameter list for calls to SYSTEM 2000 software at run time is built in an area called S2KPLIST. This area appears immediately after the S2KDUM control block. Reentrant programs should obtain enough storage to include this area when allocating S2KDUM.

```
          USING   XX,R2
XX        DSECT
          $S2KDUM DSECT
*                       S2KPLIST WILL APPEAR HERE
XXEND     EQU           *
          LA      R2,XXEND-XX
          GETMAIN R,LV=(R2)
          LR      R2,R1
          $ALLOCATE S2KDUM
```

**Illus. D.10** Description of S2KDUM Control Block Items

| Offset HEX | Offset DEC | Name | Length/ Alignment | Type/ Value | Field Description |
|---|---|---|---|---|---|
| 0 | 0 | S2KDU1 | 4 | Binary | SYSTEM 2000 operation code; set by the expanded PLEX code at execution time. |
| 4 | 4 | S2KRTC | 4 | Binary | SYSTEM 2000 return code; set by a PLEX command execution. |
| 8 | 8 | S2KCOUNT | 4 | Binary | Value of S2KCOUNT; set by the PLEX program. |
| C | 12 | S2KDU2 | 4 | Binary | Stack or Locate File number; set by the expanded code at run time. |
| 10 | 16 | S2KDU3 | 4 | Binary | Maximum number for Locate File; set at run time by the expanded START S2K code. Also, the number of the target stack or Locate File referred to in a COPYS or COPYL command; set at run time. |
| 14 | 20 | FILLER | 4 | Binary<br>1<br>2<br>3<br>4<br>5 | PLEX program type; set by PLEX processor.<br>FORTRAN<br>COBOL<br>PL/I - optimizing or checkout<br>PL/I (F)<br>ASSEMBLER |
| 18 | 24 | S2KDU4 | 4 | Binary<br>0<br>1 | SYSTEM 2000 Multi-User indicator; set at run time by the expanded START S2K code.<br>Single-user<br>Multi-User |
| 1C | 28 | FILLER | 4 | Alpha-numeric | If S2KDU4 is neither fullword binary zero or one, then S2KDU4 concatenated with this field contains the name of a special module to be used (left-justified and blank-filled). |
| 20 | 32 | FILLER | 8 | Alpha-numeric | Name of the program (left-justified and blank-filled); set by the PLEX processor as specified in the $NAME directive. |
| 28 | 40 | FILLER | 4 | Alpha-numeric | Release of the SYSTEM 2000 PLEX processor used; set by the PLEX processor. If changes are made to the Program Service Processor, this field identifies the PLEX processor release that is used. |
| 2C | 44 | FILLER | 8 | Alpha-numeric | Date of the PLEX processor run (YYDDD, left-justified and blank-filled); set by the PLEX processor. |
| 34 | 52 | FILLER | 8 | Alpha-numeric | Time of PLEX processor run (HHMMSSS, left-justified and blank-filled); set by the PLEX processor. |
| 3C | 60 | FILLER | 4 | Alpha-numeric | SYSTEM 2000 release number; set at execution time by SYSTEM 2000 software. |

*continued on next page*

**Illus. D.10** *continued*

| Offset HEX | Offset DEC | Name | Length/ Alignment | Type/ Value | Field Description |
|---|---|---|---|---|---|
| 40 | 64 | S2KDU5 | 4 | Binary | Reserved for internal use. |
| 44 | 68 | S2KDU6 | 4 | Binary | Reserved for internal use. |
| 48 | 72 | S2KDU7 | 4 | Alpha | Reserved for internal use. |
| 4C | 76 | FILLER | 4 | Binary | Address of run-time interface (S2KPLR). |
| 50 | 80 | FILLER | 4 | Binary | Address of SYSTEM 2000 software (S2KPLI, MUPLINT). |
| 54 | 84 | FILLER | 4 | Binary | Address of working storage; set by run-time interface. This address may be used by installation run-time exit routines S2KEX1 or S2KEX2. |
| 58 | 88 | S2KDU8 | 4 | Binary<br>0<br>1 | Snap request indicator; set by expanded code.<br>Off<br>On: SYSTEM 2000 software will provide snapshot data. |
| 5 | 92 | S2KDU9 | 4 | Binary<br>0<br>1 | Timing request indicator; set by the expanded code.<br>Off<br>On |
| 60 | 96 | S2KSRT | 4 | Binary | No-sort request indicator; set by the PLEX program.<br>0 = Sort normally.<br>1 = Do not sort. |
| 64 | 100 | | 4 | Binary | Reserved for internal use. |
| 68 | 104 | | 4 | Binary | Reserved for internal use. |
| 6C | 108 | | 1 | Bit<br>.... ...0<br>.... ...1 | Switches used at run time.<br>Link history was not modified.<br>Link history was modified. |
| 6D | 109 | | 3 | | Reserved for internal use. |
| 70 | 112 | S2KLKH | 40 | Binary | Ten full words representing LINK0 through LINK9 return codes; set by linked retrievals. Initialized by the PLEX processor to ten words of binary minus one. |

# PLEX PROCESSOR EXECUTION OPTIONS

## CICS Execution Option

The CICS execution option allows you to specify the $CICS ON directive to pass information to the SYSTEM 2000 CICS interface via the PLXCOMM control block. Your program will perform an EXEC CICS LINK to program PLXFRMT to communicate with SYSTEM 2000 software.

## Other PLEX Processor Execution Options

All execution options for the Assembler PLEX processor are the same as those described in the main text of this manual except for the FORM and SORMGIN execution options.

**FORM** If you specify the FORM execution option, a hyphen in column 1 of a line causes the PLEX processor to process the line. All other input is passed directly to the Assembler. (See also **Format of PLEX Commands** on page D-2.)

**SORMGIN** (*a*,*b*,*c*,*d*)

*a* is the start position of the label field. The default is column 1.

*b* is the start position of the operation field. The default is column 10.

*c* is the start position of the operand field. The default is column 16.

*d* is the start position of the source identification field. The default is column 73.

Scanning begins in column *a*. The continuation character is in column *d*-1 (column *d* minus 1). Assembler code is generated beginning in column *b* (operation) and column *c* (operands).

# SAMPLE ASSEMBLER PLEX PROGRAMS AND JCL

The SYSTEM 2000 delivery tape contains sample JCL for preprocessing, assembling, link-editing, and executing an Assembler PLEX program. The Assembler PLEX validation program provided on the SYSTEM 2000 delivery tape is a sample Assembler PLEX program.

# Index

**Special Characters**
$ALLOCATE directive D-25
$BLOCKS directive 19-2, B-57, C-34
  sample code generated for 19-3, 19-6, B-58, C-34
$CFOR directive
  format 12-17
$CICS directive 19-2, 19-7, B-61
  command level B-61
  macro level B-61
$COPY directive 19-7, B-61, C-37, C-67
  library, JCL for 20-29, B-94, C-67
$COPY library 20-29, B-94, C-67
$END for Assembler PLEX D-12
$FOR directive 12-11, 12-17
  format 12-12
$NAME directive 19-8, B-62, C-38
$STORAGE directive B-63
  examples B-64
$S2KDUM directive 19-9, D-27
$TIME command 15-1
  considerations 15-3
  format 15-1
  in queue mode 15-3
  output from 15-6
*PROCESS statement B-62

**A**
abnormal job termination and clearing database pages 10-85
access authorities 77
  assigning 10
adding records 19-13
additional Locate Files
  Multi-User environment 20-28
  single-user jobs 20-28, B-93, C-66
address retrieval with USING 0 8-131
AFTER in INSERT command
  in immediate mode 10-33
    examples 10-37
  in load mode 10-66
    examples 10-69
  in queue mode 10-53
    examples 10-57
alias of a subschema item 5-22, 5-32, 5-33, B-32, B-41, B-42, B-43, C-22, C-24, C-32
allocating dynamically
  database files 6-8
  the Keepfile 11-4, 11-9, 11-12
  the Savefile 11-4, 11-9
ancestor record
  defined 12
  retrieving 33

AND operator in where-clause 36, 37, 40, 7-15
example 7-17, 7-20 *to* 7-21, 7-30 *to* 7-31, 7-32, 7-33
APPLY command 69, 11-16
and Coordinated Recovery 11-17
considerations 11-20
examples
of syntax 11-17
of usage 11-18
exclusive use of database required 11-20
format 11-17
archival copy of database
restoring 69
saving 67
Assembler D-1
$ALLOCATE directive D-25
$END D-12
$S2KDUM directive D-27
COMMBLOCK
declarations D-7
DSECT option D-7
example D-7
format D-7
comments, statements, and PLEX commands D-2
data forms and item types D-3
table of D-3
designation bit string D-18
designation vector D-21
directives, PLEX processor D-25
execution options, PLEX processor D-32
FORM execution option D-2, D-32
format of PLEX commands D-2
inclusion list D-13
examples D-14
item list D-15
JCL D-32
NOFORM execution option D-2
PLEX processor
directives D-25
execution options D-32
PLEX program, example D-32
SORMGIN execution option D-32
subschema record D-9
considerations D-12
examples D-11
format D-9
name of D-12
S2KDUM control block D-30
USING 0 option D-24
AT operator in where-clause 7-23
considerations 7-26
example 7-22, 7-24
processing for 7-24
relative position under parent 7-23
used with two variables 7-26

attributes *See* PL/I attributes and item types
authorities, access 77
    assigning 10
AUTOMATICALLY command *See* CLEAR commands
available space 9-2

**B**
BEFORE in INSERT command
    in immediate mode 10-33
        examples 10-37
    in load mode 10-66
        examples 10-69
    in queue mode 10-53
        examples 10-57
before-images 11-30
binary operators in where-clause 7-11
    examples 7-12
    in item-to-item comparisons 7-13
bit string, designation *See* designation bit string
blanks, trailing, and TEXT item type 4-9, B-14, C-10

**C**
C-numbers, using in subschema record 5-22, 5-24, 5-32, 5-33, B-31, B-42, B-43, C-22, C-24, C-32
CANCEL QUEUE command 9-2, 9-4, 9-6, 9-7
    example of usage 9-7
    format 9-7
cascaded retrieval 49
CHAR attribute, for PL/I
    and CHAR item type B-8
    and DATE item type B-8
    usage, table of B-16
CHARACTER (CHAR) item type 4-2, 4-10, B-8, B-16, C-7, D-3
    in where-clause 4-8, B-13, C-8
CHARACTER variable type, for FORTRAN
    and CHARACTER item type C-7
    and COMMBLOCK fields C-12
    and DATE item type C-7
    and subschema item declarations C-23, C-24, C-30
    and TEXT item type C-7
    and UNDEFINED item type C-7
    usage, table of C-7
child record, defined 12
CICS
    command-level linkage 22-7
    execution option 19-10, B-70
    LANG parameter 19-10
    PLEX processor directives for 19-2, 19-9
    pseudo-conversational transactions 22-9
CLEAR AUTOMATICALLY *See* CLEAR commands
CLEAR commands 64, 10-81
    considerations 10-85
    example 10-84
    format 10-83

in frame of update commands 10-85
use of 64
CLEAR UPDATE LOG AUTOMATICALLY command *See* CLEAR commands
CLEAR UPDATE LOG command *See* CLEAR commands
clearing of updated pages 64, 11-33
CLOSE command 86, 88, 17-1
abnormal end of frame 11-33, 17-3
and FRAME command 17-3
considerations 17-3
example 17-2
format 17-1
in immediate and queue modes 17-3
closing a database 88
COBOL
comments, statements, and PLEX commands 3-3
data descriptions and item types 9, 4-1
in where-clause 4-7
table of 4-10
PLEX program
FINDORG 20-3
structure 3, 88, 20-2
COBOL II
LANG parameter 19-10
commands
allowed in a frame 11-32
and Assembler comments and statements D-2
and COBOL comments and statements 3-3
and FORTRAN comments and statements C-3
and PL/I comments and statements B-3
format, general 3-1, B-3, C-3, D-2
processing in execution modes 61
return codes from, and FOR processing 12-5, B-55
COMMBLOCK 7, 5-1, B-19, C-11, D-7
considerations 5-19, B-30, C-20
cycle date in 5-7, B-24, C-15
cycle number in 5-7, B-24, C-15
cycle time in 5-7, B-23, C-14
damage flag in 5-7, B-24, C-15
DSECT option for Assembler language D-7
examples 5-9, B-25, C-16, D-7
fields 5-3, 5-19, B-20, B-21, C-12
and GETP command 5-4, 5-20, B-22, C-13
and OPEN command 6-8
for Locate Files 5-20, B-30, C-20
format 5-2, B-20, C-12, D-7
in LINKAGE SECTION 5-1, 5-2, 5-19
level and ordinal fields in 5-5, B-23, C-14
level numbers
in COBOL description 5-2, 5-13
in PL/I description B-20, B-28
LINKAGE SECTION, example 5-15
Locate File pointer in 5-5, B-23, C-14
PARAMETER option B-20, B-26, C-19
parent record field in 5-8, 5-11, B-24, B-27, C-15

password in 5-4, B-22, B-27, C-13
REDEFINES and RENAMES clauses in 5-2, 5-19
example 5-17, 5-21
return code in 5-3, B-21, C-13
subschema record name in 5-3
system separator in 5-7
COMMIT command 74, 11-42
considerations 11-44
example 11-43
format 11-42
communications between a program and a database 6
COMP usage
and DECIMAL and MONEY item types 4-6, 4-11
and INTEGER item type 4-4, 4-9, 4-10
COMP-1 usage and REAL and DOUBLE item types 4-7, 4-11
COMP-2 usage and REAL and DOUBLE item types 4-7, 4-11
COMP-3 usage
and DECIMAL and MONEY item types 4-5, 4-9, 4-11
and INTEGER item type 4-3, 4-9, 4-10
component name
defined 16
in subschema record 5-22, 5-33, B-31, B-41, B-42, C-22, C-31, C-32
component number, defined 16
CONDITIONAL in FRAME command 11-32, 11-37
constructing a where-clause 7-30
continuation of PLEX commands 3-3, C-4
Coordinated Recovery 72, *See also* logical unit of work, synchpoints
and clearing database pages 64
defined 11-27
examples 75
use of 72
with APPLY command 11-17
with CLEAR commands 10-85
with COMMIT command 74, 11-42
with ENABLE/DISABLE ROLLBACK commands 11-27
with FRAME and END FRAME commands 11-29
with OPEN command 11-31
with RESET ROLLBACK command 11-48
with ROLLBACK command 11-45
with suspended update logging 11-24
COPY execution option 19-11, B-70, C-39
COPYL command 43, 8-147
considerations 8-150
examples
of syntax 8-148
of usage 8-149
format 8-147
COPYS command 27, 8-147
considerations 8-150
examples
of syntax 8-148
of usage 8-149
format 8-147

cursor in PLEX processor messages 22-15
cycle date 5-7, B-24, C-15
cycle number 67, 5-7, B-24, C-15
cycle time 5-7, B-23, C-14

**D**

damage flag 65, *See also* Coordinated Recovery
  defined 10-81
  in COMMBLOCK 5-7, B-24, C-15
  with FRAME command 11-31, 11-36
damaged database 65, *See also* Coordinated Recovery
  and immediate mode insert sequence 10-37
  and queue mode updates 9-4
  recovering 11-10
data forms, for Assembler PLEX D-3
data records 13
  addresses of 24, 26
  hierarchical arrangement for loading 21-2
  inserting 57
  physical and logical order of 16
  removing 57, 59
data set name *See* DSN
Data Table 16, 62
  full passes of 7-34
  processing in load mode 10-66
  reusable space and REMOVE TREE command 10-24
  space, use of in updating 62
data types *See* item types
data values
  changing 54, 56
  defined 13
  key, and Distinct Values Table 13-1
  removing 54, 55
  retrieving 24
  types of *See* item types
database
  archival copy 67, 11-3
  closing 88, 17-1
  cycle date 5-7, B-24, C-15
  cycle number 67, 5-7, B-24, C-15
  cycle time 5-7, B-23, C-14
  damage 65
    recovery from 11-10
  damage flag in COMMBLOCK C-15
  exclusive use of 78, 6-7
  files
    allocating dynamically 6-8, B-94
    JCL for 20-29, B-94, C-64, C-67
  loading 21-1
    error handling during 21-6
    recovery during 21-8
  logical view 16
  maximum number available to program 10, 86, 6-12
  multiple 86

navigation 26
non-exclusive use of 78
opening 10, 6-7
with FRAME command 11-31
pages, clearing 64
with CLEAR commands 10-81
with FRAME and END FRAME commands 11-37
recovery 67, *See also* Coordinated Recovery
during loading process 21-8
with Update Log 69
restoring 67, 11-8, 21-8
saving 67, 11-2
schema 14, 21-2
system separator in COMMBLOCK 5-7
Database Administrator password *See* DBA password
date format of a subschema item 5-23, 5-24, 5-34, B-32, B-33, C-23, D-10
DATE item type 4-3, 4-10, B-8, B-16, C-7, D-3
in where-clause 4-7, B-13, C-8
DBA password 77
commands authorization required for
APPLY 11-16, 11-20
ENABLE/ DISABLE ROLLBACK 11-28
ENABLE/DISABLE VALUES PADDING 13-1, 13-5
FULL PASSES 14-1
RELEASE 11-26
RESET ROLLBACK 11-48
RESTORE 11-10
SAVE 11-7
SUSPEND 11-24
DECIMAL item type 4-5, 4-11, B-11, B-17, D-3
in where-clause 4-7, B-13, C-10
DECK execution option 19-11, B-71, C-39
declarations of COMMBLOCKs and subschema records 5-1, B-19, B-31, C-11, C-22, D-7, D-9
descendant record
defined 12
retrieving 29
designation bit string B-45, B-49, D-13, D-18
and LINK command B-55
examples B-50, D-18
designation vector B-45, B-52, D-13, D-21
examples B-52, D-21
diagnostic messages, PLEX processor 19-14
directives, PLEX processor 19-2, B-57, C-34, D-25
DISABLE ROLLBACK command 72, 11-27
and exclusive use 11-28
considerations 11-28
format 11-27
DISABLE ROUTINE command 16-1
considerations 16-2
format 16-1
DISABLE VALUES PADDING command 13-1
considerations 13-5
format 13-4
in queue mode 13-5

DISPLAY usage
with CHAR item type 4-2, 4-10
with DATE item type 4-3, 4-10
with DECIMAL and MONEY item types 4-6, 4-11
with INTEGER item type 4-5, 4-10
with TEXT item type 4-2, 4-10
with UNDEFINED item type 4-7, 4-12
Distinct Values Table
adding data values to 13-2
padding 13-2
domain of GETD command 33
DOUBLE item type 4-7, 4-11, B-12, B-18, C-7, D-3
in where-clause C-10
DROP HOLD command 8-144
considerations 8-146
examples 8-144
format 8-144
DSECT option for Assembler PLEX D-7
DSN
in RESTORE command 11-8
in SAVE command 11-3, 11-7
dynamic allocation of
database files 6-8
the Keepfile 11-4, 11-9, 11-12
the Savefile 11-4, 11-9
DYNAMIC where-clause 38, 7-35
advantages of 7-35
considerations 7-38
error handling in 7-40
examples 7-36
format 7-35
how to code 7-39
opcodes for 7-39

**E**

ECHO execution option 19-11, B-71, C-40
efficiency considerations 22-2
ENABLE ROLLBACK command 72, 11-27
and exclusive use 11-28
considerations 11-28
format 11-27
ENABLE ROUTINE command 16-1
considerations 16-2
format 16-1
ENABLE VALUES PADDING command 13-1
considerations 13-5
format 13-4
in queue mode 13-5
END FRAME command 72, 11-29
examples 11-39
format 11-29
function of 11-29
rejected, effects of 11-35

END PROCEDURE directive 19-8, C-37
environments for running a PLEX program 4
EQ operator in where-clause 7-11, 7-13
    and GETP command 8-152
    example 7-12, 7-14
EQ, in subschema item description 5-22, 5-23, B-33, C-22, C-24
errors
    and clearing updated pages 64
    guidelines for avoiding 22-13
    handling during database loading 21-6
ERRORS execution option 19-12, B-72, C-40
ERRSYM execution option 19-12, B-72, C-40
example of a PLEX program 20-1, B-82, C-51, C-52, D-32
EXCLUSIVE option in OPEN command 78, 87, 6-7
exclusive use of a database 78, 87, 6-7, 6-11, 6-12, 6-13
    and FRAME command 11-36
execution modes 61, 9-1
    table space for updates in 62
    with FRAME and END FRAME commands 11-34
execution options, PLEX processor 3-2, 19-10, B-70, C-39, D-32
execution-time PLEX exits 22-7
EXIST *See* EXISTS operator in where-clause
EXISTING *See* EXISTS operator in where-clause
EXISTS operator in where-clause 7-10
EXIT execution option 19-12, B-73, C-41
exit routine for PLEX processor 19-12
    diagnostic messages 19-14
exits
    execution-time 22-7
    user (Universal Software Interface) 4, 16-1
expressions in where-clause 7-2
Extended Field Table 4-2, B-7

**F**

FAIL *See* FAILS operator in where-clause
FAILING *See* FAILS operator in where-clause
FAILS operator in where-clause 7-10
family of a record 12
files for database and Locate Files, JCL for 20-29, B-93, C-63
FINDORG program
    COBOL 20-3
FINDORG program, sample
    COBOL
        flowchart 20-4
        output 20-24
        source statements 20-14
    FORTRAN C-52
    PL/I B-83
        input source file B-84
        output B-89
FINDORG sample program
    FORTRAN
        input source file C-53
        output C-62

FIRST
    and GET command 8-109
    and GETD command 8-64
    and GET1 command 8-22
FIXED BINARY attribute, for PL/I
    and DECIMAL and MONEY item types B-11
    and INTEGER item type B-10
    usage, table of B-16
FIXED DECIMAL attribute, for PL/I
    and DECIMAL and MONEY item types B-11
    and INTEGER item type B-9
    usage, table of B-16
fixed where-clause 7-2
FLOAT BINARY attribute, for PL/I
    and DECIMAL and MONEY item types B-12
    and INTEGER item type B-10
    usage, table of B-16
FLOAT DECIMAL attribute, for PL/I
    and DECIMAL and MONEY item types B-12
    and INTEGER item type B-10
    usage, table of B-16
FOR command 12-5, B-55
    and NAME or USE option, format B-56
    and return codes with FRAME command 11-34
    considerations 12-9
    examples B-56
        of syntax 12-7
        of usage 12-10
    format 12-6, B-55
    levels of 12-8
FOR, $FOR, and $CFOR 12-1
FORM execution option 3-2, 19-16, B-3, B-77, C-3, C-45, D-2, D-32
FORM, for date format of a subschema item 5-23, 5-24, 5-34
format of PLEX commands, general 3-1, B-3, C-3, D-2
FORTRAN C-2
    $BLOCKS directive C-34
    $COPY directive C-37
    $NAME directive C-38
    additional Locate Files, single-user C-66
    COMMBLOCK
        declarations C-11
        examples C-16
        fields C-12
        format C-12
    comments, statements, and PLEX commands C-3
    COPY/NOCOPY execution option C-39
    DECK/NODECK execution option C-39
    directives, PLEX processor C-34
    ECHO/NOECHO execution option C-40
    END PROCEDURE directive C-37
    ERRORS/NOERRORS execution option C-40
    ERRSYM execution option C-40
    execution options, PLEX processor C-39
    EXIT execution option C-41

FINDORG program, sample C-52
FORM/NOFORM execution option C-3, C-45
format of PLEX commands C-3
GEN/NOGEN execution option C-45
ITEM execution option C-46
JCL
for PLEX jobs C-63
variations C-66
Keepfile, allocating C-67
LANG execution option C-46
LINECNT execution option C-46
LOAD/NOLOAD execution option C-47
MEMORY execution option C-47
Multi-User environment, JCL for C-63
PLEX processor
directives C-34
execution options C-39
PLEX program structure C-51
PROGRAM directive C-38
Rollback Log, allocating C-67
Savefile, allocating C-67
single-user PLEX program, JCL for C-63
SORMGIN execution option C-48
SOURCE/NOSOURCE execution option C-48
SSR execution option C-49
subschema record C-22
considerations C-31
examples C-25
format C-23
SYSSEP execution option C-49
TERM/NOTERM execution option C-49
TRANS execution option C-50
Update Log, allocating C-67
variables and item types
table of C-7
where-clause items C-8
FRAME command 72, 11-29
and clearing updated pages 11-37
and HOLD option 11-37
card continuation 11-38
causing recovery with 11-30, 11-31
clearing of database pages 11-37
examples 11-38 *to* 11-39
format 11-29
opening a database with 11-31, 11-36
rejected, effects of 11-35
frame of commands *See also* Coordinated Recovery, logical unit of work
abnormal end with CLOSE command 11-33, 17-3
acceptable commands in 11-32
clearing updated pages 11-37
FULL PASSES command 14-1
and where-clause 7-34
and where-clauses 14-1
considerations 14-1

format 14-1

**G**

GE operator in where-clause 7-11
and GETP command 8-152
example 7-12
GEN execution option 19-16, B-77, C-45
GET command 43, 8-100
and COMMBLOCK fields 8-104, 8-105
considerations 8-104
examples 44
of syntax 8-102
of usage 8-103
FIRST 8-109
example 8-109
format 8-101
LAST 8-110
examples 8-110
NEXT 8-107
example 8-108
PRESENT 8-105, 8-115
example 8-116
PREVIOUS 8-104, 8-113
example 8-114
S2KCOUNT 8-105, 8-117
examples 8-117
use of 22-3, 22-5
GETA command 33, 8-47
considerations 8-50
examples 33, 8-51
of syntax 8-48
of usage 8-49
format 8-47
GETD command 29, 8-54
considerations 8-58
domain of 8-54
examples 29
of syntax 8-56
of usage 8-57
FIRST 8-64
examples 8-64
format 8-55
LAST 8-67
examples 8-67
NEXT 8-60
examples 8-60
PRESENT 8-70
examples 8-70
S2KCOUNT 8-73
examples 8-73
usage of 29, 22-5
GETP command 48, 8-151
considerations 8-155
examples 8-156
of syntax 8-154

of usage 8-154
format 8-153
GET1 command 28, 48, 8-5
and DYNAMIC where-clause 7-35
and no-sort capability A-1
considerations 8-9
examples 28, 48
of syntax 8-7
of usage 8-8
FIRST 8-22
examples 8-22
format 8-6
LAST 8-26
examples 8-27
NEXT 8-12
examples 8-13
PLEX where-clause 8-44
examples 8-45
PRESENT 8-34
examples 8-34
S2KCOUNT 8-39
examples 8-39
usage of 22-3
where-clause in 48
global hold 80, 82, 8-134
obtaining and ending 8-134
with FRAME command 11-29, 11-31, 11-35
GT operator in where-clause 7-11
and GETP command 8-152
example 7-12

**H**

HAS operator in where-clause 38, 41, 7-8, 7-19
example 7-20, 22-4
hierarchical structure, illustration of 12, 15
Hierarchical Table 16
processing in load mode 10-66
reusable space and REMOVE TREE command 10-24
space, in updating 62
HOLD option in retrieval commands 80, 8-131, *See also* global hold, local hold, multiple holds
and $TIME output 15-3
and FRAME command 11-35, 11-37
considerations 8-137
examples of usage 81
in immediate mode 80, 8-135, 8-142
in queue mode 80, 8-134, 8-143
use of, in a Multi-User environment 8-134
holds *See* global hold, local hold, multiple holds

**I**

immediate mode 62, 9-1
INSERT command in 10-31
table space for updates in 62

IMMEDIATE, in FRAME command 11-32, 11-37
implicit opening, defined 6-11
inclusion list 23, 8-121, B-45, D-13
    examples 8-122
        usage, Assembler PLEX D-14
    format 8-121, B-45, D-13
    in LINK command 8-172, B-55
    kinds of 8-122, B-46, D-15
input sequence in loading 21-3
INSERT command 57, 10-28, *See also* insert sequence
    considerations for all execution modes 10-30
    examples 60
        of syntax 10-29
    format 10-28
    in immediate mode 62, 10-31
        considerations 10-35
        effects of REMOVE TREE 10-36
        examples of usage 10-34, 10-37
    in load mode 62, 10-66
        and stacks and Locate Files 10-66
        considerations 10-68
        example of usage 10-67, 10-69
    in queue mode 62, 10-52
        considerations 10-55
        examples of usage 10-54, 10-57
insert sequence
    hierarchical order required 10-30
    in immediate mode
        and damaged database 10-37
        and local hold 10-35
        and MOVE TREE command 10-35
        and REMOVE TREE command 10-35
        and stack entries 10-35
        considerations 10-35
        defined 10-32
        ending 10-32
        examples 10-37
        processing 10-32
    in queue mode 10-52
        and global hold 10-55
        considerations 10-55
        defined 10-52
        ending 10-52
        examples 10-57
        processing 10-52
INTEGER item type 4-3, 4-10, B-9, B-16, C-7, D-3
    and nulls B-14
    in where-clause 4-7
item alias
    in subschema item 5-32, 5-33
    in subschema record 5-22, B-31, B-41, B-42, B-43, C-22, C-24, C-32
ITEM execution option 19-17, B-78, C-46

item list 8-122, B-45, B-47, D-13, D-15
examples 8-122, B-47, D-15
item types
and Assembler data forms D-3
table of D-3
and COBOL data descriptions 8, 4-1
table of 4-10
where-clause items 4-7
and FORTRAN variable types C-7
table of C-7
where-clause items C-8
and null items 4-10, B-16, C-7, D-3
and PL/I attributes B-7
table of B-16
where-clause items B-13
and return codes 4-9, B-15, C-10
item-to-item comparisons in where-clause 7-13

**J**

JCL 20-25, B-90, C-63, D-32
for FORTRAN PLEX jobs C-63
for PL/I PLEX jobs B-90
Multi-User environment 20-25, B-90, C-63
sample for database files and Locate Files 20-26
single-user environment 20-25, B-90, B-91, C-63
variations for PLEX programs 20-28, B-93, C-66
Job Control Language *See* JCL
job steps for PLEX programs 89, C-63

**K**

KEEP command 69, 11-12
considerations 11-15
examples
of syntax 11-13
of usage 11-14
format 11-12
Keepfile 69
allocating B-94, C-67
dynamically allocating 11-4, 11-9, 11-12
in SAVE command 11-3, 11-4, 11-7
JCL for 20-29
with APPLY command 11-16, 11-17, 11-20
with KEEP command 11-12
with SUSPEND command 11-22, 11-24
key conditions in where-clause 7-2, 7-3, 7-4, 7-32, 7-33, 7-34
key items in Distinct Values Table 13-1
keywords, PLEX 3-3

**L**

LANG execution option C-46
LANG parameter, COBOL II 19-10
LAST
and GET command 8-110
and GETD command 8-67

and GET1 command 8-26
LE operator in where-clause 7-11
and GETP command 8-152
example 7-12
level and ordinal fields in COMMBLOCK 5-5, B-23, C-14
LHOLD Multi-User execution parameter 8-137
LINECNT execution option 19-17, B-78, C-46
LINK command 49, 87, 8-163
and S2KLKH array 8-169
considerations 8-165
examples of syntax 8-165
format 8-164
HOLD option 8-137, 8-172
inclusion list 8-172, B-55
infinite loops, preventing 8-168
link history array (S2KLKH) 8-169
order of retrieval in 8-167
processing 8-166
retrieval cascade
example 8-167
order of retrieval in 8-167
return codes and S2KLKH array 8-169
source control 8-170
target control 8-171
via-clause 8-169
linked retrieval 48, 49, *See also* LINK command
Linking requirements for CICS COBOL II 22-7
links, number of, established in PLEX program 49
LOAD command 62, 9-5
considerations 9-5
format 9-5
LOAD execution option 19-17, B-78, C-47
load mode 62, 9-5
database recovery 21-8
error handling in 21-6
explained 10-66
input sequence 21-3
Locate Files 10-66
stacks in 10-66
table space for updates in 62
techniques 21-6
use of 21-1
loading a database 62, 21-1, *See also* load mode
program for, testing 21-8
local hold 80, 84, 8-131
and DROP HOLD command 8-144
and insert sequence in immediate mode 10-35
and MODIFY and REMOVE commands 8-137
deadlock, resolving 8-139
obtaining and ending 9-4
LOCATE command 42, 87, 8-81
and DYNAMIC where-clause 7-35
and no-sort capability A-1
considerations 8-84

examples 42
of syntax 8-83
of usage 8-83
format 8-81
usage 22-3, 22-4, 22-5
with exclusive use of a database 78
Locate File pointer in COMMBLOCK 5-5, 8-104, B-23, C-14
Locate Files 10, 42
additional Locate Files
Multi-User environment 20-28
single-user jobs 20-28, B-93, C-66
and COMMBLOCK fields 5-20, 8-85, B-30, C-20
and ORDER BY command 8-92, 8-95
and REMOVE TREE command 8-84
and SAME LOCATE in where-clause 7-27
and stack for same database 8-11
canceled by CLOSE command 17-1
copying 43, 8-147 *to* 8-149, 8-150
for multiple databases 87
JCL for (single-user) B-93, C-66
multiple 43, 87
number of, in START S2K command 6-1, 6-6, B-44
LOCK option in OPEN command 6-7, *See also* EXCLUSIVE option in OPEN command
logical operators AND and OR in where-clause 7-32
logical order of records 16, 21
logical unit of work 72, *See also* Coordinated Recovery
and COMMIT command 11-42
and FRAME/END FRAME commands 11-30
and RESET ROLLBACK command 11-48
and ROLLBACK command 11-45
canceling 11-45
committing 72, 11-42
defined 72, 11-30
LT operator in where-clause 7-11
and GETP command 8-152
example 7-12
LUW *See* logical unit of work, Coordinated Recovery

**M**

main memory and the PLEX processor 22-16
master password 77
commands requiring
APPLY 11-16, 11-20, 11-24
ENABLE/DISABLE ROLLBACK 11-28
FULL PASSES 14-1
RELEASE 11-26
RESTORE 11-10
SAVE 11-7
SUSPEND 11-24
VALUES PADDING 13-5
memory and the PLEX processor 22-16
MEMORY execution option 19-17, B-79, C-47

messages
DYNAMIC where-clause, error handling 7-40
from the PLEX processor 22-14
long and short formats 22-14
severity codes 22-15
MODIFY command 54, 56, 10-11
and local hold 10-15, 10-21
considerations 10-14
examples 10-16
of syntax 10-12
of usage 10-13
for key update 10-16
for non-key update 10-16
format 10-12
modifying records 19-13
module (overlay) structure of SYSTEM 2000 software 15-2
MONEY item type 4-5, 4-11, B-11, B-17, C-7, D-3
in where-clause 4-7, B-13, C-10
MOVE TREE command 61, 10-75
and GET command 8-106
and GETA command 8-50
and GETD command 8-58
and GET1 command 8-10
considerations 10-77
examples 61, 10-78
format 10-77
Multi-User environment
and START S2K command 6-2, 6-6, B-44
and STOP S2K command 17-4, 17-5
holds in 8-133
JCL for 20-25, C-63
number of Locate Files available in 6-6
Multi-User PLEX program, modules for 92
multiple databases 86
multiple holds 80, 8-131, *See also* HOLD option in retrieval commands, local hold
and GET command 8-106
and GETA command 8-50
and GETD command 8-59
and GETP command 8-156
and GET1 command 8-11
and global holds 11-31
and LINK command 8-165
multiple Locate Files 43, 87
Multiple Occurrence Table, padding for 13-1, 13-4
multiple stacks 27

**N**

N-authority 77
name table 8-122, 8-125
examples 8-126
navigation through database 26
NE operator in where-clause 7-11, 7-13
example 7-12, 7-14

NEXT
and GET command 8-107
and GETD command 8-60
and GET1 command 8-12
NO FULL PASSES command 7-34, 14-1
considerations 14-1
format 14-1
no-sort capability 42, A-1
use of 22-6
NOCOPY execution option 19-11, B-70, C-39
NODECK execution option 19-11, B-71, C-39
NOECHO execution option 19-11, B-71, C-40
NOERRORS execution option 19-12, B-72, C-40
NOFORM execution option 3-2, 19-16, B-3, B-77, C-3, C-45, D-2
NOGEN execution option 19-16, B-77, C-45
NOLOAD execution option 19-17, B-78, C-47
non-key conditions in where-clause 7-2, 7-3, 7-4, 7-32, 7-33, 7-34
normalization 38, 7-15, 7-16, 7-32
defined 38
NOSOURCE execution option 19-19, B-79, C-48
NOT operator in where-clause 36, 37, 7-18
examples 7-31
not allowed with SAME expression 7-28
NOTERM execution option 19-20, B-80, C-49
null values
and item types 4-9, 4-10, B-16, C-7, C-10, D-3
and REMOVE command 10-2
defined 54
retrieving 4-9, B-16, C-10, D-3

**O**

opcodes for PLEX commands 15-1
list of 15-4
OPEN command 10, 86, 6-7
and damage flag 6-8
and dynamically allocating database files 6-8
and load mode recovery 6-13
and other COMMBLOCK fields 6-8
and rollback processing 6-8, 6-12
considerations 6-11
examples
of syntax 6-9
of usage 6-10
for exclusive use 6-7
format 6-7
opening *See also* OPEN command, OPENR command
a database with FRAME command 11-31, 11-36
a subschema record 4-4, 5-32, B-42, C-32
OPENR command 6-7, 6-12
operands in where-clause 4-7, 4-8, 7-9, B-13, C-8
maximum number allowed 7-8
operators in where-clause
maximum number allowed 7-8
precedence 7-4, 7-30

processing 7-32
options, execution, for PLEX processor 19-10, B-70, C-39, D-32
OPT039 execution parameter 4-7, 4-11, B-12, B-17, D-5
OPT040 execution parameter 4-7, 4-11, B-12, B-17, D-5
OR operator in where-clause 36, 37, 7-15
example 7-17
ORDER BY command 43, 8-92
and Locate File addresses 8-95
considerations 8-95
examples
of syntax 8-93
of usage 8-94, 8-97
format 8-92
overlay (module) structure of SYSTEM 2000 software 15-2

**P**

padding for Distinct Values Table 13-2
pages, clearing 64
parent record
defined 12
field in COMMBLOCK 5-8, 5-11, 22-6, B-24, C-15, C-17
retrieving
with GETA command 33, 8-47
with GET1 command 8-34
password 77, 86
assigning authorities for 10
DBA 77
for opening a database 6-12
in COMMBLOCK 5-4, 5-11, B-22, B-27, C-13
master 77
secondary 77
path of a record 12
physical order of records 16
PICTURE attribute, for PL/I
and CHAR item type B-8
and DATE item type B-8
and DECIMAL and MONEY item types B-12
and INTEGER item type B-9, B-10
and TEXT item type B-7
usage, table of B-16
PL/I B-2
$BLOCKS directive B-57
$CICS directive B-61
$COPY directive B-61
library, JCL for B-94
$NAME directive B-62
$STORAGE directive B-63
*PROCESS statement B-62
additional Locate Files, single-user B-93
attributes and item types *See* CHAR attribute, FIXED BINARY, FIXED DECIMAL, FLOAT BINARY, FLOAT DECIMAL, PICTURE, B-7
table of B-16
where-clause items B-13
attributes, usage table of B-16

CHAR attribute for CHAR item type B-8
CHARACTER item type B-8
CICS execution option B-70
COMMBLOCK
    considerations B-30
    declarations B-19
    examples B-25
    fields B-21
    format B-20
comments, statements, and PLEX commands B-3
COPY/NOCOPY execution option B-70
DATE item type B-8
DECK/NODECK execution option B-71
designation bit string B-49
designation vector B-52
directives, for PLEX processor B-57
DOUBLE item type B-18
ECHO/NOECHO execution option B-71
ERRORS/NOERRORS execution option B-72
ERRSYM execution option B-72
execution options for PLEX processor B-70
EXIT execution option B-73
FINDORG program, sample B-83
FIXED BINARY attribute
    and DECIMAL and MONEY item types B-11
    and INTEGER item type B-10
FIXED DECIMAL attribute
    and DECIMAL and MONEY item types B-11
FLOAT BINARY attribute
    and DECIMAL and MONEY item types B-12
    and INTEGER item type B-10
FLOAT DECIMAL attribute
    and DECIMAL and MONEY item types B-12
    and INTEGER item type B-10
FOR command B-55
FORM/NOFORM execution option B-77
GEN/NOGEN execution option B-77
inclusion list B-45
    in LINK command B-55
INTEGER item type B-9
    and nulls B-14
ITEM execution option B-78
item list B-47
JCL B-90
    variations B-93
Keepfile, allocating B-94
LINECNT execution option B-78
LINK command and inclusion list B-55
LOAD/NOLOAD execution option B-78
MEMORY execution option B-79
PICTURE attribute
    and DECIMAL and MONEY item types B-12
    and INTEGER item type B-10
PLEX commands, format of B-3

PLEX processor
directives B-57
execution options B-70
PLEX program
example B-82
structure B-82
REAL item type B-17
reserved words in PLEX B-6
Rollback Log, allocating B-94
Savefile, allocating B-94
SORMGIN execution option B-79
SOURCE/NOSOURCE execution option B-79
SSR execution option B-80
START S2K command B-43
subschema record B-31
considerations B-40
examples B-34
format B-32
SYSSEP execution option B-80
S2KDUM control block B-59
TERM/NOTERM execution option B-80
TEXT item type B-7
TRANS execution option B-81
UNDEFINED item type B-18
Update Log, allocating B-94
PLEX commands 3
and comments and statements 3-3, B-3, C-3, D-2
format, general 3-1, B-3, C-3, D-2
opcodes for 15-1
processing in execution modes 61
return codes from, and FOR command 12-5, B-55
PLEX execution-time exits 22-7
PLEX keywords and reserved words 3-3, B-6, C-6
PLEX messages 22-13
PLEX processing 88
PLEX processor 3
diagnostic messages 19-14
directives 19-2, B-57, C-34, D-25
execution options 19-10, B-70, C-39, D-32
memory used by 22-16
messages 22-14
formats 22-14
severity codes 22-15
PLEX program
examples 20-1, B-82, C-51, C-52, D-32
job steps C-63
structure 20-2, C-51
termination 88
PLEX where-clause 7-2, *See also* where-clause and DYNAMIC where-clause
PLEX, capabilities and scope of 2, 4
PLXCOMM control block 19-6
populating a database 62, 21-1

precedence of where-clause operators 7-4, 7-30
PRESENT
and GET command 8-115
and GETD command 8-70
and GET1 command 8-34
PREVIOUS in GET command 8-113
primary stack entry, defined 26
PRINT command for testing sample loading 21-9
PROGRAM directive C-38
program structure 88
program-database communication 6
program, PLEX
examples 20-1, B-82, C-52, D-32
structure 3, 20-2, B-82, C-51
terminating 88, 17-1
Programming Facility messages 22-13
pseudo-conversational CICS transactions 22-9

**Q**

QUEST where-clause *See* DYNAMIC where-clause
QUEUE command 9-2
considerations 9-3
example 9-3
format 9-2
queue mode 62, 9-2
INSERT command in 10-52
table space for updates in 62, 9-2
QUOTE execution option 19-18

**R**

R-authority 77
READ in FRAME command 11-31, 11-37
REAL item type 4-7, 4-11, B-12, B-17, C-7, D-3
in where-clause 4-7, C-10
recovery 67, *See also* Coordinated Recovery
during data loading 21-8
with Update Log 69
REDEFINES clause
in COMMBLOCK 5-2, 5-19, 5-20
example 5-17, 5-21
in subschema record 5-23, 5-31, 5-33
example 5-29
RELEASE command 11-25
considerations 11-26
example of usage 11-25
exclusive use of database required 11-26
format 11-25
REMOVE command 54, 10-2
and local hold 10-6
considerations 10-5
examples 10-7
of syntax 10-3
of usage 10-4
for key update 10-6

for non-key update 10-6
format 10-3
REMOVE TREE command 57, 58, 59, 10-21
and GET command 8-106
and GETA command 8-50
and GETD command 8-58
and GET1 command 8-10
and GET1 PRESENT command 8-34
and Locate File addresses 10-24
and reusable table space 10-24
considerations 10-23
effects on immediate mode insert sequence 10-35
examples
of syntax 10-22
of usage 10-22
format 10-21
RENAMES clause
in COMMBLOCK 5-2, 5-19
example 5-17
in subschema record 5-23, 5-33
example 5-29
replacing records 19-13
reserved words in PLEX 3-3, C-6
RESET ROLLBACK command 11-48
considerations 11-53
examples 11-51
format 11-48
resetting the Rollback Log 6-13, *See also* RESET ROLLBACK command
RESTORE command 67, 11-8
considerations 11-10
examples
of syntax 11-9
of usage 11-10
format 11-8
stacks and Locate Files cleared by 11-10
restoring a database 67, 11-8, 21-8
retrieval 24, 8-3
commands for 26, 42
in logical order 25
linked 49
return codes 7, 77, 22-17
in COMMBLOCK 5-3, B-21, C-13
in S2KDUM control block (S2KRTC) 5-3, C-13
overview of 22-17
processing with FOR command 12-5, B-55
with FOR and FRAME commands 11-34
reusable space in database tables, use of 8-96, 9-2, 10-24, 10-36, 10-56
ROLLBACK command 11-45
considerations 11-47
example 11-46
format 11-45
Rollback Log 72, *See also* Coordinated Recovery
allocating B-94
JCL for 20-29, C-67

**S**

SAME LOCATE in where-clause 7-27
considerations 7-28
example 7-27
SAME POSITION in where-clause 7-27
considerations 7-28
example 7-27
SAME STACK in where-clause 7-27
considerations 7-28
example 7-27
sample skeleton PLEX program 20-2, B-82, C-51
SAVE command 67, 11-3
considerations 11-7
examples
of syntax 11-5
of usage 11-6
format 11-3
Savefile
allocating 11-3, B-94, C-67
and RESTORE command 11-8
and SAVE command 11-3
dynamically allocating 11-4, 11-9
with APPLY command 11-16
saving a database 67, 11-3
SCF QUEST where-clause *See* DYNAMIC where-clause
schema of a database 14, 21-2
schema record 14
scratch pads 20-25, 20-28, B-91
secondary passwords 77
secondary stack entry, defined 26
security 77
Security by Entry 78, 7-8
Self-Contained Facility messages for PLEX programs 22-18
SEQID execution option 19-18
sibling record, defined 12
similar data records, defined 14
single key condition in GETP command 8-151
single-user PLEX program, sample JCL for 20-25, B-91, C-63
SORMGIN execution option 19-19, B-79, C-48, D-32
sort key
combined 8-96
defined 8-92
sorting Locate Files *See* ORDER BY command
source control in LINK command 8-170
SOURCE execution option 19-19, B-79, C-48
source record with LINK command, defined 49
source stack with COPYS command 8-147
space, reusable and available 62
SPANS operator in where-clause 7-13
example 7-14
SR *See* schema record
SSR *See* subschema record

SSR execution option 19-20, B-80, C-49
stack 26
and INSERT sequence 57
and Locate File for same database 8-11
and SAME STACK (SAME POSITION) in where-clause 7-27
canceled
by CLOSE command 17-1, 17-3
by load mode 10-66
copying 27, 8-147
entries in 26
for multiple databases 87
levels of 26 *to* 27
multiple 27, 87
use of 26
with COPYS command 27
with GET command 43
with GETA command 33
with GETD command 29
with GETP command 8-157
with GET1 command 28
with INSERT command 57, 10-35, 10-56
START S2K command 10, 6-1, B-43, C-51, C-63
considerations 6-6
examples
of syntax 6-2
of usage 6-3, B-43
format 6-1
return code in S2KRTC 6-6
STOP S2K command 88, 17-4
considerations 17-5
example 17-5
format 17-4
structure modification 57
structure of a PLEX program 3, 20-2, B-82, C-51
subschema items
and Assembler data forms D-9
and COBOL description of 5-23
and FORTRAN variable types C-7, C-23
and PL/I attributes of B-31, B-32
and PL/I PARAMETER option B-32, B-33
as sort keys in ORDER BY command 8-95
considerations 5-32, B-41, C-31, C-32
examples 5-25, B-34, C-25, D-11
execution option for 19-17, B-78
in where-clause 4-7, C-8
maximum number of 5-34, B-43, C-33
names of, qualified in where-clause 7-9
types of 4-1, B-7, D-3
subschema record 7, 5-22, B-31, C-22, D-9
as sort key in ORDER BY command 8-95
considerations 5-31, 5-32, B-40, C-31, D-12
descriptions, relation to schema 7, 21
examples 5-25, B-34, C-25, D-11
execution option for 19-20, B-80, C-49

format 5-23, 5-31, B-32, C-23, D-9
function of 7
in LINKAGE SECTION 5-22, 5-23, 5-31
level numbers in description 5-27, 5-33, B-32, B-36
maximum number of items in 5-34, B-43, C-33
name of 5-31, B-40, C-23, C-31, D-12
opening 4-4, 5-32, B-9, B-42, C-32
REDEFINES and RENAMES clauses in 5-23, 5-31, 5-33
examples 5-29
without items 5-29, 5-31, B-39, B-40, C-29, C-31
subset retrieval 36
subtree, deleting 57, *See also* REMOVE TREE command
SUSPEND command 69, 11-22
considerations 11-24
examples
of syntax 11-22
of usage 11-23
format 11-22
symbolic relations with LINK command 48
synchpoints 6-6
and DROP HOLD command 8-144
and ENABLE/DISABLE ROLLBACK 11-28
and LINK command 8-172
and load mode 9-6
and queue mode 9-4
SYSSEP execution option 19-20, B-80, C-49
system separator, in COMMBLOCK 5-7
SYSTEM 2000 Error Code and COMP items 4-4, 4-6
SYSTEM 2000 facilities 1
SYSTEM 2000 module (overlay) structure 15-2
S2K *See* SYSTEM 2000
S2KCOUNT
and GET command 8-117
and GETD command 8-73
and GET1 command 8-39
S2KDUM control block 7, 49, 5-3, 19-2, 19-3, 19-9, B-21, B-56 *to* B-57, B-59, B-62, C-13, D-30
items, description of 19-4, B-59, C-35, D-30
S2KLKH link history array (in S2KDUM) 8-169, 12-9
S2KRTC return code (in S2KDUM) 5-3, 6-6, C-13
with FRAME and END FRAME commands 11-33
S2KSRT for no-sort capability A-1

**T**

table space
and REMOVE TREE command 10-24
use of 62, 9-2
TALLY command for testing sample loading 21-9
target control in LINK command 8-170
target Locate File with COPYL command 8-147
target record with LINK command, defined 49
target stack with COPYS command 8-147
teleprocessing environment and START S2K command 6-2, 6-6

TERM execution option 19-20, B-80, C-49
TERMINATE command 9-2, 9-5, 9-6
  and synchpoints 9-6
  considerations 9-3, 9-5
  example 9-3
  format 9-2, 9-5
  in load mode 62
  in queue mode 62
termination
  abnormal 22-18
  of processing in PLEX 88, 17-1
ternary operators in where-clause 7-13
TEXT item type 4-2, 4-10, B-7, B-16, C-7, D-3
  and trailing blanks 4-9, B-14, C-10
  in where-clause 4-8, B-13, C-8
timing output 15-1
  example 15-6
trailing blanks and TEXT item type 4-9, C-10
TRANS execution option 19-21, B-81, C-50

**U**
U-authority 77
unary operators in where-clause (EXISTS and FAILS) 7-10
UNDEFINED item type 4-7, 4-12, B-13, B-18, C-7, D-3
  in where-clause B-13, C-8
Universal Software Interface (user exits) 4, 16-1
update cycles *See also* cycle number
  defined 11-16
  with APPLY command 11-16
  with FRAME and END FRAME commands 11-30
  with KEEP command 11-12, 11-15
UPDATE in FRAME command 11-31
Update Log 67
  allocating B-94
  and RESTORE command 11-9
  and SAVE command 11-4, 11-7
  and SUSPEND command 11-22
  JCL for 20-29, C-67
  use of 69
  with APPLY command 11-16, 11-17
  with KEEP command 11-12
updated pages, clearing 64
updates 52, 10-2
  preparation for 52
  use of database tables for 62
user exits (Universal Software Interface) 4, 16-1
user synchpoints *See* synchpoints, *See also* Coordinated Recovery
USING 0 in retrieval commands 24, 52, 8-131
  examples 8-131
  for Assembler PLEX D-24

## V

value modification 54, *See also* MODIFY command, REMOVE command
VALUES PADDING 13-1
variable types (FORTRAN) and item types C-8
via-clause in LINK command 49

## W

W-authority 77, 7-8
WAS NOT USED message from PLEX processor 22-16
where-clause 36, 7-2
    and full passes of Data Table 7-34
    and no-sort capability A-1
    AND operator 7-15
    and Security by Entry 7-8
    AT operator 7-23
    binary operators 7-11
    conditions
        evaluation of 7-3
        placement of 7-33
    considerations
        for DYNAMIC 7-38
        for fixed 7-8
    construction of 7-30
    DYNAMIC 38, 7-35
        examples 7-36
    evaluation, order of conditions 7-3
    examples
        of DYNAMIC 7-36
        of fixed 7-7
    fixed 36, 7-5
    format
        of DYNAMIC 7-35
        of fixed 7-5
    HAS operator 38, 7-19
    in GETP command 8-151
        example 8-157
    in GET1 command 28, 8-44, A-1
    in LOCATE command 8-81, A-1
    item types
        and COBOL data descriptions of items in 4-7
        and FORTRAN variables C-8
        and PL/I attributes B-13
    item-to-item comparisons 7-13
    key and non-key conditions 7-32, 7-34
    normalization 7-15, 7-16
    NOT operator 7-18
    operands and operators, maximum number 7-8
    operator precedence 7-4
    OR operator 7-15
    processing of logical operators 7-32
    qualification of subschema item names, in fixed 7-9
    QUEST 38, 7-35
    SAME STACK and SAME LOCATE operators 7-27
    SCF QUEST 38, 7-35

simplifying 22-6
ternary operators 7-13
unary operators (EXISTS and FAILS) 7-10
use of 22-2
W-authority for password 7-8
with NO FULL PASSES command 7-34
WORKING-STORAGE variables in 4-8
word table 8-128
examples 8-129
WORKING-STORAGE variables in where-clause 4-8
WOULD BE FREE message from PLEX processor 22-16

**Y**

YEARCUTOFF
considerations 18-2
examples of syntax 18-2
format 18-1

# Your Turn

If you have comments or suggestions about *SYSTEM 2000® PLEX Manual, Version 12, First Edition*, please send them to us on a photocopy of this page or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Institute Inc.
Publications Division
SAS Campus Drive
Cary, NC 27513
**email:** yourturn@unx.sas.com

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
**email:** suggest@unx.sas.com

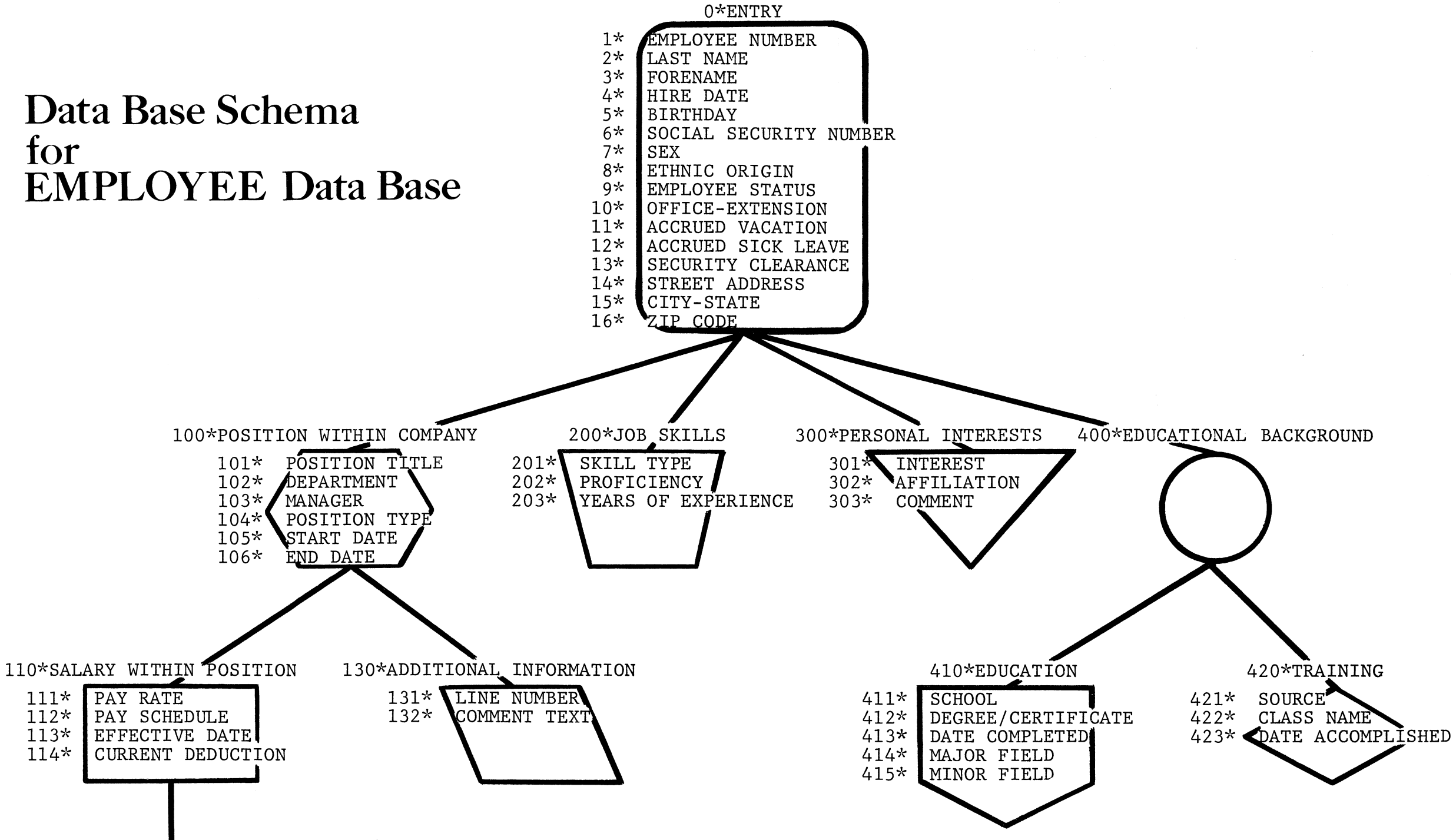
# Actual DESCRIBE Output from EMPLOYEE Data Base

```
---
DESCRIBE:
SYSTEM RELEASE NUMBER  XXX
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER            2
DATA BASE CYCLE NUMBER      24
     1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999)
     2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
     3*  FORENAME (NON-KEY CHAR X(20))
     4*  HIRE DATE (DATE)
     5*  BIRTHDAY (DATE)
     6*  SOCIAL SECURITY NUMBER (NON-KEY CHAR X(11))
     7*  SEX (CHAR X(6) WITH MANY FUTURE OCCURRENCES )
     8*  ETHNIC ORIGIN (CHAR X(9) WITH SOME FUTURE OCCURRENCES )
     9*  EMPLOYEE STATUS (CHAR X(9) WITH MANY FUTURE OCCURRENCES )
    10*  OFFICE-EXTENSION (NON-KEY CHAR X(9))
    11*  ACCRUED VACATION (NON-KEY DECIMAL NUMBER 999.99)
    12*  ACCRUED SICK LEAVE (NON-KEY DECIMAL NUMBER 999.99)
    13*  SECURITY CLEARANCE (INTEGER NUMBER 999 WITH MANY FUTURE OCCURR
         ENCES )
    14*  STREET ADDRESS (NON-KEY CHAR X(20))
    15*  CITY-STATE (NON-KEY CHAR X(15))
    16*  ZIP CODE (CHAR X(5) WITH  FEW FUTURE OCCURRENCES )
   100*  POSITION WITHIN COMPANY (RECORD)
     101*  POSITION TITLE (NON-KEY CHAR X(10) IN 100)
     102*  DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES )

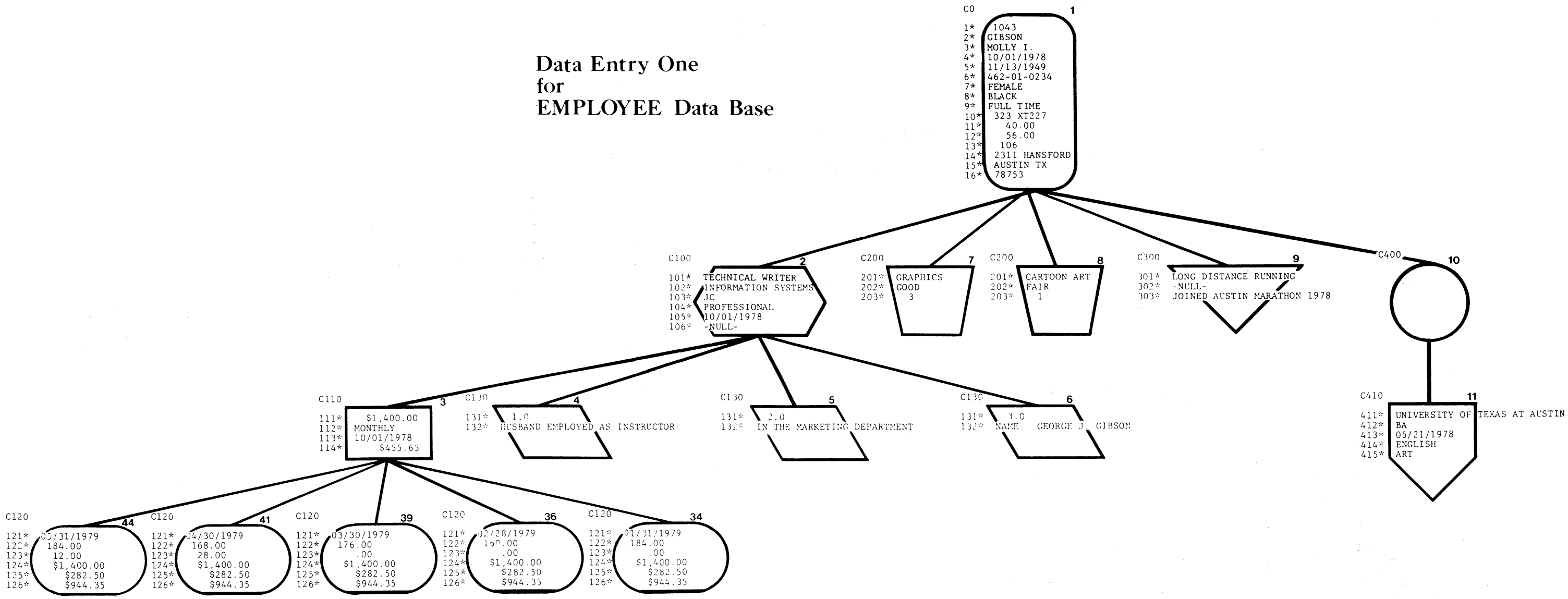
     103*  MANAGER (CHAR XXX IN 100 WITH  FEW FUTURE OCCURRENCES )
     104*  POSITION TYPE (CHAR X(12) IN 100 WITH SOME FUTURE OCCURRENCE
           S )
     105*  START DATE (DATE IN 100)
     106*  END DATE (NON-KEY DATE IN 100)
     110*  SALARY WITHIN POSITION (RECORD IN 100)
       111*  PAY RATE (MONEY $9999.99 IN 110)
       112*  PAY SCHEDULE (CHAR X(7) IN 110)
       113*  EFFECTIVE DATE (DATE IN 110)
       114*  CURRENT DEDUCTION (NON-KEY MONEY $9999.99 IN 110)
       120*  MONTHLY PAYROLL ACCOUNTING (RECORD IN 110)
         121*  PAYROLL MONTH (DATE IN 120)
         122*  REGULAR HOURS (NON-KEY DECIMAL NUMBER 999.99 IN 120)
         123*  OVERTIME HOURS (NON-KEY DECIMAL NUMBER 999.99 IN 120)
         124*  GROSS PAY (NON-KEY MONEY $9999.99 IN 120)
         125*  FEDERAL TAX DEDUCTION (NON-KEY MONEY $9999.99 IN 120)
         126*  NET PAY (NON-KEY MONEY $9999.99 IN 120)
     130*  ADDITIONAL INFORMATION (RECORD IN 100)
       131*  LINE NUMBER (DECIMAL NUMBER 99.9 IN 130)
       132*  COMMENT TEXT (NON-KEY TEXT X(7) IN 130)
   200*  JOB SKILLS (RECORD)
     201*  SKILL TYPE (CHAR X(12) IN 200 WITH SOME FUTURE OCCURRENCES )

     202*  PROFICIENCY (NON-KEY CHAR X(5) IN 200)
     203*  YEARS OF EXPERIENCE (NON-KEY INTEGER NUMBER 99 IN 200)
   300*  PERSONAL INTERESTS (RECORD)
     301*  INTEREST (CHAR X(12) IN 300 WITH  FEW FUTURE OCCURRENCES )
     302*  AFFILIATION (NON-KEY CHAR X(5) IN 300)
     303*  COMMENT (NON-KEY TEXT X(5) IN 300)
   400*  EDUCATIONAL BACKGROUND (RECORD)
     410*  EDUCATION (RECORD IN 400)
       411*  SCHOOL (CHAR X(15) IN 410)
       412*  DEGREE/CERTIFICATE (CHAR X(7) IN 410 WITH  FEW FUTURE OCCU
             RRENCES )
       413*  DATE COMPLETED (DATE IN 410)
       414*  MAJOR FIELD (NON-KEY CHAR X(16) IN 410)
       415*  MINOR FIELD (NON-KEY CHAR X(12) IN 410)
     420*  TRAINING (RECORD IN 400)
       421*  SOURCE (NON-KEY CHAR X(12) IN 420)
       422*  CLASS NAME (CHAR X(12) IN 420 WITH  FEW FUTURE OCCURRENCES
             )
       423*  DATE ACCOMPLISHED (DATE IN 420)
---
```

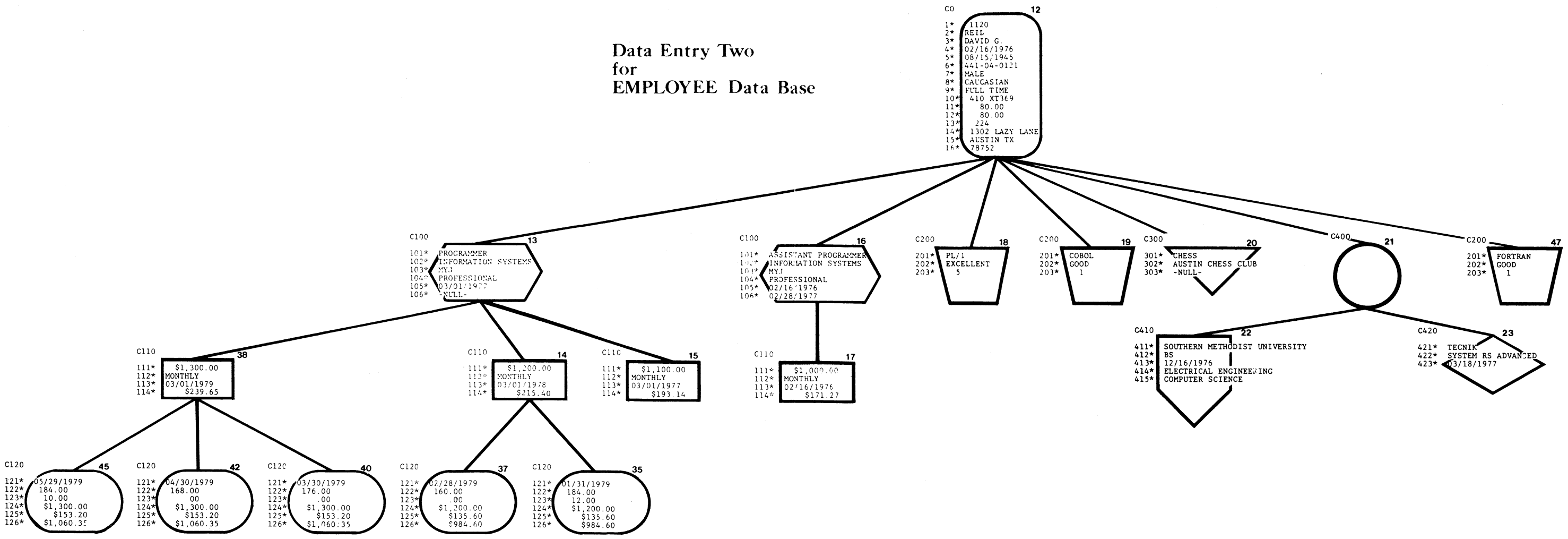
# Data Base Schema for EMPLOYEE Data Base



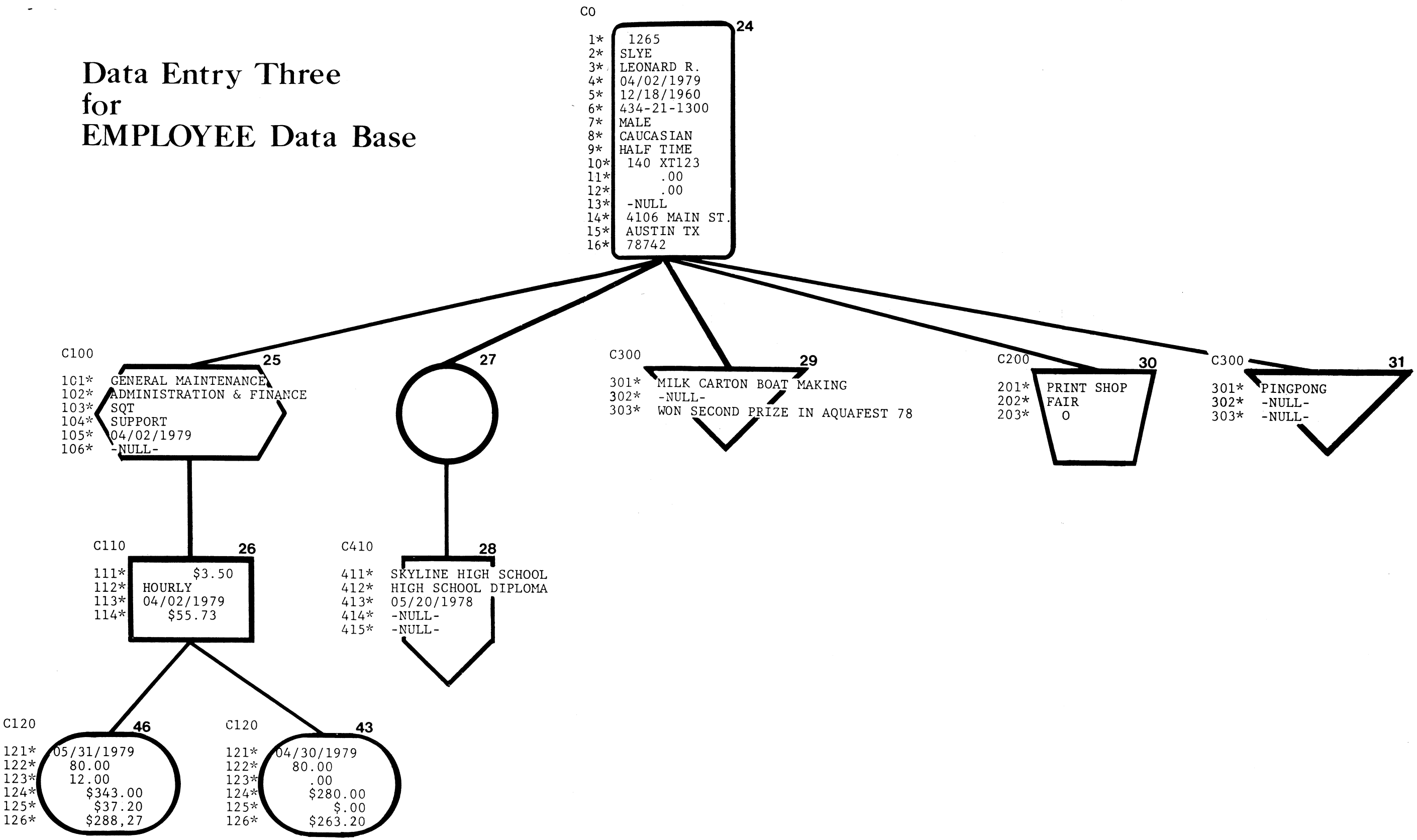
# Data Entry One
# for
# EMPLOYEE Data Base



# Data Entry Two
# for
# EMPLOYEE Data Base



# Data Entry Three
# for
# EMPLOYEE Data Base



# Data Entry Four
# for
# EMPLOYEE Data Base

