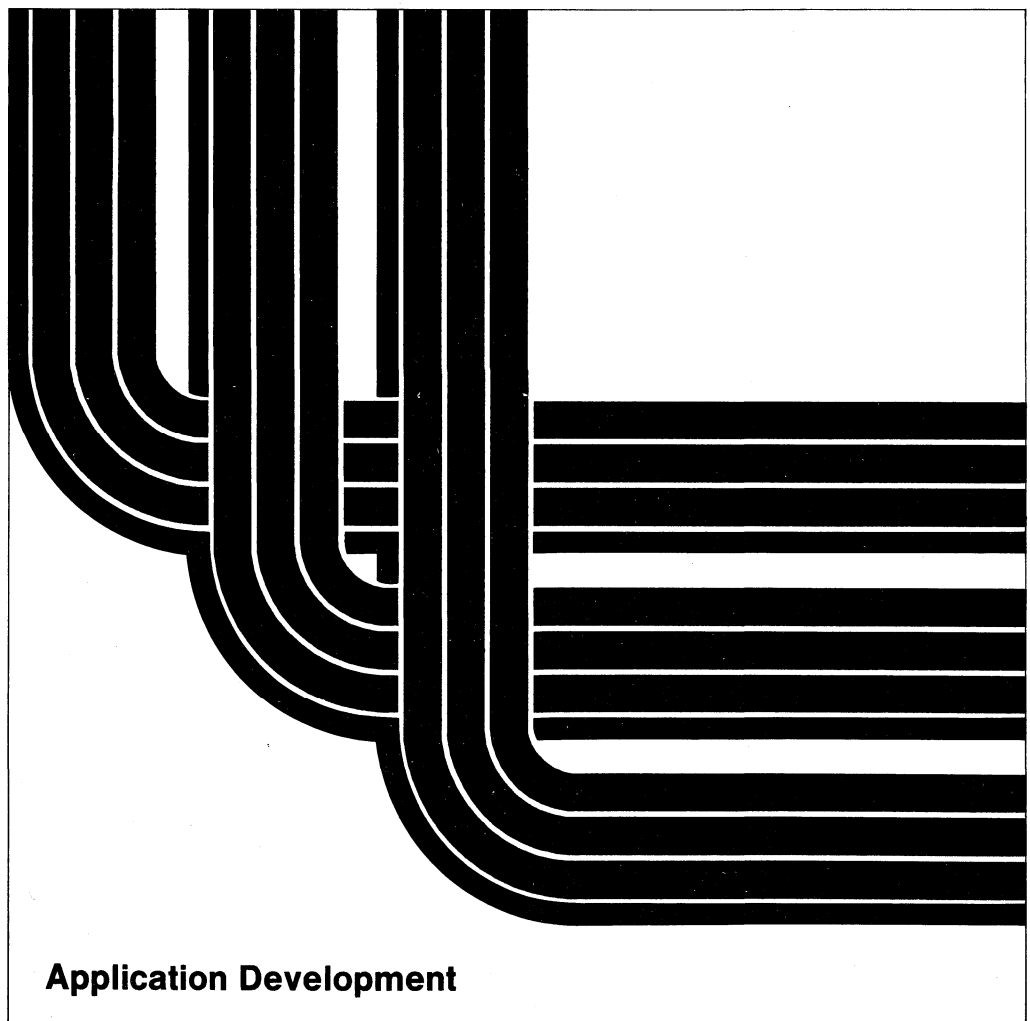


Application System/400™

SC09-1383-01

Languages:
Systems Application Architecture™ AD/Cycle™
COBOL/400™ User's Guide

Version 2





Application System/400™

SC09-1383-01

Languages:

**Systems Application Architecture™ AD/Cycle™
COBOL/400™ User's Guide**

Version 2

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page ix.

Second Edition (September 1992)

This edition applies to the IBM Systems Application Architecture* AD/Cycle* COBOL/400* licensed program (Program 5738-CB1), Version 2 Release 2, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the proper edition for the level of the product.

Order publications through your IBM representative or the IBM branch serving your locality. Publications are not stocked at the address given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, you may address your comments to:

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, Ontario, Canada M3C 1V7

You can also send your comments by facsimile to (416) 448-6057, addressed to the attention of the RCF Coordinator. If you have access to Internet, you can send your comments electronically to torrcf@vnet.ibm.com; IBMlink*, to [toribm\(torrcf\)](mailto:toribm(torrcf)@vnet.ibm.com); IBM PROFS*, to [torolab4\(torrcf\)](mailto:torolab4(torrcf)@vnet.ibm.com).

If you choose to respond through Internet, please include either your entire Internet network address, or a postal address.

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you or restricting your use of it.

© Copyright International Business Machines Corporation 1988, 1992. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

IBM is a registered trademark of International Business Machines Corporation, Armonk, N.Y.

Contents

Notices	ix
Programming Interface Information	x
Trademarks and Service Marks	x
About This Manual	xi
Who Should Use This Manual	xi
Industry Standards Used in Compiler Design	xiii
An Acknowledgment	xiii
Summary of Changes	xv
Version 2 Release 1 Modification 1 Changes	xv
Version 2 Release 2 Changes	xvi
Chapter 1. An Introduction to the COBOL/400 Programming Language	1
Extensions to the ANSI Standard	1
Features of the COBOL/400 Compiler	2
Other Application Development Tools	2
AD/Cycle CoOperative Development Environment/400*	2
Using COBOL/400 Syntax Notation	3
Reading the Syntax Diagrams	3
Reading IBM Extensions	5
CL Entry Codes	5
An Overview of COBOL/400 Programming	6
Chapter 2. Entering Your Source Program on the AS/400 System	9
Designing Your COBOL/400 Program	9
Entering Source Using SEU	11
Chapter 3. Compiling a COBOL/400 Program	15
Using the Create COBOL Program (CRTCLPGM) Command	15
Using the CRTCLPGM Prompt Displays	16
Entering CRTCLPGM from the Command Line	28
Entering CRTCLPGM from a CL Program	28
Syntax of the CRTCLPGM Command	29
Compiling Your Source Program For the Previous Release	30
Using the PROCESS Statement to Specify Compiler Options	31
Understanding Compiler Output	36
Specifying the Format of Your Listing	36
Browsing Your Compiler Listing Using SEU	37
A Sample Program and Listing	39
Chapter 4. Running Your COBOL Program	49
Replying to Run-Time Inquiry Messages	50
Chapter 5. Debugging Your Program	53
Avoiding Common Coding Errors	54
Using Breakpoints	55
Example of a Program Using Breakpoints	55
Changing Program Variables	61
Considerations for Using Breakpoints	61
Using a Trace	62

Example of Using a Trace	62
Considerations for Using a Trace	64
Using a Debug Run-Time Switch	65
Using a COBOL Formatted Dump	65
Chapter 6. COBOL/400 Exception and Error Handling	67
Standard Error Handling	67
Error Handling Overview	67
Using Error-Handling Application Programming Interfaces (APIs)	68
Internal and External File Status	68
General Error Detection	70
How File Status is Set	70
Message Monitor Generation	71
Ending of a COBOL Program	72
Return Codes	73
Standard and Nonstandard Error Handling Models	74
Effects of *STDERR and *NOSTDERR on File Status	77
Processing of I/O Verbs	78
Common Exceptions and Some of Their Causes	79
Recovery After a Failure	80
Chapter 7. File and Data Management	87
Device Independence and Device Dependence	87
Spooling	89
Output Spool	89
Input Spool	90
System Override Considerations	90
File and Record Locking by COBOL	91
Locking and Releasing Records	91
Sharing an Open Data Path	92
Commitment Control Considerations	92
Unblocking Input Records and Blocking Output Records	100
File Status and Feedback Areas	101
File Descriptions	102
Program-Described Files	102
Externally Described Files	103
Data Description Specifications (DDS)	104
Format 2 COPY Statement (DD, DDR, DDS, or DDSR Option)	110
Indicators	113
Data Structures Generated	113
Key Generation Examples	119
Declaring Data Items Using CVTOPT Data Types	127
DBCS-Graphic Fields	129
Variable-length DBCS-Graphic Fields	130
Examples:	131
Cross-system Data Considerations	133
Chapter 8. Transaction Files	135
Program-Described Transaction Files	135
Externally Described Transaction Files	135
The Format 2 COPY Statement	135
Data Description Specifications	136
Processing an Externally Described Transaction File	138
Using Indicators with Transaction Files	138
Indicators in a Separate Indicator Area	138

Indicators in the Record Area	139
ASSIGN Clause and the Separate Indicator Area Attribute	139
Data Description Entry — Boolean Data	140
INDICATORS Phrase	141
Indicators in a Separate Indicator Area	142
Indicators in the Record Area	142
Indicators Example Programs	143
Subfiles	153
Use of Subfiles	155
Multiple Device Files and Single Device Files	158
Environment Division	167
File-Control Entry	167
Data Division	171
File Description Entry	171
Boolean Data Items	172
Procedure Division	172
Procedure Division Concepts	172
ACCEPT Statement	173
ACQUIRE Statement	174
CLOSE Statement	175
DROP Statement	175
OPEN Statement	176
Common Processing Facilities	177
READ Statement	179
REWRITE Statement	187
WRITE Statement	189
USE Statement	195
Examples of Work Station Programs	196
Basic Inquiry Program	196
Order Inquiry Programs Using Subfiles	202
A Payment Update Program	213
Chapter 9. Printer Files	229
SPECIAL-NAMES Paragraph and the ADVANCING Phrase	229
LINAGE Clause	229
FORMATFILE Files	230
Chapter 10. DISK and DATABASE Files	237
DATABASE versus DISK Files	237
Processing Methods for DISK and DATABASE Files	237
COBOL Indexed Files	237
Referring to a Partial Key	238
Logical File Considerations	242
COBOL Relative Files	244
COBOL Sequential Files	245
COBOL File Organization and AS/400 File Access Path Considerations	246
File Processing Methods	246
Descending File Considerations	249
Chapter 11. COBOL/400 Programming Considerations	251
Issuing a CL Command from a COBOL Program	251
Using the CORRESPONDING Phrase	252
LIKE Clause	254
Reference Modification	258
Reference Modification with Variable-length Tables	260

Reference Modification Using Data Names	260
Reference Modification with Subscripting	261
De-editing	262
De-editing Examples	262
Handling Data Errors	263
Performance Considerations	264
PICTURE Clauses for Numeric Items	264
Eight-Byte Binary Items	264
Segmentation	265
Debugging	265
*NORANGE Option	265
*DUPKEYCHK Option	265
Relative Files	265
Indicators	265
Commitment Control	266
Reading without Record Locks	266
Initializing Variables	266
Blocking Records	266
Program Loops	266
Tracing a Loop in a Program	266
Errors That Can Cause a Loop	267
Chapter 12. Communicating Between Programs	269
Transferring Control to Another Program	269
Main Programs and Subprograms	269
Returning Control from a Called Program	270
Initialization of Storage	275
Calling Another Program	275
Passing Data Using BY REFERENCE or BY CONTENT	275
In the Linkage Section	277
Grouping Data to be Passed	277
Call by Identifier	278
Using Pointers in a COBOL/400 Program	278
Defining Pointers and Pointer Alignment	279
Pointers and the REDEFINES Clause	280
Reading and Writing Pointers	281
Initializing Pointers Using the NULL Figurative Constant	282
LENGTH OF Special Register	282
Setting the Address of Linkage Items	283
Using ADDRESS OF and the ADDRESS OF Special Register	283
Using Pointers in a MOVE Statement	283
Using Pointers in a CALL Statement	285
Using Pointers and APIs to Access User Spaces	287
Processing a Chained List	298
Data Areas	301
Local Data Area	302
Program Initialization Parameters (PIP) Data Area	302
File Considerations	302
Appendix A. Segmentation Feature	305
Segmentation Concepts	305
Program Segments	305
Segmentation Logic	306
Segmentation Control	306
COBOL Source Program Considerations	306

Segmentation – Special Considerations	308
Appendix B. Debugging Features	309
COBOL Source Language Debugging	309
Compile-Time Switch	309
Run-Time Switch	310
USE FOR DEBUGGING Declarative	312
DEBUG-ITEM Special Register	315
Debugging Lines	317
Appendix C. Level of Language Support	319
ANSI X3.23-1985 COBOL Standard	319
COBOL/400 Level of Language Support	319
SAA Common Programming Interface (CPI) Support	321
Appendix D. COBOL/400 Messages, the FIPS Flagger, and SAA Flagging	323
COBOL/400 Messages	323
Interactive Messages	323
Compilation Messages	325
Responding to Messages	325
COBOL Message Descriptions	326
The Federal Information Processing Standard (FIPS) Flagger	327
SAA Flagging	329
Appendix E. Differences Between ANSI 74 COBOL and ANSI 85 COBOL	331
Migrating ANSI 74 COBOL Programs to ANSI 85 COBOL	331
Appendix F. Supporting International Languages with Double-Byte Character Sets	333
Using DBCS Characters in Literals	333
Identification Division	336
Environment Division	336
Data Division	337
Procedure Division	338
SORT/MERGE	344
Compiler-Directing Statements	344
Communications between Programs	345
FIPS Flagger	345
COBOL Program Listings	345
Appendix G. AS/400 File Processing Examples	347
Sequential File Creation	347
Sequential File Updating and Extension	349
Indexed File Creation	352
Indexed File Updating	353
Relative File Creation	357
Relative File Updating	359
Relative File Retrieval	361
Sorting and Merging Files	364
Appendix H. Example of a COBOL Formatted Dump	367
Appendix I. Glossary of Abbreviations	379
Bibliography	381

Index 383

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

This publication could contain technical inaccuracies or typographical errors.

This publication may refer to products that are announced but not currently available in your country. This publication may also refer to products that have not been announced in your country. IBM makes no commitment to make available any unannounced products referred to herein. The final decision to announce any product is based on IBM's business and technical judgment.

Important changes or additions to the text are indicated by a vertical line (|) to the left of the change or addition. Refer to the "Summary of Changes" on page xv for a summary of the changes to the COBOL/400 licensed program and how they are described in this publication.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This publication contains small programs that are furnished by IBM as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. All programs contained herein are provided to you "AS IS". THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

Programming Interface Information

This publication is intended to help the customer write COBOL/400 programs.

This publication also documents General-Use Programming Interface and Associated Guidance Information.

General-Use programming interfaces allow the customer to write programs that obtain the services of the COBOL/400 compiler.

General-Use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

```

|_____ General-Use Programming Interface _____|
|
| General-Use Programming Interface and Associated Guidance Information...
|
|_____ End of General-Use Programming Interface _____|
```

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

AD/Cycle	Application System/400	AS/400
CICS/400	COBOL/400	IBM
IBMLink	Operating System/2	Operating System/400
OS/2	OS/400	PROFS
RPG/400	SAA	SQL/400
System/370	Systems Application Architecture	400

The following term, denoted by a double asterisk (**) in this publication, is a trademark of another company as follows:

FLOW-MATIC Sperry Rand Corporation

About This Manual

This manual provides information an application programmer needs to write, compile, test, debug, and run COBOL/400* programs on the IBM* Application System/400* (AS/400*) system.

This manual refers to other IBM publications. These publications are listed in the "Bibliography" on page 381 with their full title and base order number. When they are referred to in text, a shortened version of the title is used.

Who Should Use This Manual

This manual is intended for programmers who have some experience with the COBOL programming language and for the operators who run the programs. It is a guide to programming in the COBOL/400 language for users of the AS/400 system. As a user, you should have a basic understanding of data processing concepts, the COBOL programming language, and the IBM Operating System/400* (OS/400*) operating system.

Using this manual, you will be able to:

- Design COBOL/400 programs
- Code COBOL/400 programs
- Enter, compile, and run COBOL/400 programs
- Test and debug COBOL/400 programs
- Study coded COBOL/400 examples.

Note: You should be familiar with Chapters 1 through 4 of this manual before proceeding to the other chapters.

Use this manual with the *Languages: Systems Application Architecture* AD/Cycle* COBOL/400* Reference*, SC09-1380, which describes each component and feature of the COBOL/400 language. The *COBOL/400* User's Guide* and the *COBOL/400* Reference* together describe the COBOL/400 compiler and language.

For information about the complete library of AS/400 documents, consult the *Publications Guide*, GC41-9678, which contains a brief description of the contents of each AS/400 manual.

Before you use this manual, you should be familiar with the following information:

- How to use the controls and indicators on your display and how to use the keys on your keyboard, such as:
 - Cursor movement keys
 - Function keys
 - Field exit keys
 - Insert and Delete keys
 - Error Reset key.

For information about your display station, refer to:

- *New User's Guide*, SC41-8211.

- How to operate your display station when it is linked to the IBM AS/400 system and running AS/400 software. This means knowing how to use the OS/400 operating system and its Control Language (CL) to do such things as:
 - Sign on and sign off the display station
 - Interact with displays
 - Use Help
 - Enter CL commands
 - Use Application Development Tools
 - Respond to messages
 - Perform file management.
- The *Programming: Control Language Programmer's Guide*, SC41-8077, which contains the basic concepts of OS/400 CL functions.

To find out more about the operating system and its control language, refer to these IBM publications:

- *Programming: Control Language Reference*, SC41-0030 (a three-volume manual).
- *Programming: Work Management Guide*, SC41-8078.
- *Advanced Backup and Recovery Guide*, SC41-8079.
- The *Data Management Guide*, SC41-9658, which provides information on using data management support to allow an application to work with files. The manual includes information on:
 - Fundamental structure and concepts of data management support on the system
 - Data management support for display stations, printers, tapes, and diskettes, as well as spooling support
 - Overrides and file redirection (temporarily making changes to files when an application is run)
 - Copying files by using system commands to copy data from one place to another
 - Tailoring a system using double-byte data.
- How to use the following Application Development Tools:
 - The Screen Design Aid (SDA) is used to design and code displays. Information about this product is contained in *Application Development Tools: Screen Design Aid User's Guide and Reference*, SC09-1340.
 - The Source Entry Utility (SEU) is a full-display editor you can use to enter and update your source members. Information about this product is contained in *Application Development Tools: Source Entry Utility User's Guide and Reference*, SC09-1338.
- The Structured Query Language (SQL) allows you to insert SQL statements into COBOL/400 programs. Information about this product is contained in *Systems Application Architecture* Structured Query Language/400 Programmer's Guide*, SC41-9609, and in *Systems Application Architecture* Structured Query Language/400 Reference*, SC41-9608.
- The Customer Information Control System/400 (CICS/400*) licensed program allows you to enter transactions at remote work stations, and process them concurrently with user-written application programs. The licensed program includes functions for building, using, and maintaining databases, and for communicating with CICS on other operating systems.

Information about using this product for application programming is contained in the *CICS/400 Application Programming Guide*, SC33-0822.

Industry Standards Used in Compiler Design

The COBOL/400 compiler is designed according to the following industry standards as understood and interpreted by IBM, as of September, 1987:

- The intermediate subset of the American National Standards Institute (ANSI X3.23-1985) standard.
- The International Standards Organization (ISO) 1989-1985.
- The March 1986 Federal Information Processing Standards Publication (FIPS PUB 21-2) intermediate level. Additional support is provided for many high-level features.

Portions of this manual are copied from American National Standard Programming Language COBOL, ANSI X3.23-1985, ISO 1989-1985 and reproduced with permission from *American National Standard Programming Language COBOL, ANSI X3.23-1985, ISO 1989-1985* (copyright 1985 by the American National Standards Institute), copies of which you can purchase from the American National Standard Institute at 1430 Broadway, New York, New York, 10018.

The COBOL language is maintained by the Conference On DATA SYStems Languages (CODASYL).

An Acknowledgment

The following extract from U.S. Government Printing Office Form Number 1965-0795689 is presented for your information and guidance:

Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention COBOL in acknowledgment of the source, but need not quote this entire section.

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

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

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference of Data Systems Languages.

The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC[™] Programming for the UNIVAC (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

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

Summary of Changes

Version 2 Release 1 Modification 1 Changes

The following is a summary of changes that occurred in Version 2 Release 1 Modification 1 of the COBOL/400 compiler. This information is also documented in the *Version 2 Release 1.1 Changes* manual.

TGTRLS (Target Release) Parameter: The TGTRLS (target release) parameter of the CRTCLPGM command now has a new *release-level* value that allows you to specify the operating system release level running on the target system on which a program is to be used.

Improvements to Extended ACCEPT and Extended DISPLAY Operations: The EXTDSOPT (Extended ACCEPT and DISPLAY Options) parameter is now available in the CRTCLPGM command and the PROCESS statement.

This parameter provides two options: *NOUNDSPCHR that allows you to specify the type of data that extended ACCEPT and extended DISPLAY statements handle in a particular program, and *DFRWRT that allows you to specify if you want to buffer the processing of extended DISPLAY statements.

You can use the EXTDSOPT parameter only when you specify the *EXTACCDSP generation option, or the EXTACCDSP PROCESS statement option.

Although the deferred writing (*DFRWRT) option improves performance by buffering data streams generated by consecutive extended DISPLAY statements, the no deferred writing (*NODFRWRT) option allows you to associate data errors with the statements that cause them by performing extended DISPLAY statements as they are encountered.

The no undisplayable characters (*NOUNDSPCHR) option allows you to use extended ACCEPT and extended DISPLAY statements on local workstations, or on display stations attached to remote 3174 and 3274 controllers, provided that the data does not contain undisplayable characters.

In previous releases and modifications, extended ACCEPT and extended DISPLAY statements were processed as if the default EXTDSOPT options (*DFRWRT and *UNDSPCHR) were in effect.

SAA Data Types: The COBOL/400 compiler allows you to convert three SAA data types from externally described files into standard COBOL data items. The SAA data types that you can convert are date, time, and timestamp. These conversions are available through the *DATETIME option of the CRTCLPGM CVTOPT parameter, or the DATETIME option of the PROCESS statement.

Variable-length Fields: You can bring a variable-length field into your program if you specify the *VARCHAR option of the CRTCLPGM CVTOPT parameter, or the VARCHAR option of the PROCESS statement.

Null-capable Fields: Although your program can process null-capable fields, null values are not supported. When a file with one or more null-capable fields is opened successfully, file status OP is returned. If you try to perform READ, SORT, or MERGE operations using fields that contain null values, errors occur.

Time-separation Characters: The TIMSEP parameter of job-related commands (such as CHGJOB) now specifies the time-separation character used in the WHEN-COMPILED special register, and in the time stamps that appear on compiler listings.

LIKE Clause: When you use the BLANK WHEN ZERO attribute, the PICTURE IS portion of the generated comment is now shown in a concise format, even if the parent item is numeric-edited.

Version 2 Release 2 Changes

The following is a summary of changes that occurred in Version 2 Release 2 of the COBOL/400 compiler.

Saving Data While Programs are Active: Application programs that change objects or data may run while the objects or data are being saved. Refer to the *Advanced Backup and Recovery Guide* for possible programming considerations related to Save While Active support.

LENGTH OF Special Register: You can make your programs easier to maintain with the new LENGTH OF special register, by using LENGTH OF instead of a specific length for data items. The LENGTH OF special register returns the length of a data item in number of bytes, and can be specified in the Procedure Division where a numeric literal is allowed.

Declaring Pointer Data Items: You can now declare and use pointer data items by specifying a usage of POINTER. Pointer data items can be set, compared for equality, and passed to other programs. The addition of this support allows easier migration of COBOL code from other SAA platforms, and allows access to Application Programming Interfaces (APIs) and languages that require pointers.

ADDRESS OF Special Register and NULL Figurative Constant: Pointer data items allow the use of the ADDRESS OF special register and the NULL figurative constant. The ADDRESS OF special register returns a pointer containing the address of an item in the linkage section of a program. The NULL figurative constant returns a pointer whose address is not valid. Like pointers, the ADDRESS OF special register can be set, compared for equality, and passed to other programs.

Addition to CALL USING Statement: The CALL USING statement can now contain the ANSI 85 high-level phrases BY CONTENT and BY REFERENCE. The BY REFERENCE phrase is the default for the current CALL statement. The BY CONTENT phrase is similar to the BY REFERENCE phrase, except that copies of the USING data items are passed so that the original values in the calling program cannot be changed by the called program.

Error-Handling and Multiple Run Unit APIs: Three new APIs have been defined for use with COBOL/400 programs:

QLRCHGCM - Change COBOL Main Program allows the definition of multiple run units

QLRRTVCE - allows you to retrieve the name of the current or pending error-handling routine
QLRSETCE - allows you to specify the identity of a COBOL error-handling program for a run unit.

Together, these APIs allow user-supplied error handlers to be specified for each run unit. These APIs are documented in the *System Programmer's Interface Reference*.

MOVE with DE-EDITING and CALL (NOT) ON EXCEPTION: Two new ANSI 85 high-level functions are now available: MOVE WITH DE-EDITING and CALL (NOT) ON EXCEPTION. MOVE WITH DE-EDITING adds function and usability to the MOVE statement by allowing a numeric-edited item to be moved into a numeric or a numeric-edited item. The ON EXCEPTION phrase has been added to the CALL statement so that when an error occurs on a subprogram call, the imperative statement following the ON EXCEPTION is processed. Similarly, the NOT ON EXCEPTION is processed if no errors occur when a subprogram is called.

Improving Run-Time Performance: Run-time performance can be improved by using the following enhancements:

Compiler options *NOSTDINZ or *STDINZ allow you to selectively turn off or turn on the automatic initialization of user-declared data structures to machine defaults.

Sequential blocking is now done for dynamically-accessed files that are open for input or output, and for sequentially-accessed files that are open for input and subject to a START statement. This sequential blocking can be conditionally turned off or on with the *NOBLK and *BLK compiler options.

Long Key Support: The maximum key size for data base files changes from 120 to 2 000 bytes. For more information, refer to the *DDS Reference*.

Limited Support for DBCS-Graphic Data Type: The COBOL/400 compiler now provides limited support for the new Double Byte Character Set (DBCS)-graphic data type from files. The compiler allows the DBCS-graphic field to be brought in as a character field in a program. Two new values are added to the CVTOPT parameter of the CRTCLPGM command: *GRAPHIC and *NOGRAPHIC. Also, the equivalent PROCESS statement options, CVTGRAPHIC and NOCVTGRAPHIC, are available.

Customer Information Control System (CICS) Support: COBOL/400 applications can now use the facilities of the CICS/400* product.

Extended ACCEPT/DISPLAY Statement: The SIZE clause is now supported on the extended DISPLAY statement. Two new options, *ACCUPDALL and *ACCUPDNE are now available on the EXTDSPOPT parameter of the CRTCLPGM command. These options affect the predisplaying of items in the extended ACCEPT statement.

Application Development Manager/400 Support: You can now use the Application Development Manager/400 product with the COBOL/400 compiler to help you manage multiple versions of program source and program objects in an application. It also provides an application build facility.

AD/Cycle CODE/400* Support: New options and parameters have been added to the CRTCLPGM command to support the use of the CODE/400 product to edit, compile and debug COBOL/400 programs on programmable workstations.

Usability Improvements to this Manual: The organization of this manual has changed, to improve the usability and retrievability of the information.

Information pertaining to the various file types is grouped by file type. For example, there is a chapter on transaction files, one on printer files, and another on disk and database files. In addition, common information that pertains to all file types is consolidated into Chapter 7, "File and Data Management." The chapter on programming considerations was reduced to information about programming techniques and improving performance. Large programming examples and multi-page listings were moved to the back of the manual.

The improved index makes finding information much easier.

Chapter 1. An Introduction to the COBOL/400 Programming Language

COmmon Business Oriented Language (COBOL) is a programming language that resembles English. As its name suggests, COBOL is especially efficient for processing business problems. It emphasizes describing and handling of data items and of input/output records; thus, it is well adapted for managing large files of data.

The COBOL/400 language delivers many elements of IBM Systems Application Architecture* (SAA*) Common Programming Interface (CPI) COBOL, and is the implementing product on the AS/400 system.

The COBOL/400 Compiler and Library is an IBM licensed program that accepts and runs COBOL programs that follow the ANSI X3.23-1985 (*American National Standard Programming Language COBOL, ANSI X3.23-1985, ISO 1989-1985*) standard. ANSI is an organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

Extensions to the ANSI Standard

To help you use COBOL on the AS/400 system, the COBOL/400 licensed program also includes a number of IBM extensions to the ANSI X3.23-1985 standard. Significant extensions include:

- TRANSACTION I/O: You can send or receive records from a work station.
- COPY: You can use externally described files.
- DATABASE I/O: You can use standard COBOL Environment and Data Division entries to specify file identification, field definitions, and data structures. Clauses have been added to the READ, WRITE, REWRITE, DELETE, and START verbs to support the AS/400 database.
- Extended data types: computational-3 (internal decimal or packed decimal), and computational-4 (binary) data types are supported.
- Boolean and pointer data types are supported.
- You have the option to use the apostrophe instead of a quotation mark.
- The compiler-directing statements SKIP1/2/3, EJECT, and TITLE are supported.
- Extended ACCEPT/DISPLAY: Provides support for field-level work station I/O.
- LIKE clause: You can define the characteristics of a data name by copying them from a previously-defined data name.
- Compiler listing suppression: You can selectively suppress portions of the compiler listing by using the *CBL or *CONTROL statement, or the SUPPRESS phrase of the COPY statement.
- Hexadecimal nonnumeric literals are supported.

Features of the COBOL/400 Compiler

The following language-independent features are available with the COBOL/400 compiler:

- Syntax checking:

The Source Entry Utility (SEU) provides a COBOL syntax checker that checks for errors in lines of code as you enter or change them. Error messages are displayed, allowing you to correct errors before compilation time.

- The cross-reference option:

- Provides a listing of each Data Division name and Procedure Division paragraph name
- Indicates the statement numbers of each reference to the item.

- Suppression of diagnostic messages below a user-specified level.

- The Federal Information Processing Standard (FIPS) flagger issues messages identifying obsolete or nonconforming language elements in the COBOL source program. A **source program** is a set of instructions that is written in a programming language and must be translated to machine language before the program can be run.

- SAA flagging to highlight the functions in your program that are not portable to other SAA COBOL environments.

Other Application Development Tools

The following products are now available to help you more efficiently develop COBOL applications.

The Application Development Manager/400 licensed program provides application development organizations with a mechanism for efficiently and effectively managing application objects throughout the life of the application. This product lets a group of developers create, manage, and organize multiple versions of their application through the AD/Cycle WorkStation Platform/2 product or directly from the AS/400 command line. For more information on the Application Development Manager/400 licensed program, see the *SAA* AD/Cycle* Application Development Manager/400 Concepts*, GC09-1377.

AD/Cycle CoOperative Development Environment/400*

The AD/Cycle CODE/400* product is the edit, compile, debug facility for third-generation programming language application development and maintenance within the AD/Cycle framework. CODE provides a consistent user interface across different platforms and languages, a language-sensitive editor, and an interactive debugger operating in a windowed environment on the IBM Operating System/2* (OS/2*) licensed program, in cooperation with an Application System/400 or System/370* host. For more information on AD/Cycle CODE, see the *IBM SAA AD/Cycle CoOperative Development Environment: General Information*, GC26-4661.

The COBOL/400 compiler has been enhanced to fully support the CODE/400 product.

Using COBOL/400 Syntax Notation

In COBOL, basic formats are presented in a uniform system of syntax notation which is explained in the following paragraphs. This notation is designed to assist you in writing COBOL source statements.

- COBOL keywords appear in uppercase letters; for example:

PARM1

They must be spelled exactly as shown. If any required keyword is missing, the compiler considers it an error.

- Variables representing user-supplied names or values appear in all lowercase letters; for example:

parmx

- For easier text reference, some words are followed by a hyphen and a digit or a letter; for example:

identifier-1

This suffix does not change the syntactical definition of the word.

- Arithmetic and logical operators (+, -, *, /, **, >, <, =, >=, and <=) that appear in syntax formats are required. These operators are *special character* reserved words. For a complete listing of reserved COBOL/400 words, see the “Reserved Words” section of the *COBOL/400 Reference*.
- All punctuation and other special characters appearing in the diagram are required by the syntax of the format when they are shown; if you leave them out, an error occurs in the program.
- You must write the required clauses and the optional clauses, (when used), in the order shown in the diagram unless the associated rules explicitly state otherwise.

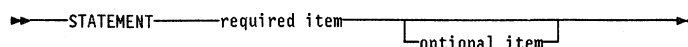
Reading the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below.

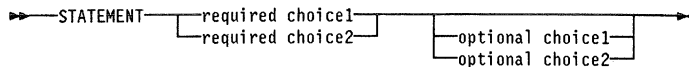
- Read the syntax diagrams from left to right, and from top to bottom, following the path of the line:
 - ▶— Indicates the beginning of a statement. Diagrams of syntactical units other than statements, such as clauses, phrases and paragraphs, also start with this symbol.
 - ▶ Indicates that the statement syntax is continued on the next line.
 - ▶— Indicates that a statement is continued from the previous line.
 - ▶ indicates the end of a statement. Diagrams of syntactical units other than statements, such as clauses, phrases and paragraphs, also end with this symbol.

Note: Statements within a diagram of an entire paragraph do not start with ▶— and end with —▶ unless their beginning or ending coincides with that of the paragraph.

- Required items appear on the horizontal line (the main path). Optional items appear below the main path:



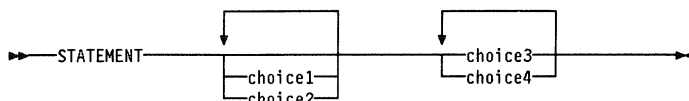
- When you can choose from two or more items, they appear vertically, in a stack. If you *must* choose one of the items, one item of the stack appears on the main path. If choosing an item is optional, the entire stack appears below the main path:



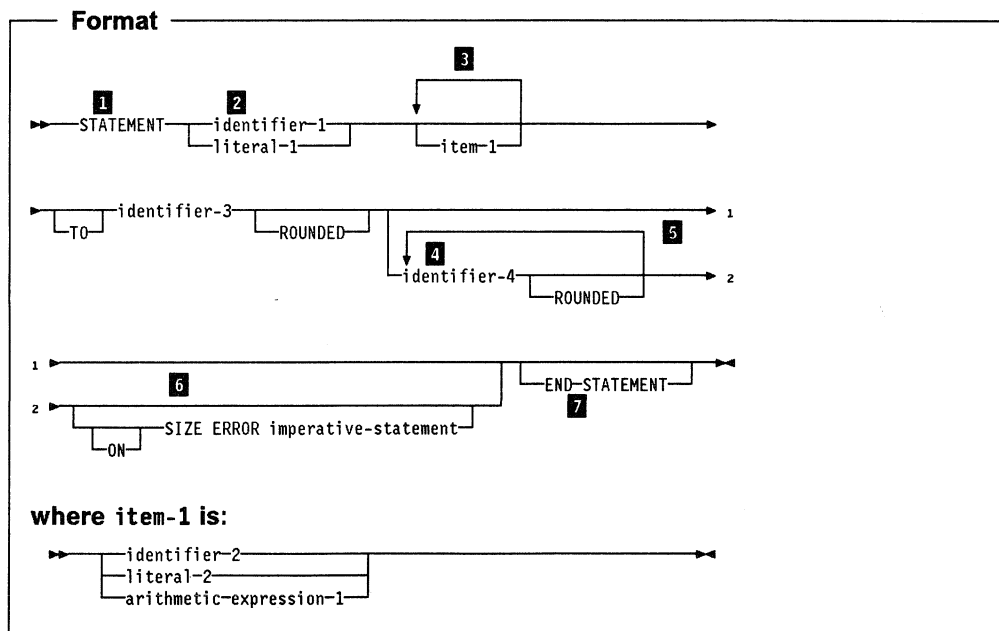
- An arrow returning to the left above an item indicates that this item can be repeated:



- A repeat arrow above a stack of required or optional choices indicates that you can make more than one choice from the stacked items, or repeat a single choice:



The following example shows how the syntax is used:



- 1** The **STATEMENT** keyword must be specified and coded as shown.
- 2** This operand is required. Either *identifier-1* or *literal-1* must be coded.
- 3** The operand *item-1* is optional. It can be coded or not, as required by the application. If coded, it can be repeated, with each entry separated by one or more blanks. Entry selections allowed for this operand are described at the bottom of the diagram.
- 4** The operand *identifier-4* is optional. If specified it may be repeated with one or more blanks separating each entry. Each entry may be assigned the keyword **ROUNDED**.

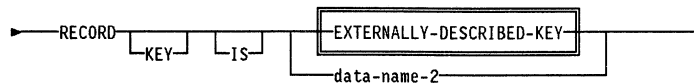
- 5 In cases where multiple lines must be continued past the right margin, line order from top to bottom is preserved.
- 6 The ON keyword is optional to the keyword SIZE ERROR, which is optional itself. If SIZE ERROR is coded, then the operand *imperative-statement* is required.
- 7 The END-STATEMENT keyword can be coded to end the statement. It is not a required delimiter.

Reading IBM Extensions

An IBM extension generally adds to or contradicts a rule or restriction that immediately precedes it. The standard is presented first, because some programmers use the COBOL/400 language without IBM extensions. The extension is then presented for those who *do* use them.

IBM extensions within figures or tables are shown in boxes unless they are explicitly identified as extensions.

Clauses and statements illustrated within syntax diagrams that are COBOL/400 language extensions to ANSI X3.23-1985 COBOL are enclosed in double lines, as follows:

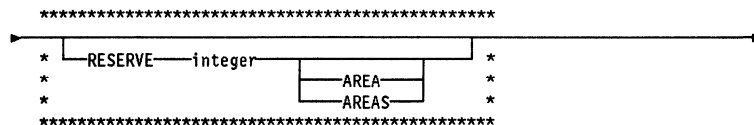


IBM Extension

COBOL/400 language extensions to ANSI X3.23-1985 COBOL that are part of the text description are enclosed in IBM Extension bars, like this paragraph.

End of IBM Extension

COBOL clauses and statements illustrated within syntax diagrams that are syntax checked, but are treated as documentation by the COBOL/400 compiler, are enclosed by asterisks, as follows:



CL Entry Codes

The box that appears in the lower right corner of each CL syntax diagram contains the entry codes that specify the environment in which the command can be entered. The codes indicate whether or not the command can be:

- Used in a batch or interactive job (outside a compiled program; Job:B or I)

- Used in a batch or interactive compiled program (Pgm:B or I)
- Used in a batch or interactive REXX procedure (REXX:B or I)
- Used as a parameter for the CALL CL command, or passed as a character string to the system program QCMDEXC (Exec).

An Overview of COBOL/400 Programming

You follow four major steps or phases to build your COBOL/400 program:

- Entering your source program
- Compiling your source program
- Debugging your program
- Running your compiled program.

Entering Your COBOL Program

The Source Entry Utility (SEU) provides a special display that corresponds to the standard COBOL coding form to help you enter an accurate COBOL source program into the system. SEU also provides a COBOL syntax checker that checks each line for errors as you enter or change them. For information on entering your COBOL/400 source, refer to Chapter 2, “Entering Your Source Program on the AS/400 System.” For more information on using SEU, see the *SEU User’s Guide and Reference*.

Compiling Your COBOL Program

After you have entered the source program into the system, you need to compile the source program using the Create COBOL Program (CRTCLPGM) command. The compiler is called to create a COBOL object program and a listing. An **object program** is a set of instructions in machine-usable form. The object program is produced by a compiler from a source program.

You can specify various compiler options by using the CRTCLPGM command, or by using the PROCESS statement with the desired options. Any options specified in the PROCESS statement override the corresponding options on the CRTCLPGM command. This process is explained in detail in Chapter 3, “Compiling a COBOL/400 Program.”

Debugging Your COBOL Program

The OS/400 operating system provides the following functions that you can use to test and debug your programs:

- Test library
- Breakpoints
- Traces.

The COBOL/400 compiler provides the following functions for program testing and debugging:

- Debugging features
- Formatted dump.

These features allow you to monitor specific program operations during run time. You must decide what to monitor and what information to retrieve for debugging purposes.

See Chapter 5, “Debugging Your Program” for more information on debugging features.

Running Your COBOL Program

You can run your COBOL program many ways, depending on how the program is written, and who is using it. You can run a COBOL program by calling it from a CL program, from an application program, from another high-level language program, or from a user-created command.

When your program has ended, the system returns control to whoever called the program.

For more information on running your program, see Chapter 4, “Running Your COBOL Program.”

Chapter 2. Entering Your Source Program on the AS/400 System

This chapter provides the information you need to enter your program. This chapter also briefly describes the tools and methodology necessary to complete this step.

There are two ways to enter a COBOL source program into the system:

- You can enter your source program using the Source Entry Utility (SEU). This is the method documented in this chapter.
- You can enter your source program from diskette or tape by using the OS/400 copy function.

Refer to the *CL Reference* for more information on how to use the COPY function for entry of the source program from diskette or tape.

To enter your COBOL source program using SEU, enter the Start Source Entry Utility (STRSEU) command, and specify CBL for the TYPE parameter. Refer to the *SEU User's Guide and Reference* for further information on the STRSEU command and using SEU.

Designing Your COBOL/400 Program

You can use the skeleton program, Figure 1 on page 10, as a model for developing readable and efficient COBOL programs. Note that not all the entries provided below are required; most are provided for informational purposes only.

```

IDENTIFICATION DIVISION. 1
PROGRAM-ID. program-name.
AUTHOR. comment-entry.
INSTALLATION. comment-entry.
DATE-WRITTEN. comment-entry.
DATE-COMPILED. comment-entry.
SECURITY.
* The SECURITY paragraph can be used to contain the program
* prologue. The prologue is a description of the program,
* and it may be as detailed or brief as the guidelines of an
* installation recommend. Lowercase letters are recommended
* for comments; however, because some printers can print
* only capital letters, the comments may be printed in
* capitals. The underscores highlight the comments.

ENVIRONMENT DIVISION. 2
CONFIGURATION SECTION. 3
SOURCE-COMPUTER. IBM-AS400.
OBJECT-COMPUTER. IBM-AS400.
SPECIAL-NAMES. REQUESTOR IS CONSOLE.
INPUT-OUTPUT SECTION. 4
FILE-CONTROL.
SELECT file-name ASSIGN TO DISK-file-name
ORGANIZATION IS SEQUENTIAL
ACCESS MODE IS SEQUENTIAL
FILE STATUS IS data-name.

DATA DIVISION. 5
FILE SECTION.
FD file-name
BLOCK CONTAINS 2 RECORDS
RECORD CONTAINS 88 CHARACTERS
LABEL RECORDS ARE STANDARD
DATA RECORD IS record-name
01 record-name PIC X(132).
WORKING-STORAGE SECTION.
77 data-name PIC XX.
LINKAGE SECTION.

PROCEDURE DIVISION. 6
DECLARATIVES
END DECLARATIVES.
main-processing SECTION.
mainline-paragraph.
COBOL statements.
STOP RUN.

```

Figure 1. Example of COBOL/400 Program Structure

The Identification Division **1** is the only division that must be included; all other divisions are optional.

The Environment Division **2** is made up of two sections: the Configuration Section **3**, which describes the overall specifications of the source and object computers, and the Input-Output Section **4**, which defines each file, and specifies information needed for transmission of data between an external medium and the COBOL program.

The Data Division **5** describes the files to be used in the program and the records contained within the files. It also describes any internal working-storage data items that are needed.

The Procedure Division **6** consists of optional declaratives, and procedures that contain sections and/or paragraphs, sentences, and statements.

Source File Format

The standard record length of your source files is 92 characters. These 92 characters are made up of a 6-character sequence number, a 6-character date-last-modified area, and an 80-character data field.

The COBOL/400 compiler supports an additional record length of 102; a field of 10 characters containing supplementary information is placed at the end of the record (positions 93-102). This information is not used by the COBOL compiler, but is placed on the extreme right of the compiler listing. You are responsible for placing information into this field. If you want to use this additional field, create a source file with a record length of 102.

IBM supplies a source file where you can store your source records if you do not want to create your own file. This file, named QLBSLRC, is in library QGPL and has a record length of 92 characters.

Entering Source Using SEU

SEU provides special display formats for COBOL. They correspond to the COBOL Coding Form and are designed to help you enter your COBOL source programs. Figure 2 shows a display format, the relationship between the headings on the COBOL Coding Form, and the labels on the display; it also identifies where you enter the source code.

SEU can display a format line to help you make changes or key in entries, position by position.

```

Columns . . . : 1 71          Edit          QGPL/QLBLSRC
SEU=>
FMT CB .....-A+++B+++++*****
***** Beginning of data *****
0001.00      ENVIRONMENT DIVISION.
0002.00      CONFIGURATION SECTION.
0003.00      SOURCE-COMPUTER.  IBM-AS400.
0004.00      INPUT-OUTPUT SECTION.
0005.00      FILE-CONTROL.
*****
0006.00      SELECT FILE-1 ASSIGN TO DATABASE-MASTER.
0007.00      SELECT FILE-2 ASSIGN TO DATABASE-MASTER.
***** End of data *****

Prompt type . . .  CB      Sequence number . . .  0005.00

Continuation

Area-A      Area-B
FILE      -CONTROL.

F3=Exit  F4=Prompt  F5=Refresh      F11=Previous record
F12=Cancel      F23=Select prompt  F24=More keys

```

Figure 2. An SEU Display Format

For a complete description of how to enter a source program using SEU, refer to the *SEU User's Guide and Reference*.

Using the COBOL Syntax Checker in SEU

To use the COBOL syntax checker in SEU, specify the TYPE(CBL) parameter of the STRSEU command. The COBOL syntax checker checks each line for errors as you enter new lines or change existing lines. Incorrect source statements are identified and error messages displayed, allowing you to correct the errors before compiling the program. Because the COBOL syntax checker checks only one statement at a time, independently of statements that precede or follow it, only syntax errors within the source data can be detected. No interrelational errors, such as undefined names and incorrect references to names, are detected. These errors are detected by the COBOL compiler when the program is compiled.

Any time a source line is entered or changed, up to 496 lines of source code can be syntax checked as one unit. The length of a single unit of syntax-checking is determined by extending from an entered or changed line as follows:

A unit of syntax-checking extends towards the beginning of the source member until the first source line, or a line that ends in a period is found.

A unit of syntax-checking extends towards the end of the source member until the last source line, or a line that ends in a period is found.

If this unit spans more than 496 source lines (not including comment lines), the system responds with an appropriate message.

If there is an error in a unit of syntax-checking, the entire unit is presented in reverse image. The message at the bottom of the display refers to the first error in the unit.

Syntax checking occurs line by line as you enter the source code. Error messages are generated by lines consisting of incomplete statements. These disappear when the statements are completed, as in the example:

```
ADD A
  TO BCD.
```

An error message is generated after the first line is entered and disappears after the second line is entered, when the statement is completed. A COBOL sentence can span a maximum of 496 lines. Also, if a source line is entered or changed, up to 496 lines of source code can be syntax checked as one unit.

The following regulations apply to syntax checking for COBOL source functions:

- Source code on a line with an asterisk (*) or a slash (/) in column 7 is not syntax checked. An asterisk indicates a comment line; a slash indicates a comment line and page eject.
- No compiler options are honored during syntax checking.

For example, the syntax checker accepts both quotation marks or apostrophes as nonnumeric delimiters provided they are not mixed within one unit of syntax checking. The syntax checker does not check if the delimiter is the one that will be specified in the CRTCLPGM command for compiling COBOL source statements, or in the PROCESS statement.

- The first sentence following any of the paragraph headers listed below must begin on the same line as the paragraph header.

```
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
```

- Character replacement specified by the CURRENCY and DECIMAL-POINT clauses of the SPECIAL-NAMES paragraph is not honored during interactive syntax checking.
- When using the REPLACING *Identifier-1* BY *Identifier-2* clause of the COPY statement and when either identifier includes reference modification, SEU checks for matching parentheses only. For more information on reference modification, see Chapter 11, "COBOL/400 Programming Considerations."

Syntax for Structured Query Language (SQL) Statements

The syntax for SQL statements embedded in a COBOL source program is:

```
►—EXEC SQL—sql-statement—END-EXEC.—◄
```


If the member type for the source program is SQLCBL or CICSSQLCBL, when the COBOL syntax checker encounters an SQL statement, the statement is passed to the SQL syntax checker. If an error is detected, a message is returned.

If an SQL statement is encountered, and if the member type is not SQLCBL or CICSSQLCBL, a COBOL message is returned indicating that a COBOL statement is in error.

If there are errors in the embedded SQL statement as well as errors in the preceding COBOL statements, the SQL error message will only be displayed after the preceding COBOL errors are corrected.

For more information about SQL statements, refer to the *SQL/400* Reference*.

Syntax for Customer Information Control System (CICS) Statements

The syntax for CICS statements embedded in a COBOL source program is:

```
▶—EXEC CICS—cics-statement—END-EXEC.—————▶
```

If the member type for the source program is CICSCBL or CICSSQLCBL, when the COBOL syntax checker encounters a CICS statement, the COBOL syntax checker checks for only basic syntax errors.

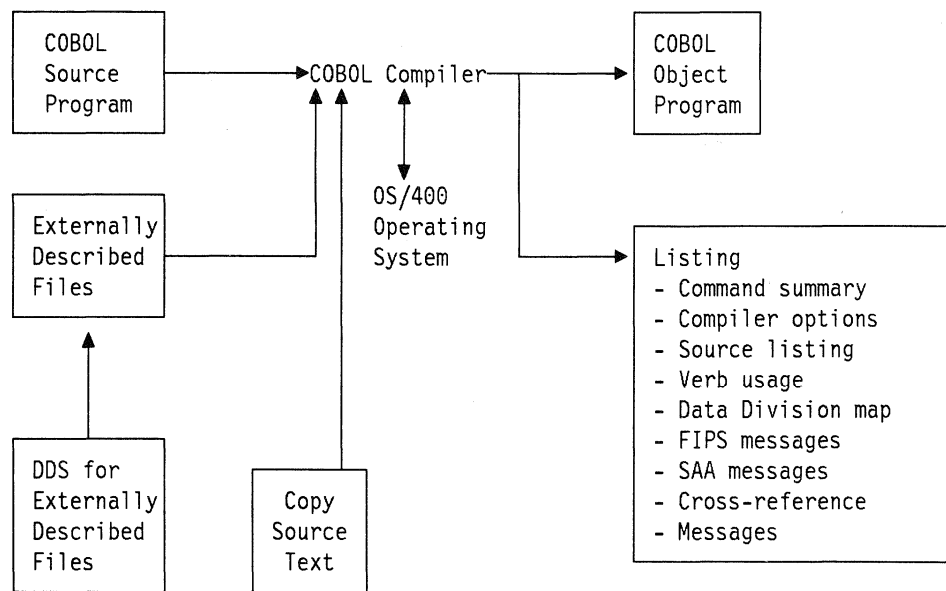
If a CICS statement is encountered, and if the member type is not CICSCBL or CICSSQLCBL, a COBOL message is returned indicating that a COBOL statement is in error.

For more information about CICS/400 statements, refer to the *CICS/400 Application Programming Guide*.

Chapter 3. Compiling a COBOL/400 Program

You need to compile the COBOL/400 source program to produce a usable object program. You do this using the Create COBOL Program (CRTCLPGM) command. The result of the compilation is a COBOL object program and a listing.

You can specify various compiler options by using the CRTCLPGM command, or from within the program using the PROCESS statement. Any options specified in the PROCESS statement override the corresponding options on the CRTCLPGM command. The PROCESS statement is discussed later in "Using the PROCESS Statement to Specify Compiler Options" on page 31.



During compilation, the compiler checks the syntax of the COBOL source program line by line, and also checks the relationships between the lines.

Using the Create COBOL Program (CRTCLPGM) Command

To compile a COBOL/400 source program into an object program, you must enter the CRTCLPGM command. This calls the COBOL/400 compiler. You can use the CRTCLPGM command interactively, or in batch jobs, or from other programs on the AS/400 system.

Programming Note: The number of entries in the Object Definition Table (ODT) and the amount of storage required by a COBOL program varies with the number and kinds of COBOL statements used in the program. A combination of these factors can cause the AS/400 internal size limits for the program to be exceeded. If this occurs, try using the *NOUNREF option of the GENOPT parameter. If the problem persists, the program must be rewritten.

When the *NOUNREF option is specified, only names that are referenced or are needed for data structuring are defined. This option is useful when the program contains many unreferenced identifiers.

If you do not specify CBL as the source member type, the compiler issues a warning.

If the Format 2 COPY statement is used in the program to access externally described files, the operating system provides information about the externally described files to the compiled program.

If the COBOL compiler stops, the message LBL9001

Compile failed. Program not created.

is issued. You can use a control language program that can monitor for this exception by using the control language Monitor Message (MONMSG) command.

Using the CRTCLPGM Prompt Displays

To enter the CRTCLPGM command from the CRTCLPGM prompt displays, type CRTCLPGM and press F4 (Prompt) to show the first display. The parameters and options are described in the order they appear on these displays, on pages 18 through 27. The default values are explained first, and are underlined.

Each parameter on this display shows a default value. Move the cursor past the items where you want default values to apply. Type over any items to set different values or options. If you are unsure about the setting of a parameter value, type a question mark (?) in the first position of the field and press Enter, or F4 (Prompt), to receive more detailed information. The question mark must be followed by a blank.

Figure 3 shows the CRTCLPGM prompt displays. When you see the first CRTCLPGM prompt display, you see only the required parameters prompted. To see the additional parameters, press F10. You see the first display shown in Figure 3. To see the remainder of the parameters, as shown in the second and third displays in Figure 3, page forward.

```

Create COBOL Program (CRTCLPGM)

Type choices, press Enter.

Program . . . . . *PGMID      Name, *PGMID
Library . . . . . *CURLIB     Name, *CURLIB
Source file . . . . . QLBSRC    Name
Library . . . . . *LIBL       Name, *LIBL, *CURLIB
Source member . . . . . *PGM      Name, *PGM
Generation severity level . . . . . 29      0-29
Text 'description' . . . . . *SRCMBRTXT

Additional Parameters

Source listing options . . . . . *SOURCE, *NOSOURCE, *SRC...
+ for more values
Generation options . . . . . *NOLIST, *LIST, *NOXREF...
+ for more values

More...
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

```

```

Create COBOL Program (CRTCLPGM)

Type choices, press Enter.

Conversion options . . . . . *NOVARCHAR, *VARCHAR...

Message Limit:
Number of messages . . . . . *NOMAX      1-9999, *NOMAX
Message limit severity . . . . . 29      0-29
Print file . . . . . QSYSVRT      Name
Library . . . . . *LIBL         Name, *LIBL, *CURLIB
FIPS flagging . . . . .
+ for more values
SAA flagging . . . . . *NOFLAG    *NOFLAG, *FLAG
Extended display options . . . . . *DFRWRT, *NODFRWRT...
+ for more values
Flagging severity . . . . . 0      0-99
Replace program . . . . . *YES      *NO, *YES
Target release . . . . . *CURRENT  *CURRENT, *PRV, V2R1M0...
User profile . . . . . *USER     *USER, *OWNER

More...
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

```

```

Create COBOL Program (CRTCLPGM)

Type choices, press Enter.

Authority . . . . . *LIBCRTAUT  Name, *LIBCRTAUT, *ALL...
Compiler debugging dump:
1      1-65535, *
65535  1-65535
Intermediate text dump . . . . . 0      0-31

Bottom
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

```

Figure 3. The CRTCLPGM Prompt Displays

Parameters of the CRTCLPGM Command

A description of the parameters for the CRTCLPGM command follows. The default values are explained first, and are underscored for identification. The parameters and options are described in the order they appear on the prompt displays.

All object names specified for the CRTCLPGM command must follow AS/400 naming conventions: the names may be basic names, 10 characters in length, composed of alphanumeric characters, the first of which must be alphabetic; or the names may be quoted names, 8 characters in length, enclosed in double quotes.

If you want to relate these parameter descriptions to the CRTCLPGM syntax diagram, refer to Figure 4 on page 29.

PGM Parameter:

Specifies the program name and library name for the COBOL program object you are creating. The possible values are:

***PGMID**

The name for the program object is taken from the PROGRAM-ID paragraph in the COBOL source program.

program-name

Enter a name to identify the compiled COBOL program. If you specify a program name for this parameter, and run the compilation in batch mode, the first program in the batch job uses this name; any other programs use the name specified in the PROGRAM-ID paragraph in the source program.

The possible library values are:

***CURLIB**

If you do not specify a library name, the current library is used. If you have not assigned a library as the current library, QGPL is used.

library-name

Enter the name of the library to contain the created program object.

SRCFILE Parameter:

Specifies the name of the source file that contains the COBOL source to be compiled. The possible values are:

QLBLSRC

Specifies that the source file, QLBLSRC, contains the COBOL source to be compiled.

source-file-name

Enter the name of the source file that contains the COBOL source to be compiled. This source file should have a record length of 92.

The possible library values are:

***LIBL**

If you do not specify a library name, the system searches the library list to find the library where the source file is located.

***CURLIB**

The current library is used. If you have not assigned a library as the current library, QGPL is used.

library-name

Enter the name of the library where the source file is located.

SRCMBR Parameter:

Specifies the name of the member that contains the COBOL source to be compiled. You can specify this parameter only if the source file referenced in the SRCFILE parameter is a database file. The possible values are:

***PGM**

If you specified a program name for the PGM parameter, the compiler looks for the source program in a member having the same name as the program, and creates an object program with the same name as the program and member.

If you did not specify a program name for the PGM parameter, the compiler looks for the program source in the first member of the database source file, and creates an object program using the name specified in the PROGRAM-ID paragraph.

source-file-member-name

Enter the name of the member that contains the COBOL source.

GENLVL Parameter:

Specifies the severity level that determines if a program object is created. The severity level corresponds to the severity level of the messages produced during compilation of the program. If the severity level of error messages is greater than the value you specify, a program object is not created. For example, if you specify 19 for this parameter, a program object is not created if the severity level of any of the messages is 20 or greater.

The possible values are:

29 If errors occur with a severity level greater than 29, no program object is created.

severity-level

Specify a one or two-digit number, 0 through 29. If errors occur with a severity level greater than this level, no program object is created.

TEXT Parameter:

Allows you to enter text that briefly describes the program and its function.

***SRCMBRTXT**

Use the same text for the program object as that which describes the database file member containing the COBOL source. If the source comes from a device or in-line file, specifying *SRCMBRTXT has the same effect as specifying *BLANK.

***BLANK**

No text is specified.

text-description

Enter the text that briefly describes the program and its function. The text can be a maximum of 50 characters in length and must be enclosed in apostrophes. The apostrophes are not part of the 50-character string.

OPTION Parameter:

Specifies the options to use when the COBOL source is compiled. The possible values are:

***SOURCE or *SRC**

The compiler produces a source listing, consisting of the COBOL source input and all compilation-time error messages.

***NOSOURCE or *NOSRC**

The compiler does not produce the source part of the listing. If you do not require a source listing, you should use this option because compilation may take less time.

***NOXREF**

The compiler does not produce a cross-reference listing for the source program.

***XREF**

The compiler produces a cross-reference listing for the source program.

***GEN**

The compiler creates a program object after the program is compiled.

***NOGEN**

The compiler does not create a program object after the source program is compiled. You might specify this option if you want only error listings at this time.

***NOSEQUENCE**

The reference numbers are not checked for sequence errors.

***SEQUENCE**

The reference numbers are checked for sequence errors. Sequence errors do not occur if the *LINENUMBER option is specified.

***NOVBSUM**

Verb usage counts are not printed.

***VBSUM**

Verb usage counts are printed.

***NONUMBER**

The source-file sequence numbers are used for reference numbers.

***NUMBER**

The user-supplied sequence numbers (columns 1 through 6) are used for reference numbers.

***LINENUMBER**

The sequence numbers created by the compiler are used for reference numbers. This option combines program source code and source code introduced by COPY statements into one consecutively numbered sequence. Use this option if you specify FIPS (Federal Information Processing Standards) flagging or SAA* flagging.

***NOMAP**

The compiler does not list the Data Division map.

***MAP**

The compiler lists the Data Division map.

***NOOPTIONS**

Options in effect are not listed for this compilation.

***OPTIONS**

Options in effect are listed for this compilation.

***QUOTE**

Specifies that the delimiter quotation mark (") is used for nonnumeric literals and Boolean literals. This also specifies that the value of the figurative constant QUOTE has the EBCDIC value of a quotation mark.

Note: Boolean data is a category of data items that are limited to a value of 1 or 0. A Boolean literal is a literal composed of a Boolean character enclosed in quotation marks and preceded by a B; for example: B"1".

***APOST**

Specifies that the delimiter apostrophe (') is used for nonnumeric literals and Boolean literals. This also specifies that the value of the figurative constant QUOTE has the EBCDIC value of an apostrophe.

***NOSECLVL**

Second level message text is not listed for this compilation.

***SECLVL**

Second level message text is listed for this compilation, along with the first-level error text.

Note: The first-level error text is printed each time the error occurs.

***PRTCORR**

The compiler inserts comment lines in the compiler listing indicating which elementary items were included as a result of the use of the CORRESPONDING phrase.

***NOPRTCORR**

The compiler does not insert comment lines in the compiler listing when the CORRESPONDING phrase is used.

Note: You can only use the *SRCDBG and *LSTDBG options if you are using the AD/Cycle CODE/400* product to compile your program. If you specify any of these options but do not have the AD/Cycle CODE/400* product installed, the COBOL/400 compiler continues processing. For more information on these options, see the *IBM SAA AD/Cycle CoOperative Development Environment: Debug Tool User's Reference*.

***NOSRCDBG**

This option determines the kind of information you see on your programmable work station when using the AD/Cycle CoOperative Development Environment/400 to compile your COBOL programs.

The compiler does not produce source-level debugging information. If *NOLSTDBG is also in effect, the compiler does not produce source-level error information either.

***SRCDBG**

This option determines the kind of information you see on your programmable work station when using the AD/Cycle CoOperative Development Environment/400 product to compile your COBOL programs.

The compiler produces source-level error information and source-level debugging information.

You cannot specify *SRCDBG and *LSTDBG together. Specify one or the other.

***NOLSTDBG**

This option determines the kind of information you see on your programmable work station when using the AD/Cycle CoOperative Development Environment/400 product to compile your COBOL programs.

The compiler does not produce a listing view, source-level error information, or listing-level debugging information.

***LSTDBG**

This option determines the kind of information you see on your programmable work station when using the AD/Cycle CoOperative Development Environment/400 product to compile your COBOL programs.

The compiler produces a listing view, and listing-level debugging information. If *NOSRCDBG is also in effect, the compiler does not produce source-level error information either.

You cannot specify *SRCDBG and *LSTDBG together. Specify one or the other.

***PRINT**

The compiler produces a spool listing.

***NOPRINT**

The compiler does not produce a spool listing.

GENOPT Parameter:

Specifies the options to use when the program object is created. The listings

could be required if a problem occurs in COBOL. The possible values are:

***NOLIST**

No IRP (intermediate representation of program), associated hexadecimal code, or error messages are listed.

***LIST**

The IRP, its associated hexadecimal code, and any error messages are listed.

***NOXREF**

Does not produce a cross-reference listing of the objects defined in the IRP.

***XREF**

Produces a cross-reference listing of all objects defined in the IRP.

***NOPATCH**

Does not reserve space in the compiled program for a program patch area.

***PATCH**

Reserves space in the compiled program for a program patch area. The program patch area can be used for debugging purposes.

***NODUMP**

Does not list the program template.

***DUMP**

Lists the program template.

***NOATR**

Does not list the attributes for the IRP source.

***ATR**

Lists the attributes for the IRP source.

***RANGE**

At runtime, the system verifies that subscripts are within the correct ranges, but does not verify index ranges. It also checks for reference modification and compiler-generated substring operations.

***NORANGE**

Does not verify ranges at run-time.

Note: The *RANGE option generates code for checking subscript ranges. For example, it ensures that you are not attempting to access the 21st element of a 20-element array.

The *NORANGE option does not generate code to check subscript ranges.

These options do not eliminate the zero subscript checking performed by the operating system. If zero subscripts occur, the operating system will not permit their use and issues message MCH0603.

***UNREF**

Includes unreferenced data items in the compiled program.

***NOUNREF**

Does not include unreferenced data items in the compiled program. This reduces the number of ODT (object definition table) entries used, allowing a larger program to be compiled. The unreferenced data items still appear in the cross-reference listings produced by specifying OPTION (*XREF).

***NOOPTIMIZE**

The compiler performs only standard optimizations for the program.

***OPTIMIZE**

The compiler attempts to create a program that operates more efficiently and uses less storage. However, specifying *OPTIMIZE can substantially increase the time required to compile a program.

***NODDSFILLER**

If no matching fields are found by a COPY DDS statement, no field descriptions are generated.

***DDSFILLER**

When no matching fields are found by a COPY DDS statement, a single character FILLER field description, "07 FILLER PIC X", is always created.

***NOSYNC**

The SYNCHRONIZED clause is syntax checked.

***SYNC**

The SYNCHRONIZED clause causes the alignment of an elementary item on a natural boundary in storage.

***NOCRTE**

Files that are unavailable at the time of an OPEN operation are not created dynamically.

***CRTF**

Files that are unavailable at the time of an OPEN operation are created dynamically. (See the discussion of the OPEN statement in the *COBOL/400* Reference* for the conditions under which dynamic file creation can occur.)

Note: The maximum record length for a file that will be created dynamically is 32 766.

***NODUPKEYCHK**

Does not check for duplicate keys for INDEXED files.

***DUPKEYCHK**

Checks for duplicate keys for INDEXED files. (See the discussion of the READ statement in the *COBOL/400* Reference* for the conditions under which dynamic file creation can occur.)

***STDERR**

Standard error handling is used. See Chapter 6, "COBOL/400 Exception and Error Handling" on page 67 for more information.

***NOSTDERR**

The error handling method of Version 1, Releases 1 and 2, is used.

***NOEXTACCDSP**

The compiler does not allow extended ACCEPT or DISPLAY statements.

***EXTACCDSP**

The compiler allows extended ACCEPT and DISPLAY statements. Refer to Appendix E of the *COBOL/400* Reference* for changes to the reserved word list that occur when you use this option.

***NOINZDLT**

Relative files with sequential access are not initialized with deleted records during the CLOSE operation if the files have been opened for OUTPUT. That is, the record boundary is determined by the number of records written. Subsequent OPEN operations allow access only up to the record boundary.

***INZDLT**

Relative files with sequential access are initialized with deleted records during the CLOSE operation if the files were opened for OUTPUT. Active records in the files are not affected. That is, the record boundary is defined as the file size for subsequent I/O operations.

***NOBLK**

The compiler allows blocking only of SEQUENTIAL access files with no START statement.

If a BLOCK CONTAINS clause is specified, the BLOCK CONTAINS clause is ignored, except for tape files.

***BLK**

When used with BLOCK CONTAINS, the compiler allows blocking from DYNAMIC access files and SEQUENTIAL access files with a START statement. Blocking is not allowed for RELATIVE files opened for output operations.

The BLOCK CONTAINS clause controls the number of records to be blocked.

When no BLOCK CONTAINS clause is specified, the compiler allows blocking

only of SEQUENTIAL access files with no START statement. The operating system determines the number of records to be blocked.

***STDINZ**

The compiler initializes user data items to system defaults, provided that the items are not subject to a VALUE clause.

***NOSTDINZ**

For those user items with no VALUE clause, the compiler does not initialize data items to system defaults.

***FS21DUPKY**

The compiler reports a file status of 21 when processing an indexed file with duplicate keys in random or dynamic access mode, if the value of the key is changed between the mandatory READ statement and a following REWRITE or DELETE statement.

***NOFS21DUPKY**

The compiler does not report a file status of 21 when processing an indexed file with duplicate keys in random or dynamic access mode. A REWRITE statement can change the key of a record.

CVTOPT Parameter:

Specifies how the compiler handles SAA date, time, and timestamp data types, DBCS-graphic data types, and variable-length character fields passed from externally-described files to your program through COPY DDS. The possible values are:

***NOVARCHAR**

Variable-length fields are ignored, and are declared as FILLER fields.

***VARCHAR**

Variable-length fields are declared as fixed-length group items, and are accessible to the program. For more information on variable-length fields, refer to "Declaring Data Items Using CVTOPT Data Types" on page 127.

***NODATETIME**

Date, time, and timestamp data types are ignored, and are declared as FILLER fields.

***DATETIME**

Date, time, and timestamp data types are declared as fixed-length character fields, and are accessible to the program.

***NOGRAPHIC**

DBCS-graphic data types are ignored, and are declared as FILLER fields.

***GRAPHIC**

Fixed-length DBCS-graphic data types are declared as fixed-length alphanumeric fields, and are accessible to the program.

When the *VARCHAR option is also in use, variable-length DBCS-graphic data types are declared as fixed-length group items, and are accessible to the program. For more information on DBCS-graphic data types, refer to "DBCS-Graphic Fields" on page 129.

MSGLMT Parameter:

Controls compilation by indicating the maximum number of error messages of a given error severity level that can occur before compilation stops.

For example, you can stop compilation if more than three errors with a severity level of 20 or higher occur. In this example, you would specify 3 for the maximum number of error messages, and 20 for the maximum error severity level. If three errors of severity level 20 or higher occur, compilation continues, but when a fourth is encountered, compilation stops. If no messages equal or exceed the maximum severity level, compilation continues regardless of the number of errors encountered.

message-limit

The possible values for the maximum number of error messages are:

***NOMAX**

Compilation continues until normal completion regardless of the number of errors encountered.

1-9999

Compilation stops if the number of errors of the specified severity level or higher exceeds the number you specify. If no messages equal or exceed the maximum severity level, compilation continues regardless of the number of errors encountered.

message-severity

The possible values for the maximum error severity level are:

29 Compilation stops if the number of errors with severity level 29 or higher exceeds the maximum number of error messages specified.

maximum-severity-level

Specify a one or two-digit number, 0 through 29. Compilation stops if the number of errors with the specified severity level or higher exceeds the maximum number of error messages you specify.

PRTFILE Parameter:

Specifies the name of the file to which the compiler listing is directed and the library where the file is located. The file should have a minimum record length of 132. If a file with a record length less than 132 is specified, information is lost.

The possible values are:

QSYSPRT

If you do not specify a file name, the compiler listing is directed to QSYSPRT, an IBM-supplied file.

file-name

Enter the name of the file to which the compiler listing is directed.

The possible library values are:

***LIBL**

If a library-name is not specified, the system searches the library list, *LIBL, to find the library where the file is located.

***CURLIB**

The current library is used. If you have not assigned a library as the current library, QGPL is used.

library-name

Enter the name of the library where the file is located.

FLAGSTD Parameter:

Specifies the options for FIPS flagging. (Select the *LINENUMBER option to ensure that the reference numbers used in the FIPS messages are unique.) The possible values are:

***NOFIPS**

The source program is not FIPS flagged.

***MINIMUM**

FIPS flag for minimum subset and higher.

***INTERMEDIATE**

FIPS flag for intermediate subset and higher.

***HIGH**

FIPS flag for high subset.

***NOSEG**

The optional module SEGMENTATION is not FIPS flagged.

***SEG1**

FIPS flag for optional module SEGMENTATION level 1 and higher.

***SEG2**

FIPS flag for optional module SEGMENTATION level 2.

***NODEB**

The optional module DEBUG is not FIPS flagged.

***DEB1**

FIPS flag for optional module DEBUG level 1 and higher.

***DEB2**

FIPS flag for optional module DEBUG level 2.

***NOOBSOLETE**

Obsolete language elements are not flagged.

***OBSOLETE**

Obsolete language elements are flagged.

SAAFLAG Parameter:

Specifies if you want flagging of COBOL/400* functions that are not supported by SAA COBOL. (Select the *LINENUMBER option to ensure that the reference numbers used in the SAA COBOL messages are unique.) The possible values are:

***NOFLAG**

SAA COBOL flagging is not performed.

***FLAG**

SAA COBOL flagging is performed.

EXTDSPOPT Parameter:

Specifies the options to use for extended ACCEPT and extended DISPLAY statements for work station I/O. The possible values are:

***DFRWRT**

Extended DISPLAY statements are held in a buffer until an extended ACCEPT statement is encountered, or until the buffer is filled.

If an extended ACCEPT statement is not encountered before the buffer is filled, the contents of the buffer are written to the display. When an extended ACCEPT statement is encountered, the current contents of the buffer are written to the display.

***NODFRWRT**

Each extended DISPLAY statement is performed as it is encountered.

***UNDSPCHR**

Displayable and undisplayable characters are handled by extended ACCEPT and extended DISPLAY statements.

***NOUNDSPCHR**

Only displayable characters are handled by extended ACCEPT and extended DISPLAY statements.

Although you must use this option for display stations attached to remote 3174 and 3274 controllers, you can also use it for local work stations. If you do use this option, your data must contain displayable characters. If the data contains values less than hexadecimal 20, the results are not predictable, ranging from

unexpected display formats to severe errors.

***ACCUPDALL**

All types of data are predisplayed in the extended ACCEPT statements regardless of the existence of the UPDATE phrase.

***ACCUPDNE**

Only numeric edited data are predisplayed in the extended ACCEPT statements that do not contain the UPDATE phrase.

FLAG Parameter:

Specifies the minimum severity level of messages to be printed. The possible values are:

0 All messages are printed.

severity-level

Enter a one or two-digit number that specifies the minimum severity level of messages to be printed. Messages that have severity levels of the specified value or higher are listed.

REPLACE Parameter:

Specifies if a new program object is created when a program object of the same name in the same library already exists. The possible values are:

***YES**

A new program object is created and any existing program object of the same name in the specified library is moved to library QRPLOBJ.

***NO**

A new program object is not created if a program object of the same name already exists in the specified library.

TGTRLS Parameter:

Specifies the release of the operating system on which you intend to use the object being created. You can specify an exact release level in the format VxRxMx, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V2R2M0 is version 2, release 2, modification level 0.

Note: To use the object on the target system, you must save the object to the target release level specified on

the create command and then restore it on the target system.

The possible values are:

***CURRENT**

The object is to be used on the release of the operating system currently running on your system. You can also use the object on a system with any subsequent release of the operating system installed.

***PRV**

The object is to be used on the previous release with modification level 0 of the operating system. You can also use the object on a system with any subsequent release of the operating system installed.

release-level

Specify the release in the format VxRxMx. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release.

USRPRF Parameter:

Specifies the user profile that will run the compiled COBOL program. The profile of the program owner or the program user is used to run the program and control which objects can be used by the program (including the authority the program has for each object). This parameter is not updated if the program already exists. To change the value of USRPRF, delete the program and recompile using the correct value.

The possible values are:

***USER**

The user profile of the program user is to be used when the program is run.

***OWNER**

The user profiles of both the program's owner and user are to be used when the program is run. The collective sets of object authority in both user profiles are to be used to find and access objects during the running of the program. Any objects that are created during the program are owned by the program's user.

Note: Specify the USRPRF parameter to reflect the security requirements of your installation. The security facilities available on the AS/400 system are described in detail in the *Security Reference*.

AUT Parameter:

Specifies the authority given to users who do not have specific authority to the program object, who are not on the authorization list, or whose group has no specific authority to the program object. You can alter the authority for all users, or for specific users after the program object is created by using the GRTOBJAUT (Grant Object Authority) or RVKOBJAUT (Revoke Object Authority) commands.

The possible values are:

***LIBCRTAUT**

The public authority for the object is taken from the CRTAUT keyword of the target library (the library that is to contain the created program object). This value is determined when the program object is created. If the CRTAUT value for the library changes after the program object is created, the new value does NOT affect any existing objects.

***ALL**

Provides authority for all operations on the program object except those limited to the owner or controlled by authorization list management authority. The user can control the program object's existence, specify security for it, change it, and perform basic functions on it, but cannot transfer its ownership.

***CHANGE**

Provides all data authority and the authority for performing all operations on the program object except those limited

to the owner or controlled by object authority and object management authority. The user can change the object and perform basic functions on it, such as running and debugging the program object.

***USE**

Provides object operational authority and read authority; authority for basic operations on the program object such as running the program. The user is prevented from changing the object.

***EXCLUDE**

The user cannot access the program object.

authorization-list-name

Enter the name of an authorization list of users and authorities to which the program is added. The program object is secured by this authorization list, and the public authority for the program object is set to *AUTL. The authorization list must exist on the system when the CRTCLPGM command is issued. Use the Create Authorization List (CRTAUTL) command to create your own authorization list.

Note: Specify the AUT parameter to reflect the security requirements of your installation. The security facilities available on the AS/400 system are described in detail in *Security Reference*.

DUMP Parameter:

An IBM COBOL/400 debugging aid for IBM service personnel.

ITDUMP (n) Parameter:

An IBM debugging aid provided for IBM service personnel. This parameter makes the compiler dump the internal text at certain times during the compilation of the source program.

Entering CRTCLPGM from the Command Line

You can enter the CRTCLPGM command from the command line. Type CRTCLPGM followed by the appropriate parameters to compile your program. Refer to the Figure 4 on page 29 for the correct syntax. If you are unsure about the parameters and their meanings, refer to the parameter and option descriptions on pages 18 through 27. Refer to the following examples of the syntax you would use to enter the CRTCLPGM command from the command line.

Example 1

```
CRTCLPGM SRCFILE(QGPL/QLBLSRC) SRCMBR(SAMPLE) SAAFLAG(*FLAG)
```

Partial Source for Member SAMPLE

```
ID DIVISION.  
PROGRAM-ID. EXAMPLE.
```

The preceding example creates a COBOL/400 program from the source member SAMPLE in file QLBLSRC and library QGPL. The resulting program is called EXAMPLE. Specifying *FLAG for the SAAFLAG parameter tells the compiler to identify any functions that are not supported by SAA COBOL. In this example, all parameter defaults were used with the exception of the SRCFILE, SRCMBR, and SAAFLAG parameters.

Example 2

```
CRTCLPGM PGM(TEST) SRCFILE(SOURCE1) CVTOPT(*GRAPHIC)
```

In the preceding example, the compiler looks for the file SOURCE1 in the library list, and looks for the member called TEST within that file. (The value for the SRCMBR parameter defaulted to *PGM, specifying to look for a member with the same name as the program to be created.) The compiler creates a COBOL/400 program called TEST from the source program found in the member TEST in the file SOURCE1. Specifying *GRAPHIC for the CVTOPT parameter indicates that if the DDS contains DBCS-graphic data types, you want the COBOL program to be able to reference them as alphanumeric fields.

Entering CRTCLPGM from a CL Program

When you issue the CRTCLPGM command from a CL program, you can use concatenation expressions for all parameter values. See the *CL Reference* for more information about concatenation expressions. Also, see the *CL Reference* for a detailed description of OS/400 object naming rules and for a complete description of OS/400 command syntax.

General-Use Programming Interface

You can use this command in QCMDXEC.

End of General-Use Programming Interface

Syntax of the CRTCLPGM Command

Figure 4 shows the syntax of the CRTCLPGM command.

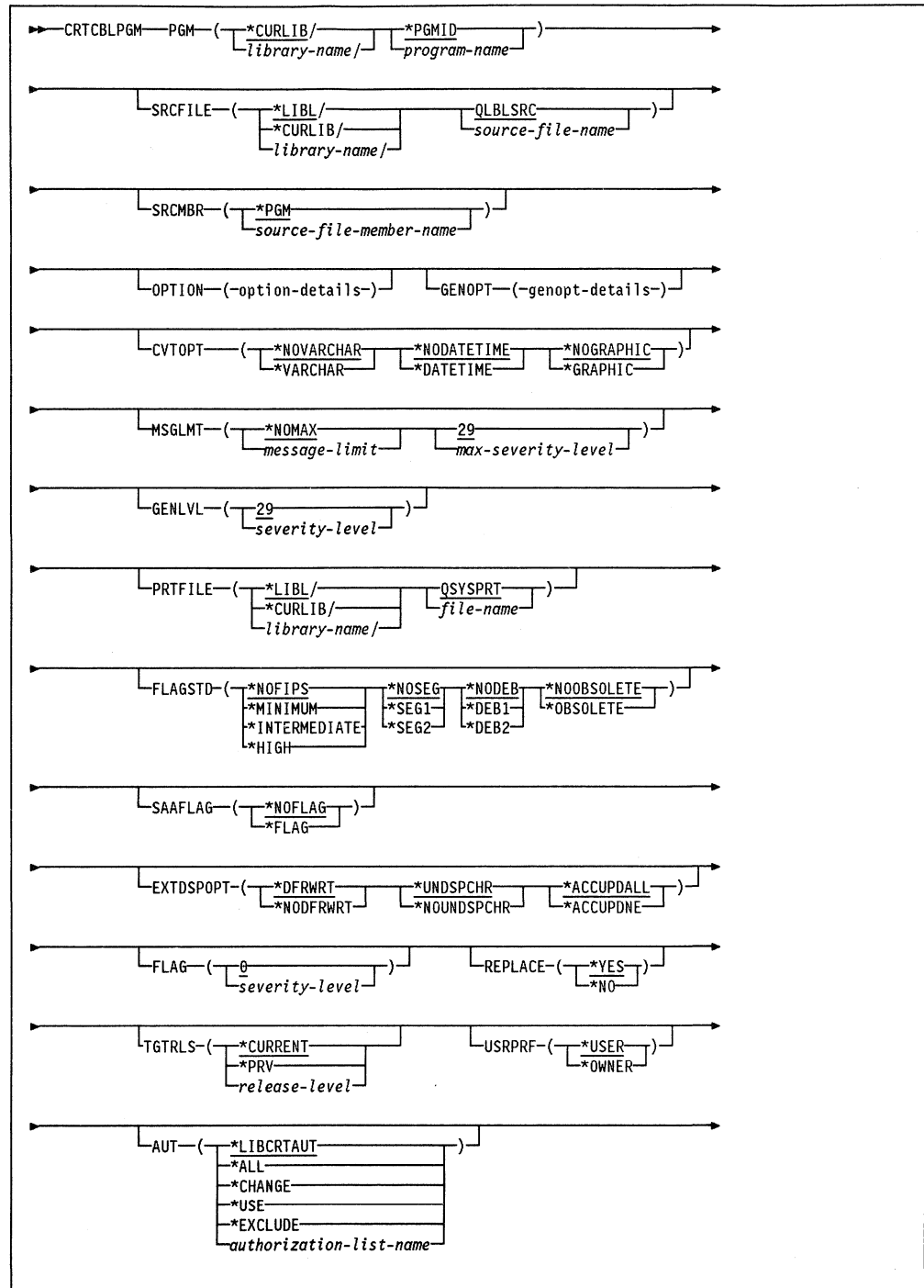


Figure 4 (Part 1 of 2). Syntax of the CRTCLPGM Command

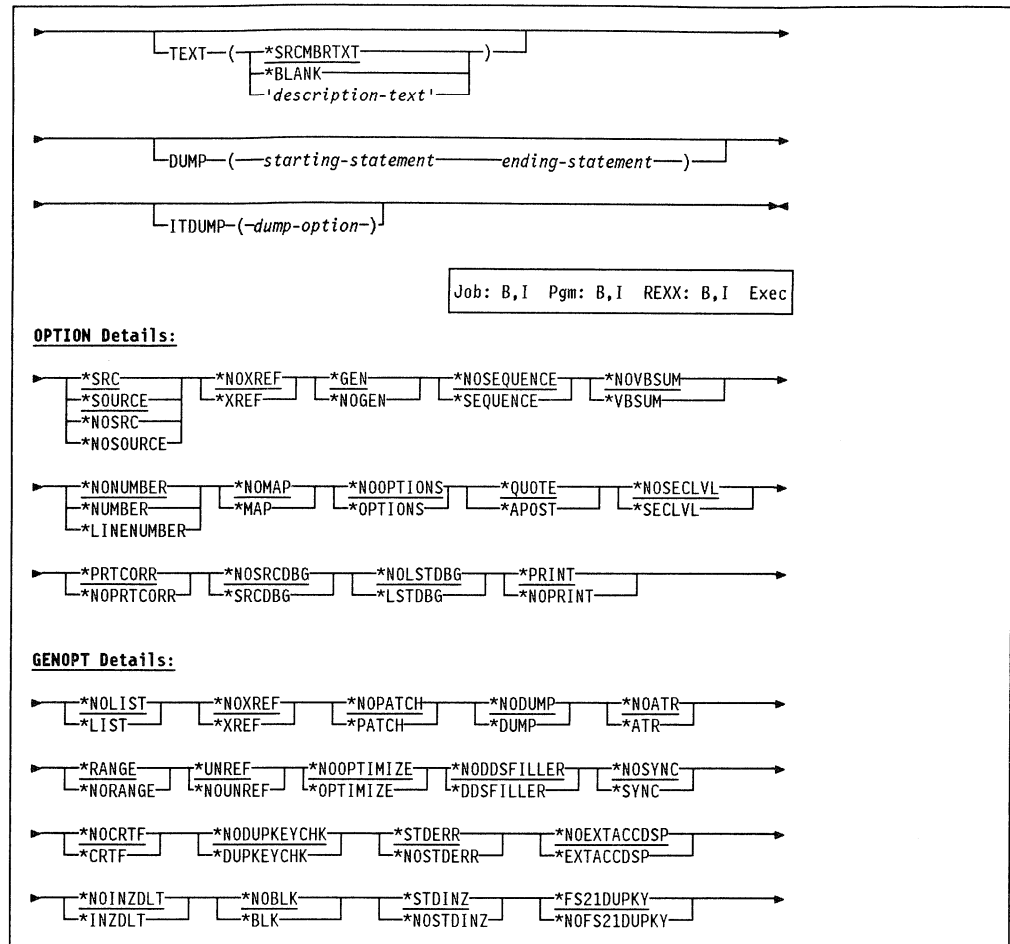


Figure 4 (Part 2 of 2). Syntax of the CRTCLPGM Command

Compiling Your Source Program For the Previous Release

You can compile a COBOL/400 program on an AS/400 system using the current release of the OS/400 operating system and restore it on an AS/400 system that uses a previous release of the operating system.

The Target Release (TGTRLS) parameter of the CRTCLPGM command allows you to specify the release level on which you intend to use the object program. The TGTRLS parameter has three possible values: *CURRENT, *PRV and *release-level*:

Specify *CURRENT if the object program is to be used on the release of the operating system currently running on your system. For example, if V2R2M0 is running on the system, *CURRENT means you intend to use the program on a system with V2R2M0 installed. This value is the default.

Specify *PRV if the object program is to be used on the previous release with modification level 0 of the operating system. For example, if V2R2M0 is running on your system, *PRV means you intend to use the program on a system with V2R1M0 installed.

release-level allows you to specify the release level on which you intend to use the object program. The values you can enter for this parameter depend

on the current version, release, and modification level, and they change with each new release.

For more information about the TGTRLS parameter, see page 26.

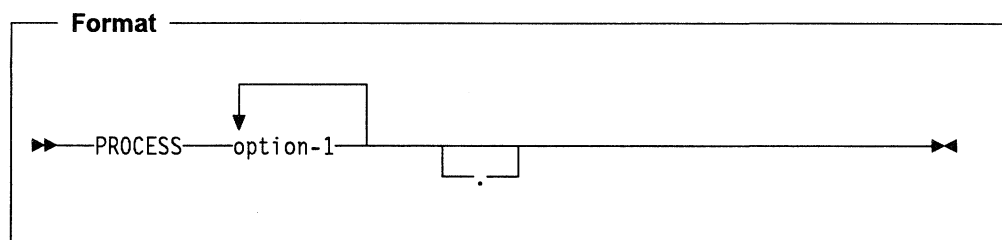
You should be aware of the following limitations:

- Support to compile for use with the previous release is only available when you use the TGTRLS parameter of the CRTCLPGM command. You must specify *PRV or the release level when you compile the program; you must then save the program, using the Save Object (SAVOBJ) or the Save Library (SAVLIB) CL command, in order to successfully restore it to the previous release of the operating system.
- You cannot use the TGTRLS parameter for COBOL programs created in the System/38 environment.
- You can restore an object program to the previous release or to a subsequent release. You cannot restore an object program on a system that is more than one release lower. That is, only one release of downward compatibility is provided.
- You cannot use functions that are new to the current release of the operating system in a program that you compile for use at the previous release level.
- Programs may be retranslated when they are restored to the previous release; therefore, you cannot delete observability if the programs are to be retranslated.
- No product library should be in the system portion of your library list.

Using the PROCESS Statement to Specify Compiler Options

The PROCESS statement is an optional part of the COBOL source program. You can use the PROCESS statement to specify options you would normally specify at compilation time. Options specified in the PROCESS statement override the corresponding options specified in the CRTCLPGM CL command.

The format of the PROCESS statement is as follows:



The following rules apply:

- The statement must be placed before the first source statement in the COBOL program immediately preceding the IDENTIFICATION DIVISION header.
- The statement begins with the word PROCESS. Options can appear on more than one line; however, only the first line can contain the word PROCESS.
- The word PROCESS and all options must appear within positions 8 through 72. Position 7 must be left blank. The remaining positions can be used as in

COBOL source statements: positions 1 through 6 for sequence numbers, positions 73 through 80 for identification purposes.

- The options must be separated by blanks and/or commas.
- Options can appear in any order. If conflicting options are specified, for example, LIST and NOLIST, the last option encountered takes precedence.
- If the option keyword is correct and the suboption is in error, the default suboption is assumed.

Not every CRTCLPGM parameter has a corresponding option in the PROCESS statement. Refer to the following tables which indicate the allowable PROCESS statement options and the equivalent CRTCLPGM command parameters and options. Defaults are underlined. Descriptions of the PROCESS statement options correspond to the parameter and option descriptions that start on page 18.

PROCESS Statement Option	CRTCLPGM
	GENLVL Parameter Option
GENLVL(nn)	nn

PROCESS Statement Options	CRTCLPGM
	OPTION Parameter Options
<u>GEN</u> NOGEN	* <u>GEN</u> *NOGEN
<u>NOMAP</u> MAP	* <u>NOMAP</u> *MAP
<u>NONUMBER</u> NUMBER LINENUMBER	* <u>NONUMBER</u> *NUMBER *LINENUMBER
<u>NOSECLVL</u> SECLVL	* <u>NOSECLVL</u> *SECLVL
<u>NOOPTIONS</u> OPTIONS	* <u>NOOPTIONS</u> *OPTIONS
<u>QUOTE</u> APOST	* <u>QUOTE</u> *APOST
<u>NOSEQUENCE</u> SEQUENCE	* <u>NOSEQUENCE</u> *SEQUENCE
<u>SOURCE</u> (or <u>SRC</u>) NOSOURCE (or NOSRC)	* <u>SOURCE</u> (or * <u>SRC</u>) *NOSOURCE (or *NOSRC)
<u>NOVBSUM</u> VBSUM	* <u>NOVBSUM</u> *VBSUM
<u>NOXREF</u> XREF	* <u>NOXREF</u> *XREF
<u>PRTCORR</u> NOPRTCORR	* <u>PRTCORR</u> *NOPRTCORR

PROCESS Statement Options	CRTCBLPGM
	GENOPT Parameter Options
<u>NOINZDLT</u> INZDLT	* <u>NOINZDLT</u> *INZDLT
<u>NOLIST</u> LIST	* <u>NOLIST</u> *LIST
<u>STDERR</u> NOSTDERR	* <u>STDERR</u> *NOSTDERR
<u>NODDSFILLER</u> DDSFILLER	* <u>NODDSFILLER</u> *DDSFILLER
<u>NOSYNC</u> SYNC	* <u>NOSYNC</u> *SYNC
<u>NOCRTF</u> CRTF	* <u>NOCRTF</u> *CRTF
<u>NODUPKEYCHK</u> DUPKEYCHK	* <u>NODUPKEYCHK</u> *DUPKEYCHK
<u>NOEXTACCDSP</u> EXTACCDSP	* <u>NOEXTACCDSP</u> *EXTACCDSP
<u>NOBLK</u> BLK	* <u>NOBLK</u> *BLK
<u>STDINZ</u> NOSTDINZ	* <u>STDINZ</u> *NOSTDINZ
<u>FS21DUPKEY</u> NOFS21DUPKEY	* <u>FS21DUPKEY</u> *NOFS21DUPKEY
<u>RANGE</u> NORANGE	* <u>RANGE</u> *NORANGE
<u>UNREF</u> NOUNREF	* <u>UNREF</u> *NOUNREF

PROCESS Statement Options	CRTCBLPGM
	CVTOPT Parameter Options
<u>NOVARCHAR</u> VARCHAR	* <u>NOVARCHAR</u> *VARCHAR
<u>NODATETIME</u> DATETIME	* <u>NODATETIME</u> *DATETIME
<u>NOCVTGRAPHIC</u> CVTGRAPHIC	* <u>NOGRAPHIC</u> *GRAPHIC

PROCESS Statement Options	CRTCBLPGM
	FLAGSTD Parameter Options
<u>NOFIPS</u> MINIMUM INTERMEDIATE HIGH	* <u>NOFIPS</u> *MINIMUM *INTERMEDIATE *HIGH
<u>NOSEG</u> SEG1 SEG2	* <u>NOSEG</u> *SEG1 *SEG2
<u>NODEB</u> DEB1 DEB2	* <u>NODEB</u> *DEB1 *DEB2
<u>NOOBSOLETE</u> OBSOLETE	* <u>NOOBSOLETE</u> *OBSOLETE

PROCESS Statement Options EXTDSPOPT(a b c)	CRTCBLPGM
	EXTDSPOPT Parameter Options
<u>DFRWRT</u> NODFRWRT	* <u>DFRWRT</u> *NODFRWRT
<u>UNDSPCHR</u> NOUNDSPCHR	* <u>UNDSPCHR</u> *NOUNDSPCHR
<u>ACCUPDALL</u> ACCUPDNE	* <u>ACCUPDALL</u> *ACCUPDNE

PROCESS Statement Options	CRTCBLPGM
	SA AFLAG Parameter Options
<u>NOSAAFLAG</u> SAAFLAG	* <u>NOFLAG</u> *FLAG

PROCESS Statement Option	CRTCBLPGM
	FLAG Parameter Option
FLAG(nn)	nn

PROCESS Statement Options	CRTCBLPGM
<u>NOFS9MTO0M</u> FS9MTO0M	not applicable
<u>NOGRAPHIC</u> GRAPHIC	not applicable

FS9MTO0M changes a file status of 9M to a file status of 0M.

The GRAPHIC option of the PROCESS statement is available for processing DBCS characters in DBCS literals. See Appendix F, "Supporting International Languages with Double-Byte Character Sets" on page 333 for information about DBCS support.

The EXTDSPOPT option on the PROCESS statement should be coded with the associated options in brackets similar to FLAG(nn) syntax. You can specify more than one option within the brackets for the EXTDSPOPT option. For example, to specify DFRWRT and UNDSPCHR, type

```
EXTDSPOPT(DFRWRT UNDSPCHR)
```

It is also valid to specify EXTDSPOPT or EXTDSPOPT().

When EXTDSPOPT alone is specified in the PROCESS statement, then all the default values for the additional options are in effect.

If you specify EXTDSPOPT(), it has no effect on your program.

If conflicting options are specified, the last option specified overrides the others.

Compiling Multiple Programs

The PROCESS statement can be used to separate multiple programs and/or sub-programs to be compiled with a single invocation of the compiler. (A **subprogram** is a called program that is combined with the calling program at run time to produce a run unit.) When compiling multiple programs, all compiler options specified on the CRTCLPGM command statement, plus all default options, plus the options specified on the last PROCESS statement preceding the program will be in effect for the compilation of that program. All compiler output is directed to the destinations specified by the CRTCLPGM command statement.

All object programs are stored in the library specified on the PGM parameter. If program-name is specified for the PGM parameter, the first program in the batch job has that name, and all other programs use the name specified in the PROGRAM-ID paragraph in the source program.

Using COPY within the PROCESS Statement

A COPY statement can be used in the source program wherever a character-string or separator can be used. Each COPY statement must be preceded by a space and followed by a period and a space. For more information on the COPY statement, refer to the "COPY Statement" section of the *COBOL/400* Reference*.

The Format 1 COPY statement can be used within the PROCESS statement to retrieve compiler options previously stored in a source library, and include them in the PROCESS statement. COPY can be used to include options that override those specified as defaults by the compiler. Any PROCESS statement options can be retrieved with the COPY statement.

Compiler options can both precede and follow the COPY statement within the PROCESS statement. The last encountered occurrence of an option overrides all preceding occurrences of that option.

The following example shows the use of the COPY statement within the PROCESS statement. The COPY statement must be followed by a period. Notice also that, in this example, NOMAP overrides the corresponding option in the library member:

000001	PROCESS XREF	MYPROG
000002	COPY DEFLTS.	MYPROG
	MAP, SOURCE, LIST	DEFLTS
000004	NOMAP, FLAG(20)	MYPROG
000010	IDENTIFICATION DIVISION.	MYPROG

Understanding Compiler Output

Compiler output can include:

- A summary of command options
- An options listing: a listing of options in effect for the compilation. Use OPTION(*OPTIONS).
- A source listing: a listing of the statements contained in the source program. Use OPTION(*SOURCE or *SRC).
- A verb usage listing: a listing of the COBOL verbs and the number of times each verb is used. Use OPTION(*VBSUM).
- A Data Division map: a glossary of compiler-generated information about the data. Use OPTION(*MAP). Also included is a mapping of user-supplied names to compiler-generated internal names.
- SAA flagging: a list of the functions in your program that are not portable to other SAA COBOL environments. Use SAAFLAG(*FLAG).
- FIPS messages: a list of messages for a FIPS COBOL subset, for any of the optional modules, for all of the obsolete language elements, or for a combination of a FIPS COBOL subset, optional modules and all obsolete elements. Refer to the information on the “FLAGSTD Parameter” on page 25 for the specific options available for FIPS flagging.
- A cross-reference listing. Use OPTION(*XREF).
- Compiler messages (including diagnostic statistics).
- Compilation statistics.
- A listing of the generated program in symbolic form.
- An object program.

The presence or absence of some of these types of compiler output is determined by options specified in the PROCESS statement or through the CRTCLPGM command. The level of diagnostic messages printed depends upon the FLAG option.

Specifying the Format of Your Listing

A slash (/) in the indicator area (column 7) of a line results in page ejection of the source program listing. The slash (/) comment line prints on the first line of the next page.

IBM Extension

If you specify the EJECT statement in your program, the next source statement prints at the top of the next page in the compiler listing. This statement may be

written anywhere in Area A or Area B and must be the only statement on the line.

The SKIP1/2/3 statement causes blank lines to be inserted in the compiler listing. A SKIP1/2/3 statement can be written anywhere in Area A or B. It must be the only statement on the line.

- SKIP1 inserts a single blank line (double spacing).
- SKIP2 inserts two blank lines (triple spacing).
- SKIP3 inserts three blank lines (quadruple spacing).

Each of the above SKIP statements causes a single insertion of one, two, or three lines.

A TITLE statement places a title on each indicated page.

You can selectively list or suppress your COBOL source statements by using the *CONTROL, *CBL, or COPY statements:

- *CONTROL NOSOURCE and *CBL NOSOURCE suppress the listing of source statements.
- *CONTROL SOURCE and *CBL SOURCE continue the listing of source statements.
- A COPY statement bearing the SUPPRESS phrase suppresses the listing of copied statements. For its duration, this statement overrides any *CONTROL or *CBL statement. If the copied member contains *CONTROL or *CBL statements, the last one runs once the COPY member has been processed.

Refer to the *COBOL/400* Reference* for additional information about the EJECT, SKIP1/2/3, *CONTROL, *CBL, COPY, and TITLE statements.

_____ End of IBM Extension _____

Time-Separation Characters

The TIMSEP parameter of job-related commands (such as CHGJOB) now specifies the time-separation character used in the time stamps that appear on compiler listings. In the absence of a TIMSEP value, the system value QTIMSEP is used by default.

Browsing Your Compiler Listing Using SEU

The Source Entry Utility (SEU) allows you to browse through a compiler listing in an output queue. You can review the results of a previous compilation while making the required changes to your source code. Figure 5 shows the split-display in SEU that allows you to browse through the listing from a work station.

```

Columns . . . . : 1 71          Edit          XMPLIB/QLBLSRC
SEU==>          XMPLE
FMT CB .....-A+++B+++++
0014.00      DATA DIVISION.
0015.00      FILE SECTION.
0016.00      FD FILE1
0017.00      RECORD CONTAINS 56 CHARACTERS
0018.00      LABEL RECORDS ARE OMITTED
0019.00      DATA RECORD IS REB-1.
0020.00      01 REC-1 PIC X(56).

Columns . . . . : 1 71          Browse          Spool file . . . : XMPLE
SEU==>
0000.50      STMT
0000.51 * 19 MSGID: LBL1327 SEVERITY: 30 SEQNBR: 001900
0000.52      Message . . . . : 'REB-1' not defined in the program. Clause
0000.53      ignored.
0000.54      * * * * * E N D   O F   M E S S A G E S   * *
0000.55      Message Summary
0000.56      Total      Info(0-4)      Warning(5-19)      Error(20-29)      Severe(30-39)

F6=Move split line  F19=Left  F20=Right
F21=System command  F24=More keys
Syntax error found.

```

Figure 5. SEU Split Edit/Browse Display

While browsing the compiler listing, you can scan for errors and correct those source statements that have errors. To scan for errors, type F *ERR on the SEU command line.

For complete information on browsing through a compiler listing, see the *SEU User's Guide and Reference*.

A Sample Program and Listing

The following pages illustrate the compiler options and source listing produced for the program example. References to the figures are made throughout the following text. These references are indexed by the reverse printing of letters on a black background, for example (**Z**). The reverse letters in the text correspond to the letters found in the figures.

Command Summary

This summary, which is produced as a result of compilation, lists all options specified in the CRTCLPGM command. Refer to "Using the Create COBOL Program (CRTCLPGM) Command" on page 15 for more information about user-defined options.

```
5738CB1 V2R2M0 001000          IBM SAA COBOL/400          TESTER/SAMPLE          AS400SYS 03/27/92 11:01:51  Page 1
Program . . . . . : SAMPLE
Library . . . . . : TESTER
Source file . . . . . : QLBSLRC
Library . . . . . : TESTER
Source member . . . . . : SAMPLE      03/27/92 11:01:34
Generation severity level . . . . . : 29
Text 'description' . . . . . : *BLANK
Source listing options . . . . . : *NONE
Generation options . . . . . : *NONE
Conversion options . . . . . : *NONE
Message limit:
  Number of messages . . . . . : *NOMAX
  Message limit severity . . . . . : 29
Print file . . . . . : QSYSPRT
Library . . . . . : *LIBL
FIPS flagging . . . . . : *NOFIPS *NOSEG *NODEB *NOOBSOLETE
SAA flagging . . . . . : *NOFLAG
Extended display options . . . . . :
Flagging severity . . . . . : 0
Replace program . . . . . : *YES
Target release . . . . . : *CURRENT
User profile . . . . . : *USER
Authority . . . . . : *LIBCRTAUT
Compiler . . . . . : IBM SAA COBOL/400
```

Figure 6. Command Summary Listing

Identifying the Compiler Options in Effect

The PROCESS statement, if specified, is printed first. Figure 7 is a list of all options in effect for the compilation of the program example: the options specified in the CRTCLPGM command, as modified by the PROCESS statement. Compiler options are listed at the beginning of all compiler output when the OPTIONS parameter is specified.

```
5738CB1 V2R2M0 001000          AS/400 COBOL Source          TESTER/SAMPLE          AS400SYS 03/27/92 11:01:51      Page 2
 1 000100 PROCESS OPTIONS, SAAFLAG, SOURCE, MAP, XREF,
 2 000200 FLAG(00), MINIMUM, VBSUM.
          CHG DATE
          03/27/92

          COBOL Compiler Options in Effect
          OPTIONS
          SOURCE
          XREF
          MAP
          VBSUM
          NONUMBER
          NOSEQUENCE
          GEN
          GENLVL(29)
          FLAG(0)
          MINIMUM
          NOSEG
          NODEB
          NOOBSOLETE
          SAAFLAG
          QUOTE
          NOSECLVL
          NOSRCDBG
          NOLSTDBG
          PRINT
          PRTCORR

          COBOL Generation Options in Effect
          NOLIST
          UNREF
          RANGE
          NOATR
          NOXREF
          NODUMP
          NOPATCH
          NOOPTIMIZE
          NODDSFILLER
          NOSYNC
          NOCRTF
          NODUPKEYCHK
          STDERR
          NOEXTACCDSP
          NOINZDLT
          NOFS9MTO0M
          NOBLK
          STDINZ
          FS21DUPKY

          COBOL Conversion Options in Effect
          NOVARCHAR
          NODATETIME
          NOGRAPHIC
```

Figure 7. List of Options in Effect

Source Listing

Figure 8 illustrates a source listing. The statements in the source program are listed exactly as submitted. The source is not listed if the NOSOURCE option is specified. After the page in which the PROGRAM-ID paragraph is listed, all compiler output pages have the program-id name listed in the heading before the system name.

```

5738CB1 V2R2M0 001000          AS/400 COBOL Source          TESTER/SAMPLE          AS400SYS 03/27/92 11:01:51    Page 4
STMT SEQNBR -A 1 B.+.....2.....3.....4.....5.....6.....7..IDENTFCN  S COPYNAME  CHG DATE
  A   B   C   D   E
3 000300 IDENTIFICATION DIVISION.
4 000400 PROGRAM-ID.          SAMPLE.
5 000500 AUTHOR.              PROGRAMMER NAME.
6 000600 INSTALLATION. COBOL DEVELOPMENT CENTRE.
7 000700 DATE-WRITTEN. 11/27/87.
8 000800 DATE-COMPILED. 03/27/92 11:01:51 .
9 000900 ENVIRONMENT DIVISION.
10 001000 CONFIGURATION SECTION.
11 001100 SOURCE-COMPUTER. IBM-AS400.
12 001200 OBJECT-COMPUTER. IBM-AS400.
13 001300 INPUT-OUTPUT SECTION.
14 001400 FILE-CONTROL.
15 001500     SELECT FILE-1 ASSIGN TO DISK-SAMPLE.
16 001600 DATA DIVISION.
17 001700 FILE SECTION.
18 001800 FD FILE-1
19 001900     LABEL RECORDS ARE STANDARD
20 002000     RECORD CONTAINS 20 CHARACTERS
21 002100     DATA RECORD IS RECORD-1.
22 002200 01 RECORD-1.
23 002300 02 FIELD-A          PIC X(20).
24 002400 WORKING-STORAGE SECTION.
25 002500 01 FILLER.
26 002600 05 KOUNT              PIC S9(2) COMP-3.
27 002700 05 LETTERS          PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
28 002800 05 ALPHA REDEFINES LETTERS
29 002900                      PIC X(1) OCCURS 26 TIMES.
30 003000 05 NUMBR            PIC S9(2) COMP-3.
31 003100 05 DEPENDENTS      PIC X(26) VALUE "01234012340123401234012340".
32 003200 05 DEPEND REDEFINES DEPENDENTS
33 003300                      PIC X(1) OCCURS 26 TIMES.
34 003400 COPY WRKRCD.
35 +000010 01 WORK-RECORD.          WRKRCD
36 +000020 05 NAME-FIELD          PIC X(1).          WRKRCD
37 +000030 05 FILLER              PIC X(1) VALUE SPACE. WRKRCD
38 +000040 05 RECORD-NO          PIC S9(3).          WRKRCD
39 +000050 05 FILLER              PIC X(1) VALUE SPACE. WRKRCD
40 +000060 05 LOCATION            PIC A(3) VALUE "NYC". WRKRCD
41 +000070 05 FILLER              PIC X(1) VALUE SPACE. WRKRCD
42 +000080 05 NO-OF-DEPENDENTS    WRKRCD
43 +000090                      PIC X(2).          WRKRCD
44 +000100 05 FILLER              PIC X(7) VALUE SPACES. WRKRCD
45 003500 77 WORKPTR USAGE POINTER.
003600*****
003700* THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO *
003800* BE CREATED AND INITIALIZES COUNTERS                *
003900*****
46 004000 PROCEDURE DIVISION.
004100 STEP-1.
47 004200     OPEN OUTPUT FILE-1.
48 004300     MOVE ZERO TO KOUNT, NUMBR.
004400*****
004500* THE FOLLOWING 3 PARAGRAPHS CREATE INTERNALLY THE *
004600* RECORDS TO BE CONTAINED IN THE FILE, WRITES THEM *
004700* ON THE DISK, AND DISPLAYS THEM                      *
004800*****
004900 STEP-2.
49 005000     ADD 1 TO KOUNT, NUMBR.
50 005100     MOVE ALPHA (KOUNT) TO NAME-FIELD.
51 005200     MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
52 005300     MOVE NUMBR          TO RECORD-NO.
005400 STEP-3.
53 005500     DISPLAY WORK-RECORD.
54 005600     WRITE RECORD-1 FROM WORK-RECORD.
005700 STEP-4.

```

Figure 8 (Part 1 of 2). An Example of a COBOL/400 Source Listing

```

55 005800    PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
005900*****
006000* THE FOLLOWING PARAGRAPH CLOSES FILE OPENED FOR *
006100* OUTPUT AND RE-OPENS IT FOR INPUT *
006200*****
006300 STEP-5.
56 006400    CLOSE FILE-1.
57 006500    OPEN INPUT FILE-1.
006600*****
006700* THE FOLLOWING PARAGRAPHS READS BACK THE FILE AND *
006800* SINGLES OUT EMPLOYEES WITH NO DEPENDENTS *
006900*****
007000 STEP-6.
58 007100    READ FILE-1 RECORD INTO WORK-RECORD
59 007200    AT END GO TO STEP-8.
007300 STEP-7.
60 007400    IF NO-OF-DEPENDENTS IS EQUAL TO "0"
61 007500    MOVE "Z" TO NO-OF-DEPENDENTS.
62 007600    GO TO STEP-6.
007700 STEP-8.
63 007800    CLOSE FILE-1.
64 007900    STOP RUN.
          * * * * *   E N D   O F   S O U R C E   * * * * *

```

Figure 8 (Part 2 of 2). An Example of a COBOL/400 Source Listing

Figure 8 displays the following fields:

- A** *Compiler-generated statement number:* The numbers appear to the left of the source program listing. These numbers are referenced in all compiler output listings except for FIPS messages listings. A statement number can span several lines, and a line can contain more than one statement.
 - B** *Reference number:* The numbers appear to the left of the source statements. The numbers that appear in this field and the column heading (shown as SEQNBR in this listing) are determined by an option specified in the CRTCLPGM command or in the PROCESS statement, as shown in the following table:
- | Option | Heading | Origin |
|------------|---------|-------------------------------------|
| NONUMBER | SEQNBR | Source-file sequence numbers |
| NUMBER | NUMBER | Standard COBOL sequence numbers |
| LINENUMBER | LINNBR | Compiler-generated sequence numbers |
- C** *Sequence error indicator column:* An S in this column indicates that the line is out of sequence. Sequence checking is performed on the reference number field only if the SEQUENCE option is specified.
 - D** *Copyname:* The copyname, as specified in the COBOL COPY statement, is listed here for all records included in the source program by that COPY statement. If the DDS-ALL-FORMATS phrase is used, the name <--ALL-FMTS appears under COPYNAME.
 - E** *Change/date field:* The date the line was last modified is listed here.

Verb Usage by Count Listing

Figure 9 shows the alphabetic list that is produced of all verbs used in the source program. A count of how many times each verb was used is also included. This listing is produced when the VBSUM option is specified.

```

5738CB1 V2R2M0 001000 AS/400 COBOL Verb Usage By Count TESTER/SAMPLE AS400SYS 03/27/92 11:01:51 Page 6
VERB COUNT
ADD 1
CLOSE 2
DISPLAY 1
GO 2
IF 1
MOVE 5
OPEN 2
PERFORM 1
READ 1
STOP 1
WRITE 1
***** END OF VERB USAGE BY COUNT *****

```

Figure 9. Verb Usage by Count Listing

Data Division Map

The Data Division map is listed when the MAP option is specified. It contains information about names in the COBOL source program. The number of bytes required for the File Section and Working-Storage Section is given at the end of the Data Division map.

```

5738CB1 V2R2M0 001000 AS/400 COBOL Data Division Map TESTER/SAMPLE AS400SYS 03/27/92 11:01:51 Page 7
STMT LVL SOURCE NAME SECTION DISP LENGTH TYPE I-NAME ATTRIBUTES
  F   G   H           I   J   K   L   M   N
 18 FD FILE-1          FS
22 01 RECORD-1        FS 00000000 20 GROUP .D00633C
23 02 FIELD-A         FS 00000000 20 AN .D0063AE
25 01 FILLER          WS 00000000 56 GROUP .D006420
26 02 KOUNT           WS 00000000 2 PACKED .D006490
27 02 LETTERS        WS 00000002 26 AN .D006512 VALUE
28 02 ALPHA           WS 00000002 1 AN .D0065B0 REDEFINES .D006512, DIMENSION(26)
30 02 NUMBR           WS 00000028 2 PACKED .D006632
31 02 DEPENDENTS     WS 00000030 26 AN .D0066B4 VALUE
32 02 DEPEND          WS 00000030 1 AN .D006754 REDEFINES .D0066B4, DIMENSION(26)
35 01 WORK-RECORD    WS 00000000 19 GROUP .D0067D6
36 02 NAME-FIELD     WS 00000000 1 AN .D00684C
37 02 FILLER          WS 00000001 1 AN .D0068C0 VALUE
38 02 RECORD-NO      WS 00000002 3 ZONED .D00693C
39 02 FILLER          WS 00000005 1 AN .D0069C2 VALUE
40 02 LOCATION       WS 00000006 3 A .D007A98 VALUE
41 02 FILLER          WS 00000009 1 AN .D007B20 VALUE
42 02 NO-OF-DEPENDENTS WS 00000010 2 AN .D007B9C VALUE
44 02 FILLER          WS 00000012 7 AN .D007C16 VALUE
45 77 WORKPTR        WS 00000000 16 POINTR .D007C92
FILE SECTION uses 20 bytes of storage
WORKING-STORAGE SECTION uses 75 bytes of storage
***** END OF DATA DIVISION MAP *****

```

Figure 10. Data Division Map

The Data Division map displays the following fields:

- F** **Statement number:** The compiler-generated statement number where the data item was defined is listed for each data item in the Data Division map.
- G** **Level of data item:** The level number of the data item, as specified in the source program, is listed here. Index-names are identified by an *IX* in the level-number and a blank type field.
- H** **Source name:** The data name, as specified in the source program, is listed here.
- I** **Section:** The section where the item was defined is shown here through the use of the following codes:

FS File Section
 WS Working-Storage Section
 LS Linkage Section
 SM Sort/Merge Section
 SR Special Register.

J *Displacement:* The offset, in bytes, of the item from the level-01 group item is given here.

K *Length:* The decimal length of the item is listed here.

L *Type:* The data class type for the item is shown here through the use of the following codes:

GROUP	Group Item
A	Alphabetic
AN	Alphanumeric
ANE	Alphanumeric edited
INDEX	Index data item (USAGE INDEX)
BOOLN	Boolean
ZONED	Zoned decimal (external decimal)
PACKED	Packed decimal (internal decimal) (USAGE COMP, COMP-3 or PACKED-DECIMAL)
BINARY	Binary (USAGE COMP-4 or BINARY)
NE	Numeric edited
POINTR	Pointer data item (USAGE POINTER)

M *Internal name:* The compiler-generated internal names are listed here and are assigned as follows:

File names

The internal name uses the form .Fnn, where .F indicates a file name, and nn is a unique two-digit number.

Data names

The internal name uses the form .Dxxxxxx, where .D indicates a data name or index name, and xxxxxx is a unique six-digit hex value. These names appear in the IRP listing if generated.

N *Attributes:* The attributes of the item are listed here as follows:

- For files, the following information can be given:

Device type
 ORGANIZATION
 ACCESS MODE
 BLOCK CONTAINS information
 RECORD CONTAINS information
 LABEL information
 RERUN is indicated
 SAME AREA is indicated
 CODE-SET is indicated
 SAME RECORD AREA is indicated
 LINAGE is indicated.

- For data items, the attributes indicate if the following information was specified for the item:

REDEFINES
 VALUE
 JUSTIFIED

SYNCHRONIZED
 BLANK WHEN ZERO
 SIGN IS LEADING
 SIGN IS LEADING SEPARATE
 SIGN IS SEPARATE
 INDICATORS.

- For table items, the dimensions for the item are listed here in the form DIMENSION (nn). For each dimension, a maximum OCCURS value is given. When a dimension is a variable, it is listed as such, giving the lowest and highest OCCURS values.

FIPS Messages

The FIPS messages, Figure 11, are listed when the FLAGSTD parameter is specified. See page 25 for more information about specifying the option for FIPS flagging. Only messages for the requested FIPS subset, optional modules and/or obsolete elements are listed.

Note: The sequence number and column number are given for each time the message is issued.

```

5738CB1 V2R2M0 001000 AS/400 COBOL FIPS Messages TESTER/SAMPLE AS400SYS 03/27/92 11:01:51 Page 8
FIPS-ID DESCRIPTION AND SEQUENCE NUMBERS FLAGGED P
0
LBL8200 Following nonconforming standard items valid only at FIPS intermediate level or higher.
LBL8201 COPY statement.
003400 0008
LBL8300 Following nonconforming standard items valid only at FIPS high level. Q
LBL8303 DATE-COMPILED paragraph.
000800 0010
LBL8500 Following nonconforming nonstandard items are IBM-defined or are IBM extensions. Q
LBL8504 Assignment-name in ASSIGN clause.
001500 0036
LBL8518 USAGE IS COMPUTATIONAL-3.
002600 0036
003000 0036
LBL8520 USAGE IS POINTER.
003500 0026
LBL8561 COPY statement with default library assumed.
003400 0019
7 FIPS violations flagged. R
***** END OF COBOL FIPS MESSAGES *****
  
```

Figure 11. FIPS Messages

The FIPS messages consist of the following fields:

- 0** *FIPS-ID:* This field lists the FIPS message number.
- P** *Description and reference numbers flagged:* This field lists a description of the condition flagged, followed by a list of the reference numbers from the source program where this condition is found.

The type of reference numbers used, and their names in the heading (shown as SEQUENCE NUMBERS in this listing) are determined by an option specified in the CRTCLPGM command or in the PROCESS statement, as shown in the following table:

Option	Heading
NONUMBER	DESCRIPTION AND SEQUENCE NUMBERS FLAGGED
NUMBER	DESCRIPTION AND USER-SUPPLIED NUMBERS FLAGGED
LINENUMBER	DESCRIPTION AND LINE NUMBERS FLAGGED

- Q** *Items grouped by level:* These headings subdivide the FIPS messages by level and category.
- R** *FIPS violations flagged:* The total number of FIPS violations flagged is included at the end of the FIPS listing.

SAA Messages

Figure 12 shows the SAA messages that are listed when you specify the SAA flagging option. See the SAAFLAG parameter on page 25 or “Using the PROCESS Statement to Specify Compiler Options” on page 31 for more information about specifying this option.

```

5738CB1 V2R2M0 001000          SAA COBOL Messages          TESTER/SAMPLE          AS400SYS 03/27/92 11:01:51  Page 9
MSGID  DESCRIPTION, SEQUENCE NUMBERS and COLUMN NUMBERS FLAGGED

LBL8800  The following items have been flagged as non-portable across other SAA COBOL systems.
LBL8801  Options APOST,NUMBER,SEQUENCE,GRAPHIC,NOCRTF,NODUPKEYCHK,NOSYNC and EXTACCDSP are not SAA COBOL.
        000100 0008
LBL8809  PROCESS statement.
        000100 0008
LBL8843  USAGE IS POINTER.
        003500 0026
        3 SAA COBOL Messages were flagged.
          * * * * *  E N D   O F   S A A   C O B O L   M E S S A G E S   * * * * *

```

Figure 12. SAA Messages

For more information about SAA flagging, see “SAA Flagging” on page 329.

Cross-Reference Listing

Figure 13 shows the cross-reference listing, which is produced when the XREF option is specified. It provides a list of all data references and procedure-name references, by statement number, within the source program.

```

5738CB1 V2R2M0 001000  AS/400 COBOL Cross Reference Listing  TESTER/SAMPLE          AS400SYS 03/27/92 11:01:51  Page 10
NAMES (* = Procedure-name)  DEFINED  REFERENCES (* = Changed)
S          T          U
ALPHA                28      50
DEPEND               32      51
DEPENDENTS           31      32
*DUMMY-SECTION       47
FIELD-A              23
FILE-1               18      15  47  56  57  58  63
KOUNT                26      48* 49* 50  51  55
LETTERS              27      28
LOCATION               40
NAME-FIELD           36      50*
NO-OF-DEPENDENTS    42      51* 60  61*
NUMBR                30      48* 49* 52
RECORD-NO            38      52*
RECORD-1             22      21  54*
*STEP-1              47
*STEP-2              49      55
*STEP-3              53      55
*STEP-4              55
*STEP-5              56
*STEP-6              58      62
*STEP-7              60
*STEP-8              63      59
WORK-RECORD          35      53  54  58*
WORKPTR              45
          * * * * *  E N D   O F   C R O S S   R E F E R E N C E   * * * * *

```

Figure 13. Cross-Reference Listing

The cross-reference listing displays the following fields:

- S** *Names field:* The data name or procedure name referenced is listed here. All procedure names are flagged with an * before the name. The names are listed alphabetically.
- T** *Defined field:* The statement number where the name was defined within the source program is listed here.
- U** *References field:* All statement numbers are listed in the same sequence as the name is referenced in the source program. An * following a statement number indicates that the item was modified in that statement.

Messages

Figure 14 shows the messages that are generated during program compilation.

```

5738CB1 V2R2M0 001000      AS/400 COBOL Messages      TESTER/SAMPLE      AS400SYS 03/27/92 11:01:51      Page 11
STMT
* V 18 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 001800 W
  Message . . . . : Blocking/Deblocking for file 'FILE-1' will
  be performed by compiler-generated code. Y
  ***** END OF MESSAGES *****
                                Message Summary
  Total      Info(0-4)  Warning(5-19)  Error(20-29)  Severe(30-39)  Terminal(40-99)
  Z 1          1          0                0                0                0
Source records read . . . . . : 79
Copy records read . . . . . : 10
Copy members processed . . . . : 1
Sequence errors . . . . . : 0
Highest severity message issued . . : 0
  LBL0901 00 Program SAMPLE created in library TESTER.
  ***** END OF COMPILATION *****

```

Figure 14. Diagnostic Messages

The fields displayed are:

- V** *Statement number:* This field lists the compiler-generated statement number associated with the statement in the source program for which the message was issued.¹
- W** *Reference number:* The reference number is issued here.¹ The numbers that appear in this field and the column heading (shown here as SEQNBR) are determined by an option specified in the CRTCLPGM command or in the PROCESS statement, as shown in the following table:

Option	Heading	Origin
NONUMBER	SEQNBR	Source-file sequence numbers
NUMBER	NUMBER	User-supplied sequence numbers
LINENUMBER	LINNBR	Compiler-generated sequence numbers

When a message is issued for a record from a copy file, the number is preceded by a +.

- X** *MSGID and Severity Level:* These fields contain the message number and its associated severity level. Severity levels are defined as follows:

¹ The statement number and the reference number do not appear on certain messages that reference missing items. For example, if the PROGRAM-ID paragraph is missing, message LBL0031 appears on the listing with no statement or reference number listed.

00 Informational
10 Warning
20 Error
30 Severe Error
40 Unrecoverable (usually a user error)
50 Unrecoverable (usually a compiler error)

Y *Message:* The message identifies the condition and indicates the action taken by the compiler.

Z *Message statistics:* This field lists the total number of messages and the number of messages by severity level.

The totals listed are the number of messages generated for each severity by the compiler and are not always the number listed. For example, if FLAG(10) is specified, no messages of severity less than 10 are listed. The counts, however, do indicate the number that would have been printed if they had not been suppressed.

Chapter 4. Running Your COBOL Program

This chapter provides the information you need to run your COBOL/400 program.

The most common ways to run a COBOL program are:

- Using a Control Language (CL) CALL command
- Using the COBOL CALL statement
- Using a menu-driven application program
- Issuing a user-created command.

You can use a CL CALL command interactively, as part of a batch job, or include it in a CL program. An example of a CL CALL command is CALL PAYROLL. PAYROLL is the name of a COBOL program that is called and run.

Any COBOL program can call another program with the COBOL CALL statement. (See the "CALL Statement" section of the *COBOL/400* Reference* for more information.)

Another way to run a COBOL program is from a menu-driven application. The work station user selects an option from a menu, calling the appropriate program. The following figure illustrates an example of an application menu.

PAYROLL DEPARTMENT MENU

1. Inquire into employee master
2. Change employee master
3. Add new employee
4. Return

Option: _____

Figure 15. Example of an Application Menu

The menu shown in this figure is normally displayed by a CL program in which each option calls a separate COBOL program.

You can also create a command yourself to run a COBOL program by using a command definition. A **command definition** is an object that contains the definition of a command (including the command name, parameter descriptions, and validity-checking information), and identifies the program that performs the function requested by the command. The system-recognized identifier for the object is *CMD.

For example, you can create a command, PAY, that calls a program, PAYROLL. PAYROLL is the name of a COBOL program that is called and run. You can enter the command interactively, or in a batch job. See the *CL Programmer's Guide* for further information about using the command definition.

When a COBOL program ends normally, the system returns control to the caller. The caller could be a work station user, a CL program (such as the menu-handling program), or another COBOL program.

If a COBOL program ends abnormally during run time, the escape message LBE9001

Error `message-id` caused program to end.

is issued. A CL program can monitor for this exception by using the Monitor Message (MONMSG) command. See the *CL Reference* for more information about control language commands.

If a program ends for any reason other than by the use of the STOP statement or by falling through to the end of the program, the return code is set to 2. See the RTVJOBA and DSPJOB commands in the *CL Programmer's Guide* for more information about return codes.

When you are running a batch job that uses the ACCEPT statement, the input data is taken from the job stream. This data must be placed immediately following the CL CALL for the COBOL program. It is your responsibility to request (through multiple ACCEPT statements) the same amount of data as is available. See the "ACCEPT Statement" section of the *COBOL/400* Reference* for more information.

Note: If more data is requested than is available, the CL command following the data is treated as input data. If more data is available than is requested, each extra line of data is treated as a CL command. In each instance, undesirable results can occur.

Replying to Run-Time Inquiry Messages

When you run a COBOL program, run-time inquiry messages may be generated. The messages require a response before the program continues running.

You can add the inquiry messages to a system reply list to provide automatic replies to the messages. The replies for these messages may be specified individually or generally. This method of replying to inquiry messages is especially suitable for batch programs, which would otherwise require an operator to issue replies.

You can add the following COBOL/400 inquiry messages to the system reply list:

LBE7200
LBE7201
LBE7203
LBE7204
LBE7205
LBE7206
LBE7207
LBE7208
LBE7209
LBE7210
LBE7211
LBE7604.

The reply list is only used when an inquiry message is sent by a job that has the Inquiry Message Reply (INQMSGRPY) attribute specified as INQMSGRPY(*SYSRPLY).

The INQMSGRPY parameter occurs on the following CL commands:

- Change Job (CHGJOB)
- Change Job Description (CHGJOBDD)

- Create Job Description (CRTJOB)
- Submit Job (SBMJOB).

You can select one of four reply modes by specifying one of the following values for the INQMSGRPY parameter:

- | | |
|---------|---|
| SAME | No change is made in the way that replies are sent to inquiry messages |
| RQD | All inquiry messages require a reply by the receiver of the inquiry messages |
| DFT | A default reply is issued |
| SYSRPLY | The system reply list is checked for a matching reply list entry. If a match occurs, the reply value in that entry is used. If no entry exists for that inquiry message, a reply is required. |

You can use the Add Reply List Entry (ADDRPYLE) command to add entries to the system reply list, or the Work with Reply List Entry (WRKRPLYE) command to change or remove entries in the system reply list. See the *CL Reference* for details of the ADDRPYLE and WRKRPLYE commands. You can also reply to runtime inquiry messages with a user-defined error-handler. For more information about error-handling APIs, refer to the *System Programmer's Interface Reference*.

Chapter 5. Debugging Your Program

The COBOL/400 language and the OS/400 operating system provide functions for debugging the programs you develop. This chapter describes those functions that allow you to debug your programs.

OS/400 Functions	COBOL/400 Functions
Breakpoints Traces	Debugging features Formatted dump

The OS/400 functions let you test programs while protecting your production files, and let you observe and debug operations as a program runs. No special source code is required for using the OS/400 functions.

The COBOL functions can be used independently of the OS/400 functions or in combination with them to:

- Debug a program
- Produce a formatted dump of the contents of fields, data structures, arrays, and tables.

Source code is required for using COBOL debugging features and formatted dump capability. A formatted dump can also be obtained by a user's response to a run-time message.

OPEN-FEEDBACK and I-O-FEEDBACK contents can provide additional debugging information. The method for obtaining this information is described later in this chapter in "File Status and Feedback Areas" on page 101.

While testing your programs, ensure that your library list is changed to direct the programs to a test library containing test data so that any existing real data is not affected.

To prevent database files in production libraries from being modified unintentionally, you can specify UPDPROD(*NO) on the Start Debug (STRDBG) command or by using the Change Debug (CHGDBG) command. See the *CL Reference* for more information.

Note: Refer to the *CL Programmer's Guide* for the CL commands required for testing and debugging programs.

No special statements for testing are contained in the program being tested. The program can be run normally without modification. All testing functions are specified in the job that contains the program, not in the actual program.

Testing functions apply only to the job in which they are specified. A program can be used concurrently in two jobs: one job that is in a test environment and another that is in a normal processing environment.

Testing functions allow you to observe the operations being performed while the program is running. These functions include using breakpoints and traces. (See "Using Breakpoints" on page 55 and "Using a Trace" on page 62 for more information.)

Avoiding Common Coding Errors

The errors made most frequently by COBOL programmers fall into two classes: compilation-time errors and run-time errors.

The compiler can detect errors when compiling your source program. While it makes corrections based on assumptions about certain errors it finds, you still need to correct the source and compile again if you have errors.

Common coding mistakes include:

- Unmatched record descriptions with externally described files
- Missing copy files
- Misspellings
- Faulty punctuation, especially missing periods
- Incorrect or incomplete syntax
- Misuse of reserved words.

The following errors appear only when you run your program:

- Failing to match the record description in your source program with the format of the actual records on the file to be read. This can either be an error by you (the records are correct, but your description is incorrect) or an error by the person who created the records your program reads. (For example, your description is correct, but one or more records were entered incorrectly.)
- Moving a data item whose subscript or index is too large, is negative, or is 0. Such a move could overlay and destroy part of your code or could fetch faulty data.
- Forgetting to define a sign field for items that can hold negative values. (In such a case, the sign is lost, and the negative number mistakenly becomes positive.)
- Moving data into an area too small for it, causing unwanted truncation.
- Forgetting to initialize the data items in the Working-Storage section before they are used. This may result in a decimal data error.
- In a called program, incorrectly matching the data descriptions in the Linkage Section with those of the caller. Or, in the calling program, incorrectly identifying the data to be passed.
- Moving a group item to another group item when the subordinate data descriptions are incompatible.
- Specifying USAGE for a redefined data item that is different from the USAGE originally specified for the redefined item, and then forgetting about the change once the redefinition takes place.
- Including a GO TO statement with no procedure name, and failing to initialize it with an ALTER statement before the running program reaches that point.
- Failing to include the AT END or INVALID KEY clauses or the USE procedures on files described in the program.
- Failing to match the TRANSACTION file source record description with the display format record description.

Using Breakpoints

A breakpoint is a statement number or a label in your program that stops program processing, and gives control to the display station user or to a specified program. If you use a statement number, it can be a statement number that appears on the compiler listing of the COBOL source program. If you use a label as a breakpoint, the label can be:

- Associated with a function performed by your COBOL program. For example,

```
.OPEN
```

indicates the open file function.

- An internal COBOL compiler generated label. For example,

```
.L000001
```

indicates the first internally generated label.

Note: To determine the internally generated labels for your program, use the GENOPT parameter on the CRTCLPGM command to get an IRP listing of the program.

When a breakpoint statement is about to be run for an interactive job, the system displays the breakpoint at which the program has stopped and, if requested, the values of program variables. After you get this information (in a display), you can go to a Command Entry display and then enter OS/400 commands to request other functions (such as displaying or changing a variable, adding a breakpoint, or adding a trace). See the *CL Programmer's Guide* for more information on breakpoint concepts.

For a batch job, a breakpoint program can be called when a breakpoint is reached. The breakpoint information is passed to the breakpoint program.

Example of a Program Using Breakpoints

Figure 16 shows an example of a COBOL program using breakpoints. The following OS/400 commands add breakpoints at statements 43 and 52. The value of variable KOUNT is displayed when the breakpoint at statement 52 is reached.

OS/400 Commands:

```
STRDBG    TESTPRT
ADDBKP    STMT(43)
ADDBKP    STMT(52)
           PGMVAR(KOUNT)
```

The OS/400 commands are explained in the *CL Reference*.

```

5738CB1 V2R2M0 001000 AS/400 COBOL Source TESTER/TESTPRT AS400SYS 03/30/92 17:05:37 Page 2
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7...IDENTFCN S COPYNAME CHG DATE
1 000100 IDENTIFICATION DIVISION.
2 000200 PROGRAM-ID. TESTPRT.
3 000300 AUTHOR. PROGRAMMER NAME.
4 000400 INSTALLATION. COBOL DEVELOPMENT CENTRE. 03/30/92
5 000500 DATE-WRITTEN. 11/27/87.
6 000600 DATE-COMPILED. 03/30/92 17:05:37 .
7 000700 ENVIRONMENT DIVISION.
8 000800 CONFIGURATION SECTION.
9 000900 SOURCE-COMPUTER. IBM-AS400. 03/30/92
10 001000 OBJECT-COMPUTER. IBM-AS400. 03/30/92
11 001100 INPUT-OUTPUT SECTION.
12 001200 FILE-CONTROL.
13 001300 SELECT FILE-1 ASSIGN TO DISK-SAMPLE.
14 001400 DATA DIVISION.
15 001500 FILE SECTION.
16 001600 FD FILE-1
17 001700 LABEL RECORDS ARE STANDARD
18 001800 RECORD CONTAINS 20 CHARACTERS
19 001900 DATA RECORD IS RECORD-1.
20 002000 01 RECORD-1.
21 002100 02 FIELD-A PIC X(20).
22 002200 WORKING-STORAGE SECTION.
23 002300 01 FILLER.
24 002400 05 KOUNT PIC S9(2) COMP-3.
25 002500 05 LETTERS PIC X(26) VALUE "ABCDEFGHIJKLMNORSTUVWXYZ".
26 002600 05 ALPHA REDEFINES LETTERS
27 002700 PIC X(1) OCCURS 26 TIMES.
28 002800 05 NUMBR PIC S9(2) COMP-3.
29 002900 05 DEPENDENTS PIC X(26) VALUE "01234012340123401234012340".
30 003000 05 DEPEND REDEFINES DEPENDENTS
31 003100 PIC X(1) OCCURS 26 TIMES.
32 003200 01 WORK-RECORD.
33 003300 05 NAME-FIELD PIC X(1).
34 003400 05 FILLER PIC X(1) VALUE SPACE.
35 003500 05 RECORD-NO PIC S9(3).
36 003600 05 FILLER PIC X(1) VALUE SPACE.
37 003700 05 LOCATION PIC A(3) VALUE "NYC".
38 003800 05 FILLER PIC X(1) VALUE SPACE.
39 003900 05 NO-OF-DEPENDENTS
40 004000 PIC X(2).
41 004100 05 FILLER PIC X(7) VALUE SPACES.
004200*****
004300* THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO *
004400* BE CREATED AND INITIALIZES COUNTERS *
004500*****
42 004600 PROCEDURE DIVISION.
004700 STEP-1.
43 004800 OPEN OUTPUT FILE-1. 1
44 004900 MOVE ZERO TO KOUNT, NUMBR.
005000*****
005100* THE FOLLOWING 3 PARAGRAPHS CREATE INTERNALLY THE *
005200* RECORDS TO BE CONTAINED IN THE FILE, WRITES THEM *
005300* ON THE DISK, AND DISPLAYS THEM *
005400*****
005500 STEP-2.
45 005600 ADD 1 TO KOUNT, NUMBR.
46 005700 MOVE ALPHA (KOUNT) TO NAME-FIELD.
47 005800 MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
48 005900 MOVE NUMBR TO RECORD-NO.
006000 STEP-3.
49 006100 DISPLAY WORK-RECORD.
50 006200 WRITE RECORD-1 FROM WORK-RECORD.
006300 STEP-4.
51 006400 PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
006500*****
006600* THE FOLLOWING PARAGRAPH CLOSES FILE OPENED FOR *
006700* OUTPUT AND RE-OPENS IT FOR INPUT *
006800*****
006900 STEP-5.
52 007000 CLOSE FILE-1. 2
53 007100 OPEN INPUT FILE-1.
007200*****
007300* THE FOLLOWING PARAGRAPHS READS BACK THE FILE AND *
007400* SINGLES OUT EMPLOYEES WITH NO DEPENDENTS *
007500*****

```

Figure 16 (Part 1 of 2). Example of a COBOL Program Using Breakpoints

```
5738CB1 V2R2M0 001000          AS/400 COBOL Source          TESTER/TESTPRT          AS400SYS 03/30/92 17:05:37          Page 2
STMT SEQNBR -A 1 B...+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S COPYNAME  CHG DATE
007600 STEP-6.
54 007700      READ FILE-1 RECORD INTO WORK-RECORD
55 007800      AT END GO TO STEP-8.
007900 STEP-7.
56 008000      IF NO-OF-DEPENDENTS IS EQUAL TO "0"
57 008100          MOVE "Z" TO NO-OF-DEPENDENTS.
58 008200      GO TO STEP-6.
008300 STEP-8.
59 008400      CLOSE FILE-1.
60 008500      STOP RUN.
          ***** END OF SOURCE *****
```

Figure 16 (Part 2 of 2). Example of a COBOL Program Using Breakpoints

1 The first breakpoint shows you where you are in the program. The following information is displayed when the break occurs:

```
Display Breakpoint
Statement/Instruction . . . . . : 43 /0017
Program . . . . . : TESTPRT
Recursion level . . . . . : 1

Press Enter to continue.
F3=Exit Program F10=Command entry
```

Figure 17. First Breakpoint Displayed

- 2** The following information is displayed as a result of reaching the second breakpoint:

```

                                Display Breakpoint
Statement/Instruction . . . . . : 52 /0056
Program . . . . . : TESTPRT
Recursion level . . . . . : 1
Start position . . . . . : 1
Format . . . . . : *CHAR
Length . . . . . : *DCL

Variable . . . . . : 05 KOUNT
  Type . . . . . : PACKED
  Length . . . . . : 2 0
  ' 26'

Press Enter to continue.

F3=Exit Program  F10=Command entry

```

Figure 18. Second Breakpoint Displayed

To specify a variable for the PGMVAR parameter, begin every name you enter with an alphanumeric character (A through Z, \$, #, or @). It can be followed by the characters (A through Z, 0 through 9, \$, #, @, or _).

The following example shows how to display a COBOL variable, RECORD-NO, in the program example. Because the hyphen is treated by the OS/400 operating system as a special character, RECORD-NO must be enclosed in quotation marks.

```

STRDBG      TESTPRT
ADDBKP      STMT(58)
            PGMVAR('RECORD-NO')

```

To display the value of a table element, enter the appropriate occurrence numbers (subscripts) with the variable name. Up to seven dimensions of subscripting are allowed, and the subscripts must be separated by commas.

Do not use an index-name or index data-item as a subscript. When an index is entered as a subscript, the operating system uses the internal value of the index as the subscript, and undesirable results can occur.

The following example shows how to specify the COBOL variable TABLE1 with three dimensions.

```
PGMVAR('TABLE1(SUB1, SUB2, SUB3)')
```

One or more blanks are allowed after each comma separating subscripts, but the total length of the variable plus subscripts, parentheses, commas, and blanks specified with the PGMVAR keyword cannot exceed 132 characters. For more information on how to code variables in CL commands, see the *CL Reference*.

Variable names can be qualified in the PGMVAR parameter. For example:

PGMVAR('NAME-FIELD OF WORK-RECORD')

Another technique can be used to display variables that are not elements of a multi-dimensional table. For example, to display the field NAME-FIELD, you can use the COBOL Data Division map to find its COBOL internal name (I-NAME). Next, use the IRP cross-reference listing to find the Object Definition Table (ODT) number for the internal-name. (See "Using the PROCESS Statement to Specify Compiler Options" on page 31 for information on how to obtain these listings.) Figure 19 shows the Data Division map, and Figure 20 on page 60 shows the cross-reference listing for the program example, TESTPRT.

STMT	LVL	SOURCE NAME	SECTION	DISP	LENGTH	TYPE	I-NAME	ATTRIBUTES
5738CB1	V2R2M0	001000	AS/400 COBOL Data Division Map		TESTER/TESTPRT		AS400SYS 03/30/92 17:05:37 Page 4	
16	FD	FILE-1	FS				.F01	DEVICE DISK, ORGANIZATION SEQUENTIAL, ACCESS SEQUENTIAL, RECORD CONTAINS 20 CHARACTERS, LABEL RECORDS STANDARD
20	01	RECORD-1	FS	00000000	20	GROUP	.D00633C	
21	02	FIELD-A	FS	00000000	20	AN	.D0063AE	
23	01	FILLER	WS	00000000	56	GROUP	.D006420	
24	02	KOUNT	WS	00000000	2	PACKED	.D006490	
25	02	LETTERS	WS	00000002	26	AN	.D006512	VALUE
26	02	ALPHA	WS	00000002	1	AN	.D0065B0	REDEFINES .D006512, DIMENSION(26)
28	02	NUMBR	WS	00000028	2	PACKED	.D006632	
29	02	DEPENDENTS	WS	00000030	26	AN	.D0066B4	VALUE
30	02	DEPEND	WS	00000030	1	AN	.D006754	REDEFINES .D0066B4, DIMENSION(26)
32	01	WORK-RECORD	WS	00000000	19	GROUP	.D0067D6	
33	02	NAME-FIELD	WS	00000000	1	AN	.D00684C	1
34	02	FILLER	WS	00000001	1	AN	.D0068C0	VALUE
35	02	RECORD-NO	WS	00000002	3	ZONED	.D00693C	
36	02	FILLER	WS	00000005	1	AN	.D0069C2	VALUE
37	02	LOCATION	WS	00000006	3	A	.D006A98	VALUE
38	02	FILLER	WS	00000009	1	AN	.D006B20	VALUE
39	02	NO-OF-DEPENDENTS	WS	00000010	2	AN	.D006B9C	
41	02	FILLER	WS	00000012	7	AN	.D006C16	VALUE
FILE SECTION uses 20 bytes of storage								
WORKING-STORAGE SECTION uses 75 bytes of storage								
***** END OF DATA DIVISION MAP *****								

Figure 19. Data Division Map for TESTPRT

1 The I-NAME for NAME-FIELD

```

5738SS1 V2R2M0 920925   IBM COBOL/400 5738CB1 V2R2M0   IRP LISTING FOR TESTPRT   03/30/92 17:05:37 Page 43
ODT  ODT Name
0184 .DMPFBH1 514*
0185 .DMPFBH2 515*
014F .DMPFBIB 452*
0148 .DMPFBLN 445*
015B .DMPFBLO 471*
0186 .DMPFBLP 512 516*
0182 .DMPFBLS 512*
014C .DMPFBL1 449* 1065 1066
014D .DMPFBL2 450*
0160 .DMPFBMF 476*
014E .DMPFBMN 451* 995 1098 1099 1118 1119
0180 .DMPFBND 509*
0150 .DMPFB0B 453*
015A .DMPFB0F 470*
0152 .DMPFB0L 458*
015F .DMPFBP0 475*
0161 .DMPFBQN 477*
0155 .DMPFBRC 461*
0153 .DMPFBRW 459*
0158 .DMPFBSC 468*
0149 .DMPFBSF 446*
014A .DMPFBSL 447*
014B .DMPFBSN 448*
0146 .DMPFBTY 443* 1097 1117
0159 .DMPFBUF 469*
0183 .DMPFBVL 513*
018B .DMPIOFB 522*
01A0 .DMPIOFS 545* 546 547
01A6 .DMPKYLN 551*
0165 .DMPNDEV 481* 1087 1145
0144 .DMPOFBS 441* 442 443 444 445 446 447 448 449 450 451 452 453 454 458 459 460 461 462 467 468 469 470 471 472 473 474 475 476
      477 478 479 480 481 482 508 509 510
01AA .DMPRCD 555*
01AC .DMPRCDN 557*
01AE .DMPRDUP 559*
01A1 .DMPRFMT 546*
01A7 .DMPRRN 552*
01A5 .DMP SRC 550*
0220 .D006A98 685*
0221 .D006B20 686*
0222 .D006B9C 687* 767 914 916
0223 .D006C16 688*
0211 .D0063AE 670*
0210 .D00633C 669* 789 904
0212 .D006420 671* 672 673 676 677
0213 .D006490 672* 753 757 761 765 815
0216 .D0065B0 675* 763
0214 .D006512 673* 674
0218 .D0066B4 677* 678
0217 .D006632 676* 754 758 769
021B .D0067D6 680* 681 682 683 684 685 686 687 688 778 789 904
021A .D006754 679* 767
021D .D0068C0 682*
021C .D00684C 681* 763 1
021F .D0069C2 684*
021E .D00693C 683* 769

```

Figure 20. Section of IRP Cross-Reference Listing for TESTPRT

1 021C is the ODT number for NAME-FIELD

Now you can use ODT number 021C (for NAME-FIELD), with the following commands, to add a breakpoint to the program example at statement 52.

```

STRDBG   TESTPRT
ADDBKP   STMT(52)
          PGMVAR(' /021C')

```

These commands are explained in the *CL Reference*.

The following is displayed when this breakpoint is reached:


```

                                Display Breakpoint
Statement/Instruction . . . . . : 52 /0056
Program . . . . . : TESTPRT
Recursion level . . . . . : 1
Start position . . . . . : 1
Format . . . . . : *CHAR
Length . . . . . : *DCL

Variable . . . . . : /021C
Type . . . . . : CHARACTER
Length . . . . . : 2
*...+....1....+....2....+....3....+....4....+....5
'Z'

Press Enter to continue.

F3=Exit Program  F10=Command entry

```

Figure 21. Breakpoint at Statement 52

Changing Program Variables

Now you can change the value of program variables to alter your program's processing. You can use the Change Program Variable (CHGPGMVAR) command to change the value of a variable. This procedure is explained in more detail in the *CL Reference*.

You can use the DSPPGMVAR command to display pointer data items, but you cannot use CHGPGMVAR to change pointer data items. To change pointer data items, you use the CHGHLLPTR or CHGPTR commands. For more information on the CHGHLLPTR and CHGPTR commands, refer to the *CL Reference*.

Considerations for Using Breakpoints

You should know the following breakpoint characteristics before using breakpoints:

- If a breakpoint is bypassed by, for example the GO TO statement, that breakpoint isn't processed.
- When a breakpoint is set on a statement, the breakpoint occurs before that statement is processed.
- Breakpoint functions are specified through OS/400 commands.

These functions include:

- Adding breakpoints to programs
- Removing breakpoints from programs
- Displaying breakpoint information
- Resuming the running of a program after a breakpoint has been reached (displayed).

See the *CL Programmer's Guide* for descriptions of these commands and for more details about breakpoints.

Using a Trace

A trace is a record of some or all of the statements run in a program. If requested, a trace also records the values of specific variables used in the program statements.

Program	Trace
Statement	Processing Order Variables
1	1 →
2	6 →
3	7 →
4	8 →
5	6 →
6	7 →
7	2 →
8	6 →
.	7 →
.	.
.	.

A trace differs from a breakpoint because the number of statements involved in the trace affects where the trace will end. The system records all the traced statements that were processed. You can request a display of the traced information, which shows the sequence in which the statements were processed and, if requested, the values of the variables used in the statements.

You specify which statements the system will trace. You can also specify that variables be displayed only when their value has changed since the last trace statement was run.

You can specify a trace of one statement in a program, a group of statements in a program, or all the statements in an entire program.

Example of Using a Trace

Figure 22 on page 63 shows a portion of a COBOL program example, TESTPRT. The following OS/400 command adds a trace of statements 54 through 58 in that program. The variable NO-OF-DEPENDENTS is to be recorded only if its value changes between statements 54 and 58:

```
ADDTRC      STMT(54 58)
            PGMVAR('NO-OF-DEPENDENTS')
            OUTVAR(*CHG)
```

Note: STRDBG must be entered before the ADDTRC statement.

```

573BCB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7...IDENTFCN S COPYNAME  CHG DATE
004200*****
004300* THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO *           03/07/09
004400* BE CREATED AND INITIALIZES COUNTERS *                       03/07/09
004500*****
42 004600 PROCEDURE DIVISION.                                       03/07/09
    004700 STEP-1.                                                  03/07/09
43 004800 OPEN OUTPUT FILE-1.                                       03/07/09
44 004900 MOVE ZERO TO KOUNT, NUMBR.                                03/07/09
    005000*****
    005100* THE FOLLOWING 3 PARAGRAPHS CREATE INTERNALLY THE *       03/07/09
    005200* RECORDS TO BE CONTAINED IN THE FILE, WRITE THEM *       03/07/09
    005300* ON THE DISK, AND DISPLAY THEM *                          03/07/09
    005400*****
    005500 STEP-2.                                                  03/07/09
45 005600 ADD 1 TO KOUNT, NUMBR.                                     03/07/09
46 005700 MOVE ALPHA (KOUNT) TO NAME-FIELD.                       03/07/09
47 005800 MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.                03/07/09
48 005900 MOVE NUMBR TO RECORD-NO.                                 03/07/09
    006000 STEP-3.                                                  03/07/09
49 006100 DISPLAY WORK-RECORD.                                       03/07/09
50 006200 WRITE RECORD-1 FROM WORK-RECORD.                         03/07/09
    006300 STEP-4.                                                  03/07/09
51 006400 PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
    006500*****
    006600* THE FOLLOWING PARAGRAPH CLOSES FILE OPENED FOR *       03/07/09
    006700* OUTPUT AND RE-OPENS IT FOR INPUT *                       03/07/09
    006800*****
    006900 STEP-5.                                                  03/07/09
52 007000 CLOSE FILE-1.                                             03/07/09
53 007100 OPEN INPUT FILE-1.                                         03/07/09
    007200*****
    007300* THE FOLLOWING PARAGRAPHS READ BACK THE FILE AND *       03/07/09
    007400* SINGLE OUT EMPLOYEES WITH NO DEPENDENTS *               03/07/09
    007500*****
    007600 STEP-6.                                                  03/07/09
54 007700 READ FILE-1 RECORD INTO WORK-RECORD
55 007800 AT END GO TO STEP-8.
    007900 STEP-7.
56 008000 IF NO-OF-DEPENDENTS IS EQUAL TO "8"
57 008100 MOVE "Z" TO NO-OF-DEPENDENTS.
58 008200 GO TO STEP-6.
    008300 STEP-8.
59 008400 CLOSE FILE-1.
60 008500 STOP RUN.
***** END OF SOURCE *****

```

Figure 22. Example of a COBOL Program Using a Trace

Figure 23 on page 64 is an example of a listing of the traced information. This information is produced by the Display Trace Data (DSPTRCDTA) command:

```
DSPTRCDTA OUTPUT(*PRINT) CLEAR(*YES)
```

This command is explained in the *CL Reference*.

```

5738SS1 V2R2M0                               Display Trace Data
Job . . . . : DSP02      User . . . . : PGMR5      Number . . . . : 004122
Program      Statement/
TESTPRT     Instruction      Recursion Level      Sequence Number
Start position . . . . . : 1
Length . . . . . : *DCL
Format . . . . . : *CHAR
*Variable . . . . . : 05 NO-OF-DEPENDENTS
Type . . . . . : CHARACTER
Length . . . . . : 2
*...+...1...+...2...+...3...+...4...+...5
'0 '

Program      Statement/
TESTPRT     Instruction      Recursion Level      Sequence Number
TESTPRT     56                1                    2
TESTPRT     57                1                    3
TESTPRT     58                1                    4
Start position . . . . . : 1
Length . . . . . : *DCL
Format . . . . . : *CHAR
*Variable . . . . . : 05 NO-OF-DEPENDENTS
Type . . . . . : CHARACTER
Length . . . . . : 2
*...+...1...+...2...+...3...+...4...+...5
'Z '

Program      Statement/
TESTPRT     Instruction      Recursion Level      Sequence Number
TESTPRT     54                1                    5
TESTPRT     56                1                    6
Start position . . . . . : 1
Length . . . . . : *DCL
Format . . . . . : *CHAR
*Variable . . . . . : 05 NO-OF-DEPENDENTS
Type . . . . . : CHARACTER
Length . . . . . : 2
*...+...1...+...2...+...3...+...4...+...5
'1 '

Program      Statement/
TESTPRT     Instruction      Recursion Level      Sequence Number
TESTPRT     58                1                    7
TESTPRT     54                1                    8
TESTPRT     56                1                    9
Start position . . . . . : 1
Length . . . . . : *DCL
Format . . . . . : *CHAR
*Variable . . . . . : 05 NO-OF-DEPENDENTS
Type . . . . . : CHARACTER
Length . . . . . : 2
*...+...1...+...2...+...3...+...4...+...5
'2 '

Program      Statement/
TESTPRT     Instruction      Recursion Level      Sequence Number
TESTPRT     58                1                    10
TESTPRT     54                1                    11
TESTPRT     56                1                    12

```

Figure 23. Trace Data Display Listing

Considerations for Using a Trace

You should understand the following trace characteristics before using them:

- Statements bypassed by, for example the GO TO statement, are not included in the trace.
- Trace functions are specified through OS/400 commands in the job containing the traced program. These functions include adding trace requests to a program, removing trace requests from a program, removing data collected from previous traces, displaying trace information, and displaying the traces that have been specified for a program.
- In addition to statement numbers, names of COBOL-generated routines can appear on the trace output STMT field.

See the *CL Programmer's Guide* for more information on traces.

Using a Debug Run-Time Switch

A run-time switch is provided for the COBOL Debug facility. This switch activates the debugging code generated when WITH DEBUGGING MODE is specified. When the switch is set on, all compiled debugging sections are activated; when it is set off (the default), the USE FOR DEBUGGING Declarative procedures are deactivated. Refer to Appendix B, "Debugging Features" on page 309 for more information on COBOL debugging features and the use of the run-time switch.

Using a COBOL Formatted Dump

Some COBOL run-time messages allow you to obtain a COBOL formatted dump option by selecting either D or F. The formatted dump (choose D) includes current information about the files in your program, contents of fields, data structures, arrays, and tables for user-defined COBOL data variables.

If you choose the F option, the dump also includes a list of compiler-generated fields and their contents.

Both the D option and the F option will dump the first 256 characters of program variables. Any variable greater than 256 characters will be truncated.

If you do not want a dump, specify C (cancel with *no* dump). Reply C is also the default reply for all COBOL inquiry messages that allow a dump.

For more information about reply modes see "Replying to Run-Time Inquiry Messages" on page 50.

The output for the dump is sent to the IBM-supplied printer file QPPGMDMP.

To see an example of a formatted dump, refer to Appendix H, "Example of a COBOL Formatted Dump" on page 367.

Chapter 6. COBOL/400 Exception and Error Handling

This chapter describes COBOL/400 error handling and its use. It also explains the relationship between error handling and the processing of I/O verbs.

The COBOL/400 compiler provides two error-handling methods: standard and nonstandard. Standard error handling is not available on compilers released earlier than Version 1 Release 3.

Standard Error Handling

Standard error handling gives you extra compatibility with other IBM COBOL compilers (such as VS COBOL II) as well as non-IBM COBOL compilers. It can help you during the processing of I/O statements by catching severe errors that might not otherwise be noticed.

An important characteristic of standard error handling is the issuing of a run-time message when an error occurs during the processing of an I/O statement if there is no AT END/INVALID KEY phrase in the I/O statement, USE procedure for the file, or FILE STATUS clause in the SELECT statement for the file.

Release Sensitivity!

Standard error handling was introduced in Version 1 Release 3 as a *default* option. To get the error handling that was used in earlier releases, specify *NOSTDERR as a generation option of the CRTCLPGM command, or NOSTDERR in the PROCESS statement.

Error Handling Overview

When you run a COBOL program, several types of errors can occur. The COBOL statement active at the time of a given error causes certain COBOL clauses or phrases to run.

During arithmetic operations, typical errors are size (MCH1210) errors and decimal data (MCH1202) errors; the corresponding error-handling phrase is the SIZE ERROR phrase.

Most MCH errors are not directly detected by COBOL; they are detected by the operating system and result in system messages. COBOL then monitors for these messages, setting internal bits that determine whether to run a SIZE ERROR imperative statement or issue a run-time message (LBE7200) to end the program.

COBOL does detect errors that result from division by zero during an arithmetic operation. If detected by COBOL, these errors cause the SIZE ERROR imperative statement to run.

System message MCH1210 occurs when you move a numeric field to a receiver that is too small. This error is monitored by COBOL, and also results in the running of the SIZE ERROR imperative statement.

LBE7200 is a run-time message that is usually issued when an unmonitored severe error occurs in your COBOL program. Under *NOSTDERR, it can also be issued when an error occurs in the absence of an appropriate error handler.

System message MCH1202 is a typical example of an unmonitored severe error. This kind of error results in the COBOL run-time message LBE7200 (or LBE7204 if the error occurs in a program called by a COBOL program). System messages MCH3601 and MCH0601 are other examples of unmonitored severe errors.

For I/O operations, there are several important error handling phrases and clauses. These are the AT END/INVALID KEY and NO DATA phrases (coded at the COBOL statement level), the USE procedure, and the FILE STATUS clause (coded at the file level). During arithmetic and I/O operations, errors are detected by the system, which sends messages; the messages are then monitored by COBOL. Similar to the case of an error that results from division by zero, COBOL does detect some errors during an I/O operation. Regardless of how an error is detected during an I/O operation, the result will always be an internal file status of other than zero, a run-time message, or both.

General-Use Programming Interface

Using Error-Handling Application Programming Interfaces (APIs)

You can use COBOL/400 APIs to control error handling for you within your programs. These APIs are Retrieve COBOL Error Handler (QLRRTVCE), and Set COBOL Error Handler (QLRSETCE).

The Retrieve COBOL Error Handler (QLRRTVCE) API allows you to retrieve the name of the current or pending COBOL error-handling program. You can call it from any programming language.

The Set COBOL Error Handler (QLRSETCE) API allows you to specify the identity of a COBOL error-handling program. You can call it from any programming language.

You can also use the Change COBOL Main Program (QLRCHGCM) API to create multiple run units, each with its own error handler.

For detailed information on all of these APIs, refer to the *System Programmer's Interface Reference*.

End of General-Use Programming Interface

Internal and External File Status

You must provide a FILE-CONTROL entry to specify the organization and access method for each file used by your COBOL program. You can also code a FILE STATUS clause in this entry.

The FILE STATUS clause designates one or two data items (coded in the WORKING-STORAGE section) to hold a copy of the result of an I/O operation. Your copy of the first of these items is called the external file status. If you use a TRANSACTION file, you have a further record of the result called the external return code, which consists of the external major and minor return codes.

COBOL keeps its own copies of these two data items, both of which are stored in the COBOL File Information Block (FIB). In this chapter, *file status* and *(major/minor) return code* refer to COBOL's copies unless otherwise specified.

During the processing of an I/O statement, the file status can be updated in one of three ways, as described below. The contents of the file status determine which error handling procedures to run.

Error handling procedures take control after an unsuccessful input or output operation, which is denoted by a file status of other than zero. Before any of these procedures run, the file status is copied into the external file status.

The file status is set in one of three ways:

- Method A (all files):

COBOL checks the contents of variables in file control blocks. If the contents are not what is expected, a file status of other than zero is set. Most file statuses set in this way result from checking the COBOL File Information Block (FIB) and the system User File Control Block (UFCB).

- Method B (transaction files):

COBOL checks the major and minor return codes from the system. If the major return code is not zero, the return code (consisting of major and minor return codes) is translated into a file status. If the major return code is zero, the file status may have been set by Method A or C.

Note that for subfile READ, WRITE, and REWRITE operations, only Methods A and C apply.

For a list of return codes and their corresponding file statuses, see "File Structure Support Summary and Status Key Values" in the *COBOL/400* Reference*.

- Method C (all files):

A message is sent by the system when COBOL calls on data management to perform an I/O operation. COBOL then monitors for these messages and sets a file status accordingly.

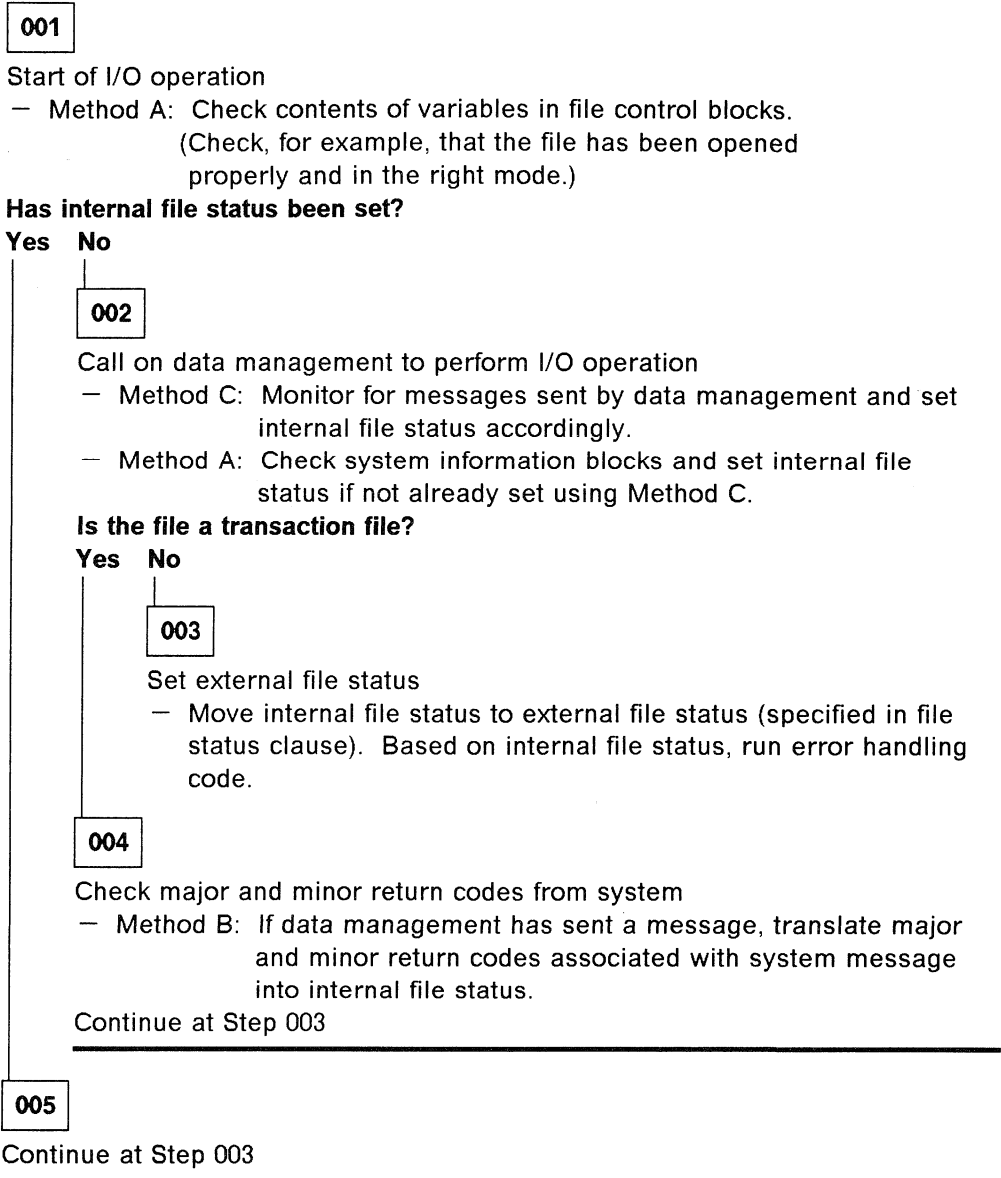
COBOL specifically monitors for a message by generating message monitors in the program object produced at compilation time. Message monitor generation is based on the types of files (organization type and access type are examples) that you specify in a program. Thus, a message that is specifically monitored for in one program may fall under the generic I/O handler in another. More information about message monitor generation will follow in this chapter.

COBOL monitors for most messages sent by the system in response to an I/O operation. Typical I/O exceptions result in CPF messages that begin with "CPF4" or "CPF5," and COBOL does specific monitoring for these.

For a list of messages for which COBOL does specific monitoring, see "File Structure Support Summary and Status Key Values" in the *COBOL/400* Reference*.

General Error Detection

How File Status is Set



Message Monitor Generation

A message monitor provides a way for a program to handle messages sent by the system or by another program. A message monitor can handle one or more messages.

In some respects, a message monitor resembles a USE procedure. Similar to the way in which a USE procedure specifies actions to take in response to an I/O error, a message monitor specifies an action to take when an error occurs during the processing of a machine interface (MI) instruction. Note that an MI instructional error is signalled by a system message, and note that each COBOL statement is composed of one or more MI instructions.

Unlike a USE procedure (which may not be active during an entire program), a COBOL message monitor becomes active as soon as the program starts. Message monitors set file statuses and indicate SIZE ERROR, END-OF-PAGE, and OVERFLOW conditions.

Message monitors generated by COBOL are grouped into several sets, generated under certain conditions within a COBOL program. The following table provides general guidelines regarding the generation of message monitors:

Cause of Message Monitor	Sample Members of Monitored Message Set
You code a file status clause	<ul style="list-style-type: none"> • File not found, external file status 35 • Permanent error condition, external file status 30 • OPEN mode not valid, external file status 37 • No next record, system message CPF5183 (part of external file status 46) • Undefined or unauthorized access type, external file status 91 • Logic error, external file status 92 (except for system messages CPF4740 and CPF5070) • Record is locked, external file status 9D • OPEN with commitment control failed, external file status 9P • WRITE not valid, system messages CPF5018 and CPF5272 (part of external file status 24).
You code an AT END phrase	<ul style="list-style-type: none"> • End-of-file handler, system messages CPF5001 and CPF5025 • File not found, external file status 35.
You specify a subfile in your program	<ul style="list-style-type: none"> • Last record written to subfile, external file status 9M or 0M • Subfile record not found, system message CPF5020 (part of external file status 23) • Subfile boundary violation, system messages CPF5021 and CPF5043 (part of external file status 24). A boundary violation is an attempt to write beyond the externally defined boundaries of a sequential file.
You code a subfile READ statement with the NEXT MODIFIED phrase	<ul style="list-style-type: none"> • No modified subfile record, external file status 12.

<i>Table 1 (Page 2 of 2). Generation of Message Monitors</i>	
Cause of Message Monitor	Sample Members of Monitored Message Set
You use an indexed sequential file	<ul style="list-style-type: none"> No specific monitor (Method A), set internal file statuses 21 and 22.
There is a keyed READ operation	<ul style="list-style-type: none"> System messages CPF5006 and CPF5013 (part of external file status 23).
There is a sequential WRITE operation	<ul style="list-style-type: none"> Boundary violation, system message CPF5116 (part of external file status 34).
There is an indexed sequential REWRITE operation	<ul style="list-style-type: none"> No specific monitor (Method A), set internal file statuses 21, 43, 44, and 95.
There is TRANSACTION I/O	<ul style="list-style-type: none"> READ timeout, system message CPF4743, set internal file status 00 No data during READ, system message CPF4742, set NO DATA bit No acquired devices, system message CPF5070 (part of external file status 92) No devices invited/acquired, system message CPF4740 (part of external file status 92 and external file status 10) Cancel job, external file status 9A WRITE failed, external file status 9I Temporary error, external file status 9N.
You specify a format clause in an I/O statement	<ul style="list-style-type: none"> Format name not valid/not found, internal file status 9K.
There is any I/O at all (including extended ACCEPT/DISPLAY operations) in your program.	<ul style="list-style-type: none"> END-OF-PAGE exception handler (system message CPF5004) Level check error, external file status 39 Generic exception handler, external file status 90 Indicator mismatch (run-time message LBE7421, system message CPF4238) Ignore COMMIT or ROLLBACK (system message CPF8350). Duplicate key, external file status 22. READ DYNAMIC invalid change of direction, internal file status 9U, system message CPF5184.
<p>Note: For a list of monitored messages that fall under a particular external file status, see "File Structure Support Summary and Status Key Values" in the <i>COBOL/400* Reference</i>.</p>	

Ending of a COBOL Program

There are three things that can cause a COBOL program to end:

A COBOL statement (EXIT PROGRAM, STOP RUN, or GOBACK)

A reply to an inquiry message

An implicit STOP RUN or EXIT PROGRAM statement.

A STOP RUN statement is implied when a main COBOL program has no next executable statement (implicit EXIT PROGRAM for a COBOL subprogram), that is, when processing falls through the last statement of a program.

Inquiry messages can be issued in response to a COBOL statement (namely a STOP literal), but they are usually issued when a severe error occurs in a program, or when a COBOL operation does not complete successfully. (Examples are LBE7205, LBE7207, and LBE7208.)

There are four common replies to a COBOL inquiry message: C, D, F, and G (cancel, cancel and dump, cancel and full dump, continue). The first three cause (as their final steps) an implicit STOP RUN followed by escape message LBE9001. LBE9001 indicates that the program is ending because of a message.

An implicit or explicit STOP RUN statement, or a GOBACK statement that appears in a main program, ends the entire COBOL run unit. If an escape message (LBE9001) is issued as the final step of a run unit, the caller of the first COBOL program can monitor for it. (This is because the first COBOL program to be called becomes the main program.)

If a COBOL run unit consists of several COBOL and non-COBOL programs, it is the main COBOL program that can issue the escape message. Thus, any non-COBOL program that is called after the main program cannot monitor for the escape message.

Return Codes

When you specify a TRANSACTION file in your program, the FILE STATUS clause of your SELECT statement can contain two data names: the external file status, and the (major and minor) return code. As described under "Internal and External File Status" on page 68, a file status can be set in one of three ways; however, return codes are set by the system after any transaction I/O that calls data management. Consequently, most error conditions that result in a system message also have an associated return code.

Return codes are similar to file status values. That is, CPF messages sent by the system are grouped together under message monitors, and each message monitor sets one or more file statuses.

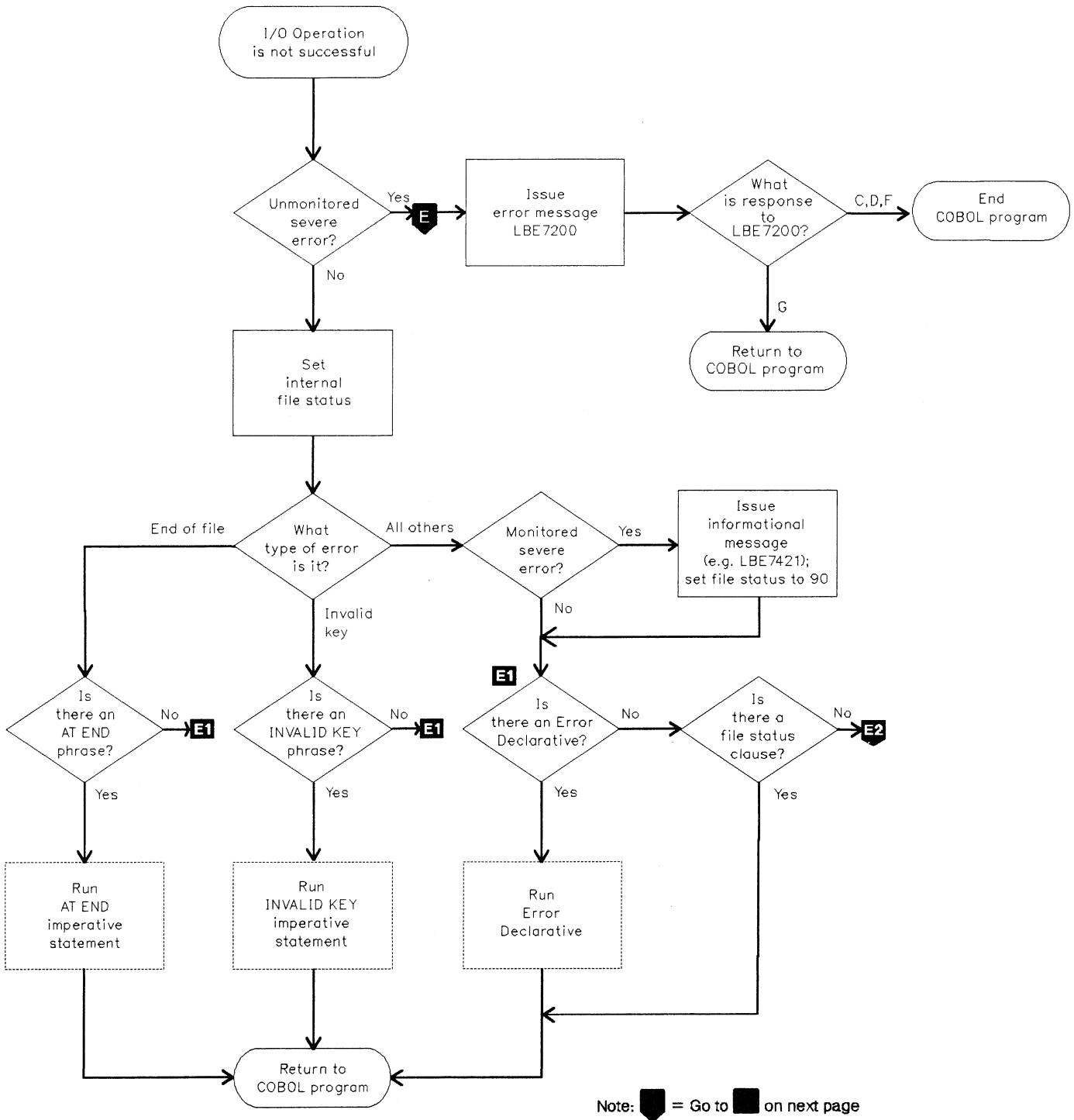
Similarly, CPF messages are grouped together, and each group of messages generates the same major return code. (The minor return code is not necessarily the same.)

The main difference between file statuses and return codes is that the grouping of CPF messages is different.

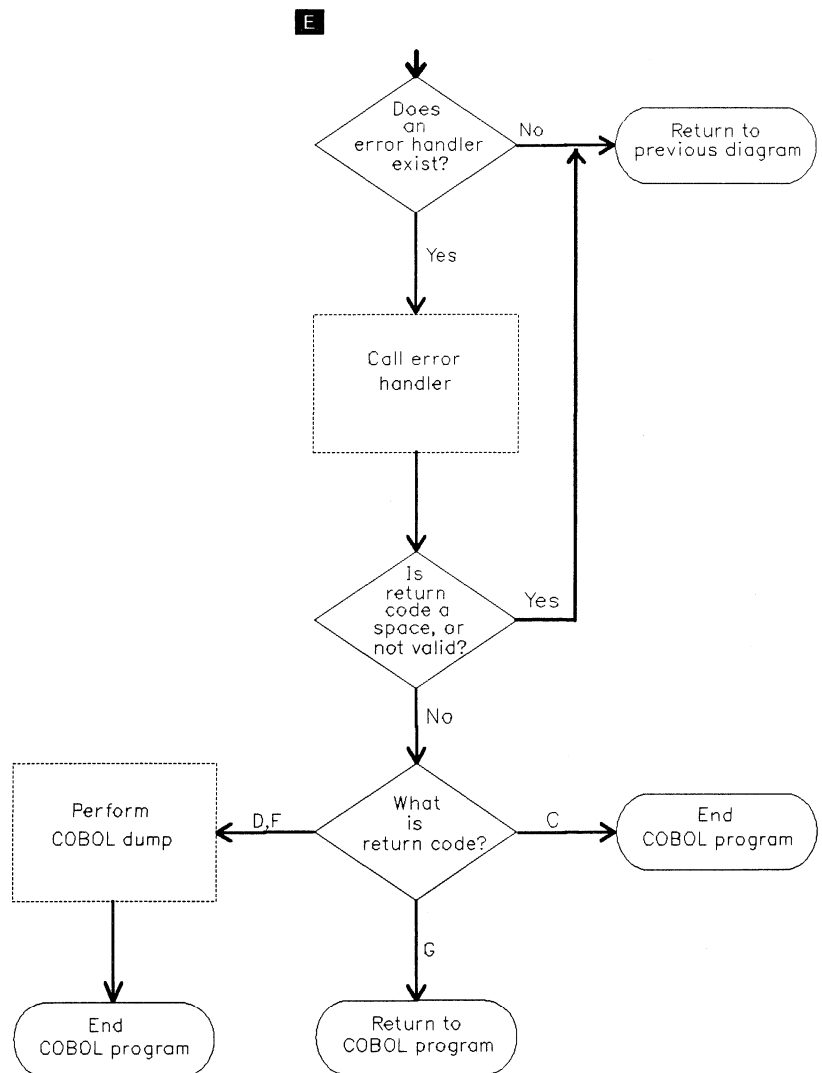
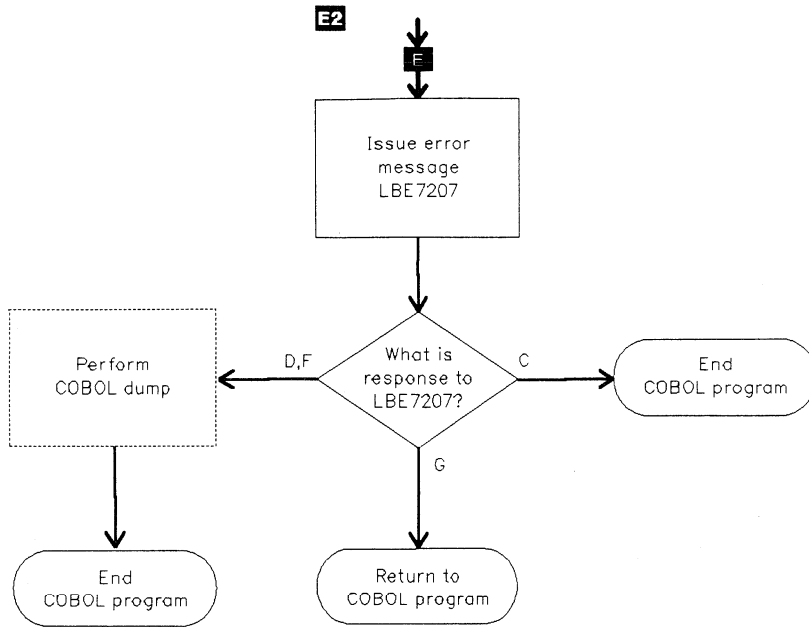
Although COBOL only sets return codes for TRANSACTION files, other types of files (such as printer files) also set return codes. You can access the return codes for these files through an ACCEPT from I-O-FEEDBACK operation.

Standard and Nonstandard Error Handling Models

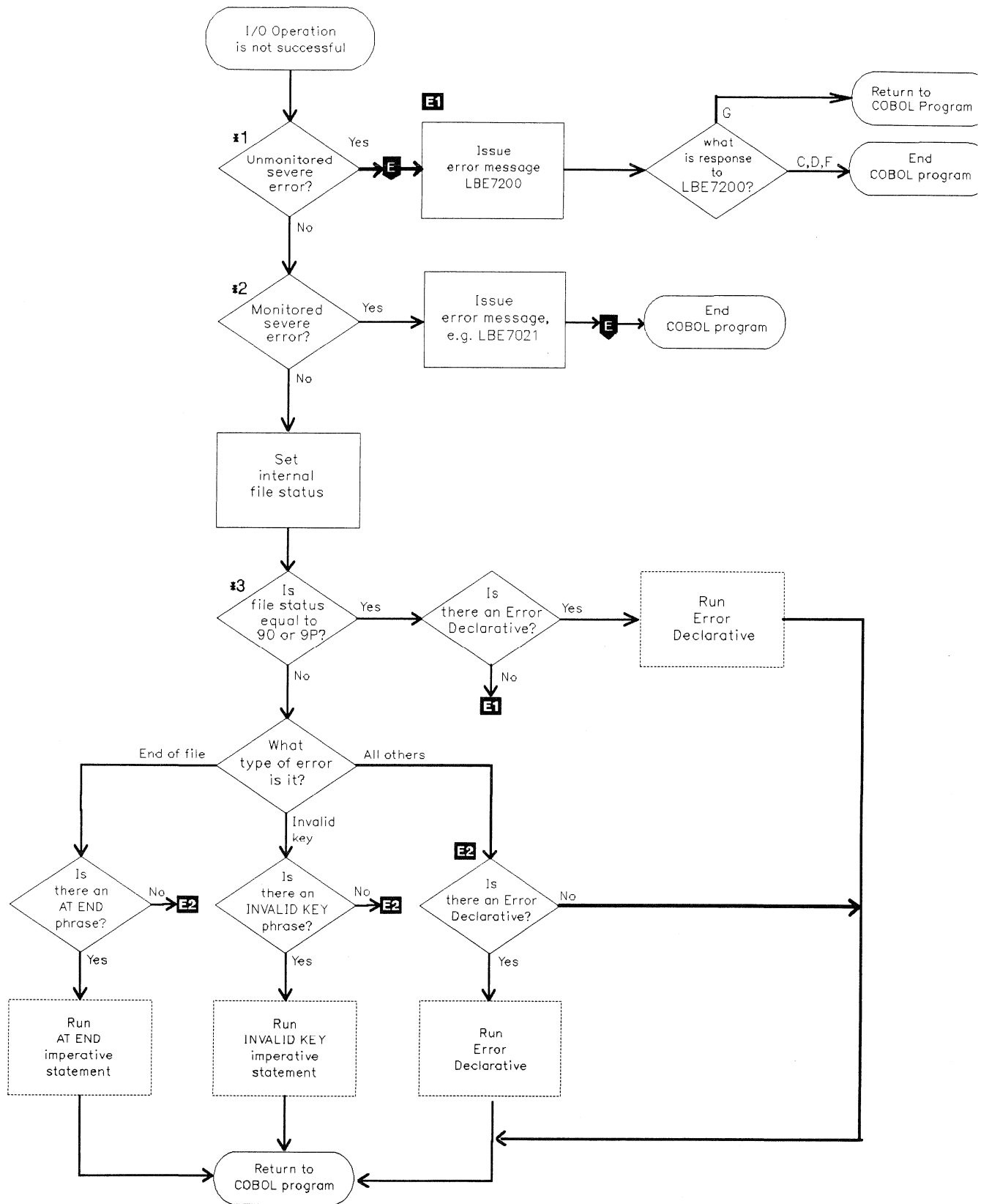
Figures 24 and 25 show the two different error handling models.



| Figure 24 (Part 1 of 2). Standard (default) Error Handling



| Figure 24 (Part 2 of 2). Standard (default) Error Handling



| Figure 25. Nonstandard Error Handling (available through *NOSTDERR option)

Other I/O exceptions may occur that COBOL does not expect. These also result in CPF4xxx and CPF5xxx messages, but there is not specific monitoring for them. Instead, they are caught by a generic I/O error handler. This error handler monitors for certain ranges of CPF4xxx and CPF5xxx messages; it sets the file status to 90 and follows the Yes branch from position *3 in Figure 25 on page 76.

An I/O exception may occur that is being specifically monitored for and which, according to the nonstandard error handling model, is severe enough to stop the program. In this situation no file status is set.

These I/O exceptions result in specific COBOL escape messages followed by an ending of the program; they follow the Yes branch from position *2 in Figure 25.

Example: CPF4238 - INDARA mismatch between program and file

There is specific monitoring for this message, and the result is error message LBE7021 followed by an ending of the program.

Other COBOL messages that fall into this category are LBE7020 and LBE7022.

During an I/O operation, a problem may occur that is not expected by the system. These problems generally result in messages (such as those starting with "MCH") that fall outside the CPF4xxx and CPF5xxx range. Such errors, known as unmonitored severe errors, follow the Yes branch from position *1 in Figure 25. These errors are handled by an all-purpose message monitor and result in an ending of the COBOL program. No file status is set.

Effects of *STDERR and *NOSTDERR on File Status

- Effects of LBE742x and LBE702x messages:

With *STDERR, file status 90 is set following the issue of LBE742x messages. The program then continues if there is a USE procedure or a FILE STATUS clause.

With *NOSTDERR, LBE702x messages cause the program to end without setting a file status.

- Ending of a program because of file status 9P or 90:

With *STDERR, a file status of 9P or 90 arising from an I/O error (signalled by CPF4xxx and CPF5xxx messages) does not cause the program to end as long as there is a USE procedure or a FILE STATUS clause. If neither exists, error message LBE7207 is issued.

With *NOSTDERR, a file status of 9P or 90 in the absence of a USE procedure causes error message LBE7200 to be issued.

- Issuing of an error message for a file status of other than zero when there is no error handler or FILE STATUS clause:

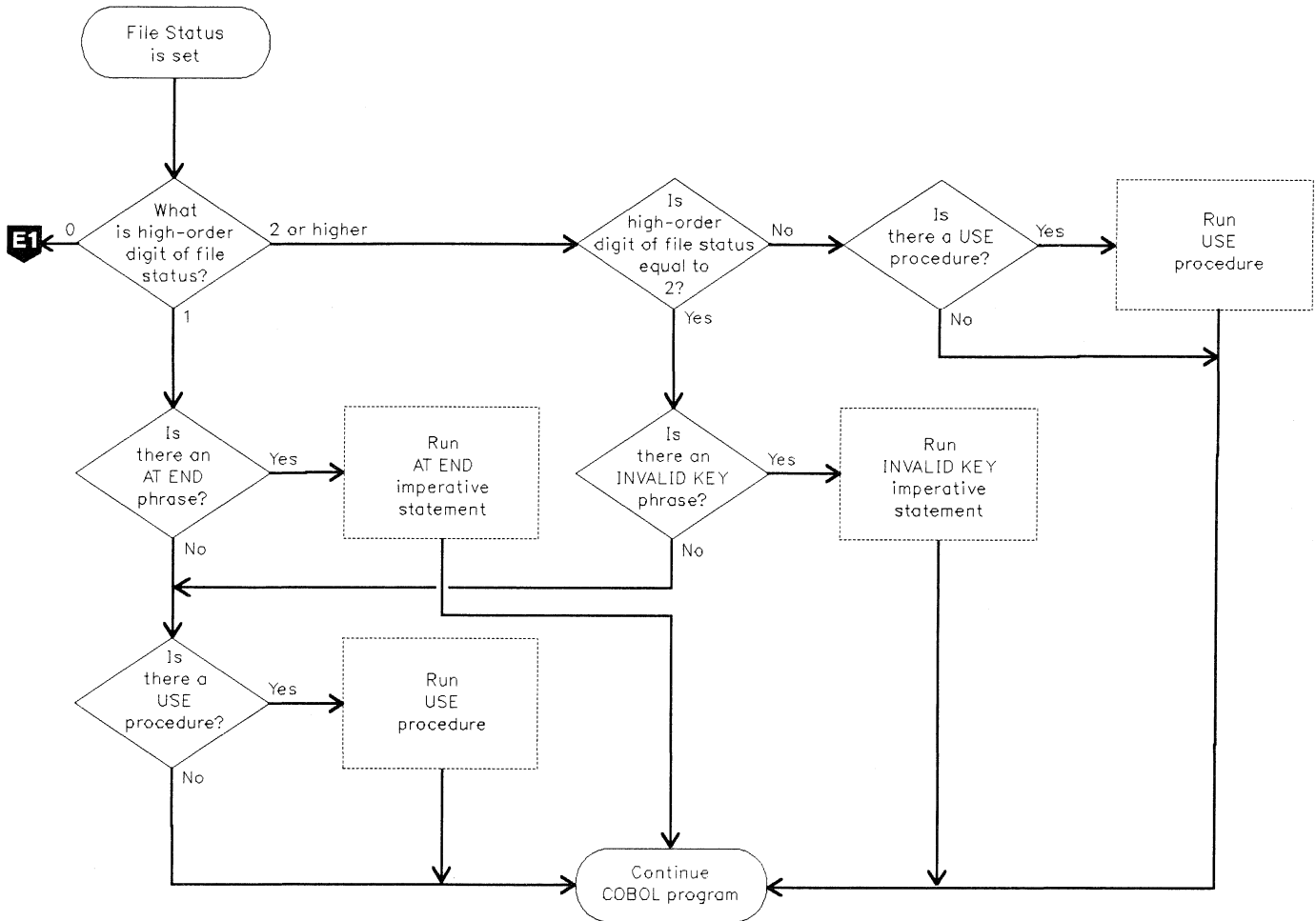
With *STDERR, a file status of other than zero when there is no AT END/INVALID KEY phrase, USE procedure, or FILE STATUS clause causes inquiry message LBE7207 (with response options C, D, F, and G) to be issued.

With *NOSTDERR, a file status of other than zero when there is no AT END/INVALID KEY phrase or USE procedure allows the program to continue unless it has already ended.

Processing of I/O Verbs

The following diagram shows when the USE procedure and the (NOT) AT END, (NOT) INVALID KEY, and NO DATA imperative statements are run. This has been in place since Version 1 Release 3, and is *independent* of the error handling method you choose (*STDERR or *NOSTDERR).

Note that the file status shown here refers to the internal file status.



Note: **E1** = Go to **E1** on next page

Figure 26 (Part 1 of 2). Processing of I/O Verbs

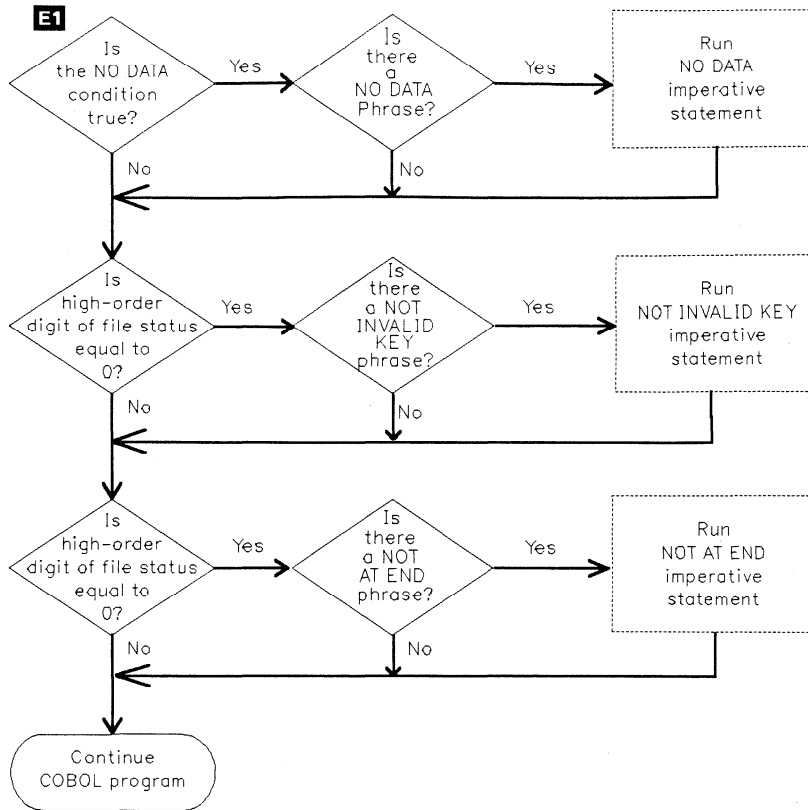


Figure 26 (Part 2 of 2). Processing of I/O Verbs

Note: Follow the parts of the diagram that apply to your statements.

Common Exceptions and Some of Their Causes

MCH1202 Decimal data error:

- A numeric elementary item has been used as a source when no valid data has been previously stored in it. The item should have a VALUE clause, or a MOVE statement should be used to initialize its value.
- An attempt has been made to place nonnumeric data in a numeric item.
- Bad data was written to a subfile earlier in the program. The subfile data is not validated until it is written to the display, so the 1202 error can occur on the WRITE of a subfile control record, but the bad data was actually put to the subfile earlier.

MCH0601 Pointer exceptions:

- Part of a linkage section item extended beyond the space allocated.
For example, if you set the address of a linkage section item, and one or more of its elementary data items extend beyond the space with a MOVE to the elementary data item, MCH0601 is issued.

For more information on using pointers, refer to "Using Pointers in a COBOL/400 Program" on page 278.

MCH0602 Pointer alignment:

- The pointer alignment in the Working-Storage Section of the calling program does not match the alignment in the Linkage Section of the called program. Alignment must be on a 16-byte boundary.

For more information on using pointers, refer to “Using Pointers in a COBOL/400 Program” on page 278.

MCH3601 Pointer error:

- A reference is made to a record or a field within a record and the associated file has been closed or has never been opened.

For example, the OPEN for the file was unsuccessful and the processing of any other I/O statement for that file is attempted. The file status should be checked before any other I/O is attempted.

CPF2415 End of requests:

- An attempt has been made to accept input from the job input stream while the system is running in batch mode and no input is available.

Recovery After a Failure

Recovery with Commitment Control

When the system is restarted after a failure, files under commitment control are automatically restored to their status at the last commitment boundary. For additional information about commitment control, see “Commitment Control Considerations” on page 92.

For a job failure (either because of user or system error), files under commitment control are restored as part of job termination to the files’ status at the previous commitment boundary.

Because files under commitment control are rolled back after system or process failure, this feature can be used to help in restarting. You can create a separate record to store data that may be useful should it become necessary to restart a job. This restart data can include items such as totals, counters, record key values, relative key values, and other relevant processing information from an application.

If you keep the restart data mentioned above in a file under commitment control, the restart data will also be permanently stored in the database when a COMMIT statement is issued. When a ROLLBACK occurs after job or process failure, you can retrieve a record of the extent of processing successfully processed before failure. Note that the above method is only a suggested programming technique and will not always be suitable, depending on the application.

TRANSACTION File Recovery

In some cases, you can recover from I/O errors on TRANSACTION files without intervention by the operator, or the varying off/varying on of work stations or communications devices.

For potentially recoverable I/O errors on TRANSACTION files, the system initiates action in addition to the steps that must be taken in the application program to attempt error recovery. For more information about action taken by the system, see the *Communications Management Guide*.

By examining the file status after an I/O operation, the application program can determine whether a recovery from an I/O error on the TRANSACTION file is possible. If the File Status Key has a value of 9N, the application program may be able to recover from the I/O error. A recovery procedure must be coded as part of the application program and varies depending on whether a single device was acquired by the TRANSACTION file or whether multiple devices were attached.

For a file with one acquired device:

1. Close the TRANSACTION file with the I/O error.
2. Reopen the file.
3. Process the steps necessary to retry the failing I/O operation. This may involve a number of steps, depending on the type of program device used. (For example, if the last I/O operation was a READ, you may have to repeat one or more WRITE statements, which were processed prior to the READ statement.) For more information on recovery procedures, see the *ICF Programmer's Guide*.

For a display file with multiple devices acquired:

1. DROP the program device that caused the I/O error on the TRANSACTION file.
2. ACQUIRE the same program device.
3. See Step 3 above.

For an ICF file with multiple devices acquired:

1. ACQUIRE the same program device.
2. See Step 3 above.

For a display file with multiple devices acquired:

Application program recovery attempts should typically be tried only once.

If the recovery attempt fails:

- If the file has only one program device attached, terminate the program through processing of the STOP RUN, EXIT PROGRAM, or GOBACK statement, and attempt to locate the source of the error.
- If the file has multiple acquired program devices, you may want to do one of the following:
 - Continue processing without the program device that caused the I/O error on the TRANSACTION file, and reacquire the device later.
 - End the program.

For a description of major and minor return codes that may help in diagnosing I/O errors on the TRANSACTION file, see the *ICF Programmer's Guide* or the *Data Management Guide*.

Figure 27 on page 82 gives an example of an error recovery procedure.

File			Keying Instruction		Graphic		Description		Page of	
Programmer		Date			Key					

Sequence Number	Conditioning							Name	Length	Location	Functions	
	Form Type	Indicator	Indicator	Indicator	Indicator	Indicator	Indicator					
1	2	3	4	5	6	7	8	9	10	11	12	
A*	DISPLAY FILE FOR ERROR RECOVERY EXAMPLE											
A*										INDARA		
A*							R	FORMAT1		CF01(01 'END OF PROGRAM')		
A*									5	I 12 28'ENTER INPUT '		
A*								INPUTFLD		I 12 42		
A*										28 26'F1 - TERMINATE'		

Figure 27. Example of Error Recovery Procedure -- DDS

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                02/01/89
 2 000200 PROGRAM-ID. RECOVERY.                                    02/05/89
 3 000300 ENVIRONMENT DIVISION.                                    02/01/89
 4 000400 CONFIGURATION SECTION.                                  02/01/89
 5 000500 SOURCE-COMPUTER. IBM-AS400.                            02/02/89
 6 000600 OBJECT-COMPUTER. IBM-AS400.                            02/02/89
 7 000700 INPUT-OUTPUT SECTION.                                  02/01/89
 8 000800 FILE-CONTROL.                                          02/01/89
 9 000900     SELECT RECOVFILE                                     02/05/89
10 001000         ASSIGN TO WORKSTATION-RECVFILE-SI                03/22/89
11 001100             ORGANIZATION IS TRANSACTION                 02/05/89
12 001200             ACCESS MODE IS SEQUENTIAL                  02/01/89
13 001300             FILE STATUS IS STATUS-FLD, STATUS-FLD-2    02/05/89
14 001400             CONTROL-AREA IS CONTROL-FLD.               02/05/89
15 001500     SELECT PRINTER-FILE                                 02/05/89
16 001600         ASSIGN TO PRINTER-QPRINT.                       02/05/89
    001700                                                         02/01/89
17 001800 DATA DIVISION.                                         02/01/89
18 001900 FILE SECTION.                                           02/01/89
19 002000 FD RECOVFILE                                           02/05/89
20 002100     LABEL RECORDS ARE OMITTED                          02/05/89
21 002200     DATA RECORD IS RECOV-REC.                          02/05/89
22 002300 01 RECOV-REC.                                           02/05/89
23 002400     COPY DDS-ALL-FORMATS OF RECVFILE.                  03/22/89
24 +000001     05 RECVFILE-RECORD PIC X(5).                       <-ALL-FMTS
+000002* INPUT FORMAT:FORMAT1 FROM FILE RECVFILE OF LIBRARY COBNATEX <-ALL-FMTS
+000003*                                                         <-ALL-FMTS
25 +000004     05 FORMAT1-I REDEFINES RECVFILE-RECORD.           <-ALL-FMTS
26 +000005     06 INPUTFLD PIC X(5).                               <-ALL-FMTS
+000006* OUTPUT FORMAT:FORMAT1 FROM FILE RECVFILE OF LIBRARY COBNATEX <-ALL-FMTS
+000007*                                                         <-ALL-FMTS
+000008*     05 FORMAT1-0 REDEFINES RECVFILE-RECORD.           <-ALL-FMTS
    002500
27 002600 FD PRINTER-FILE.
28 002700 01 PRINTER-REC.
29 002800     05 PRINTER-RECORD PIC X(132).
    002900
30 003000 WORKING-STORAGE SECTION.
    003100
31 003200 01 I-0-VERB PIC X(10).
32 003300 01 STATUS-FLD PIC X(2).
33 003400     88 NO-ERROR VALUE "00".
34 003500     88 ACQUIRE-FAILED VALUE "9H".
35 003600     88 TEMPORARY-ERROR VALUE "9N".
36 003700 01 STATUS-FLD-2 PIC X(4).
37 003800 01 CONTROL-FLD.
38 003900     05 FUNCTION-KEY PIC X(2).
39 004000     05 PGM-DEVICE-NAME PIC X(10).
40 004100     05 RECORD-FORMAT PIC X(10).
41 004200 01 END-INDICATOR PIC 1 INDICATOR 1
42 004300     VALUE B"0".
43 004400     88 END-NOT-REQUESTED VALUE B"0".
44 004500     88 END-REQUESTED VALUE B"1".
45 004600 01 USE-PROC-FLAG PIC 1
46 004700     VALUE B"0".
47 004800     88 USE-PROC-NOT-EXECUTED VALUE B"0".
48 004900     88 USE-PROC-EXECUTED VALUE B"1".
49 005000 01 RECOVERY-FLAG PIC 1
50 005100     VALUE B"0".
51 005200     88 NO-RECOVERY-DONE VALUE B"0".
52 005300     88 RECOVERY-DONE VALUE B"1".
53 005400 01 HEADER-LINE.
54 005500     05 FILLER PIC X(60)
55 005600     VALUE SPACES.
56 005700     05 FILLER PIC X(72)
57 005800     VALUE "ERROR REPORT".
58 005900 01 DETAIL-LINE.
59 006000     05 FILLER PIC X(15)
60 006100     VALUE SPACES.
61 006200     05 DESCRIPTION PIC X(25)
62 006300     VALUE SPACES.
63 006400     05 DETAIL-VALUE PIC X(92)
64 006500     VALUE SPACES.

```

Figure 28 (Part 1 of 3). Example of Error Recovery Procedure

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
65 006600 01 MESSAGE-LINE.
66 006700 05 FILLER          PIC X(15)
67 006800          VALUE SPACES.
68 006900 05 DESCRIPTION    PIC X(117)
69 007000          VALUE SPACES.
70 007100 PROCEDURE DIVISION.
   007200 DECLARATIVES.
   007300 HANDLE-ERRORS SECTION.
   007400     USE AFTER STANDARD ERROR PROCEDURE ON RECOVFILE. 1
   007500     DISPLAY-ERROR.
71 007600     SET USE-PROC-EXECUTED TO TRUE.
72 007700     WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
73 007800     MOVE "ERROR OCCURRED IN" TO DESCRIPTION OF DETAIL-LINE.
74 007900     MOVE I-O-VERB TO DETAIL-VALUE OF DETAIL-LINE.
75 008000     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
76 008100     MOVE "FILE STATUS =" TO DESCRIPTION OF DETAIL-LINE.
77 008200     MOVE STATUS-FLD TO DETAIL-VALUE OF DETAIL-LINE. 2
78 008300     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
79 008400     MOVE "EXTENDED FILE STATUS =" TO DESCRIPTION OF DETAIL-LINE.
80 008500     MOVE STATUS-FLD-2 TO DETAIL-VALUE OF DETAIL-LINE.
81 008600     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
82 008700     MOVE "CONTROL-AREA =" TO DESCRIPTION OF DETAIL-LINE.
83 008800     MOVE CONTROL-FLD TO DETAIL-VALUE OF DETAIL-LINE.
84 008900     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
   009000 CHECK-ERROR.
85 009100     IF TEMPORARY-ERROR AND NO-RECOVERY-DONE THEN
86 009200         MOVE "***ERROR RECOVERY BEING ATTEMPTED***" 3
87 009300             TO DESCRIPTION OF MESSAGE-LINE
88 009400             WRITE PRINTER-REC FROM MESSAGE-LINE
89 009500             AFTER ADVANCING 3 LINES
90 009600             PERFORM ERROR-RECOVERY
91 009700         ELSE
92 009800             IF RECOVERY-DONE THEN 4
93 009900                 MOVE "***ERROR AROSE FROM RETRY AFTER RECOVERY***"
94 010000                     TO DESCRIPTION OF MESSAGE-LINE
95 010100                     WRITE PRINTER-REC FROM MESSAGE-LINE
96 010200                     AFTER ADVANCING 3 LINES
97 010300                     MOVE "***PROGRAM TERMINATED***"
98 010400                         TO DESCRIPTION OF MESSAGE-LINE
99 010500                         WRITE PRINTER-REC FROM MESSAGE-LINE
100 010600                             AFTER ADVANCING 2 LINES
101 010700                             GO TO ERROR-EXIT
102 010800                 ELSE
103 010900                     SET NO-RECOVERY-DONE TO TRUE.
104 011000                     MOVE "***EXECUTION CONTINUES***"
105 011100                         TO DESCRIPTION OF MESSAGE-LINE.
106 011200                         WRITE PRINTER-REC FROM MESSAGE-LINE
107 011300                             AFTER ADVANCING 2 LINES.
108 011400                             GO TO END-OF-DECLARATIVES.
109 011500 ERROR-RECOVERY.
110 011600     SET RECOVERY-DONE TO TRUE.
111 011700     DROP PGM-DEVICE-NAME FROM RECOVFILE.
112 011800     ACQUIRE PGM-DEVICE-NAME FOR RECOVFILE. 5
113 011900     ERROR-EXIT.
114 012000     CLOSE RECOVFILE
115 012100         PRINTER-FILE.
116 012200     END-OF-DECLARATIVES.
117 012300     END DECLARATIVES.
118 012400
119 012500 MAIN-PROGRAM SECTION.
120 012600 MAINLINE.
121 012700     MOVE "OPEN" TO I-O-VERB.
122 012800     OPEN I-O     RECOVFILE
123 012900         OUTPUT PRINTER-FILE.
124 013000     PERFORM I-O-PARAGRAPH UNTIL END-REQUESTED. 6
125 013100     CLOSE RECOVFILE
126 013200         PRINTER-FILE.
127 013300     STOP RUN.
128 013400 I-O-PARAGRAPH.
129 013500     MOVE "WRITE" TO I-O-VERB.
130 013600     SET USE-PROC-NOT-EXECUTED TO TRUE.
131 013700     WRITE RECOV-REC FORMAT IS "FORMAT1"
132 013800         INDICATOR IS END-INDICATOR.
133 013900     IF USE-PROC-EXECUTED AND RECOVERY-DONE THEN 7
134 014000         GO TO I-O-PARAGRAPH.

```

Figure 28 (Part 2 of 3). Example of Error Recovery Procedure


```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B..+...2....+...3....+...4....+...5....+...6....+...7..IDENTFCN S COPYNAME  CHG DATE
113 014100 MOVE "READ" TO I-O-VERB.
114 014200 SET USE-PROC-NOT-EXECUTED TO TRUE.
115 014300 SET NO-RECOVERY-DONE TO TRUE.
116 014400 READ RECOVFILE FORMAT IS "FORMAT1"
      014500 INDICATOR IS END-INDICATOR. 8
117 014600 IF NO-ERROR THEN
118 014700 PERFORM SOME-PROCESSING.
      014800 SOME-PROCESSING.
119 014900 (INSERT SOME DATABASE PROCESSING, FOR EXAMPLE).
          ***** END OF SOURCE *****

```

Figure 28 (Part 3 of 3). Example of Error Recovery Procedure

- 1 This defines processing that takes place when an I/O error occurs on RECOVFILE.
- 2 This prints out information to help in diagnosing the problem.
- 3 If the file-status equals 9N (temporary error), and no previous error recovery has been attempted for this I/O operation, error recovery is now attempted.
- 4 To avoid program looping, recovery is not attempted now if it was attempted previously.
- 5 Recovery consists of dropping, then reacquiring, the program device on which the I/O error occurred.
- 6 The mainline of the program consists of writing to and reading from a device until the user signals an end to the program by pressing F1.
- 7 If the WRITE operation failed but recovery was done, the WRITE is attempted again.
- 8 If the READ operation failed, processing will continue by writing to the device again, and then attempting the READ again.

Chapter 7. File and Data Management

This chapter contains general file and data management information you may need when creating COBOL/400 applications.

This chapter describes:

- The device-independent and device-dependent characteristics of COBOL/400 programs on the AS/400 system
- Input and output spooling functions
- System override considerations
- File and record locking considerations
- Commitment control
- Unblocking and blocking records
- File status and feedback areas
- General information about the use of program-described files and externally described files in a COBOL/400 program
- The Format 2 COPY statement (DD, DDR, DDS, or DDSR option).

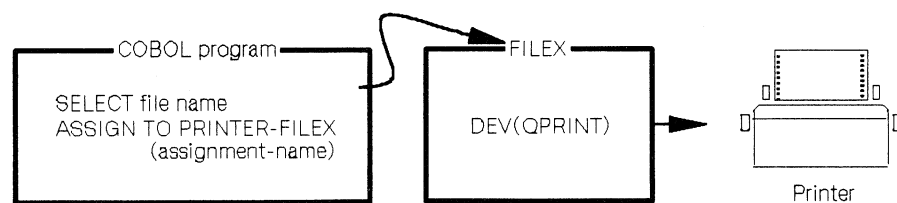
The maximum number of files that you can define and open within number of files used by a program a COBOL program is 99. If you use extended display options, the maximum number is 98. For information on specifying the extended display options, refer to page 23 .

Device Independence and Device Dependence

The key element for all I/O operations on the AS/400 system is the file. All files used are defined to the operating system. The operating system maintains a description of each file that is used by a program.

The files are kept online and serve as the connecting link between a program and the device used for I/O. The actual device association is made when the file is processed. In some instances, this type of I/O control allows the user to change the attribute of the file (and, in some cases, change the device) used in a program without changing the program.

In the COBOL/400 language, the file name specified in the ASSIGNMENT-NAME entry of the ASSIGN clause of the file control entry is used to point to the file. This file name points to the system file description:

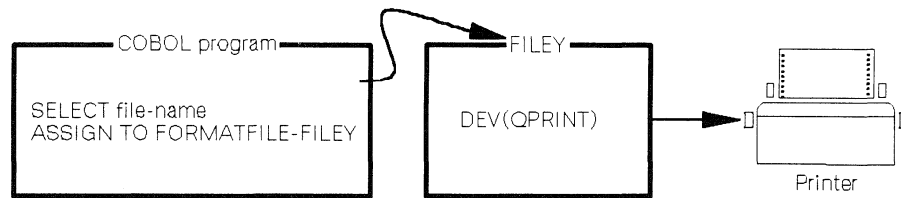


The COBOL device name in the ASSIGN clause defines the COBOL functions that can be processed on the selected file. At compilation time, certain COBOL

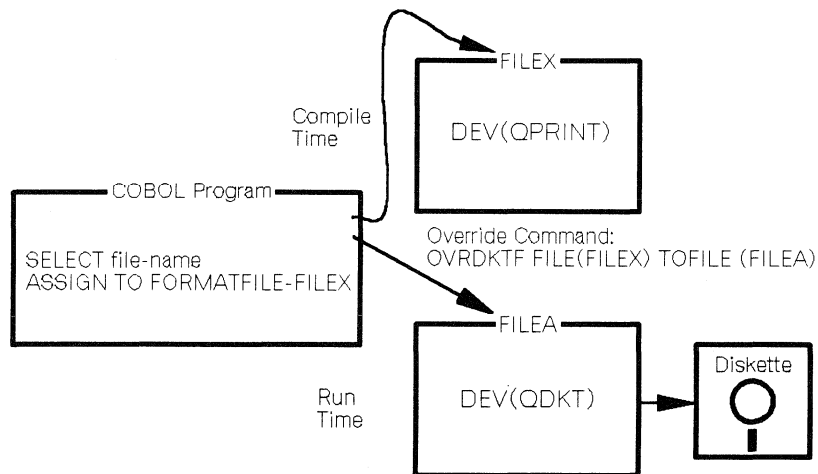
functions are valid only for a specific COBOL device name; in this respect, COBOL is device dependent. The following are examples of device dependency:

- SUBFILE operations are valid only for a WORKSTATION device.
- Indicators are valid only for WORKSTATION or FORMATFILE devices.
- LINAGE is valid only for the PRINTER device.
- OPEN INPUT WITH NO REWIND is valid only for a TAPEFILE device.

For example, assume that the file name FILEY is associated in the COBOL program with the FORMATFILE device. The device FORMATFILE is an independent device type. Therefore, no line or page control specifications are valid in the COBOL program in the WRITE ADVANCING statement. When the program is run, the actual I/O device is specified in the description of FILEY. For example, the device might be a printer; only the default line and page control or those defined in the DDS would be used:



CL commands can be used to override a parameter in the specified file description or to redirect a file at compilation time or run time. File redirection allows the user to specify one file at compilation time and another file at run time:



In the preceding example, the Override to Diskette File command (OVRDKTF) allows the program to run with an entirely different device file than was specified at compilation time.

Not all file redirections or overrides are valid. At run time, checking occurs to ensure that the specifications within the COBOL program are valid for the file being processed. The OS/400 operating system allows some file redirections even if device specifics are contained in the program. For example, if the COBOL device name is PRINTER and the actual file the program uses is not a

printer, the operating system ignores the COBOL print spacing and skipping specifications.

There are other file redirections that the operating system does not allow and that cause program termination. For example, if the COBOL device name is DATABASE or DISK and a keyed READ operation is specified in the program, the program is terminated if the actual file the program uses is not a disk or data-base file.

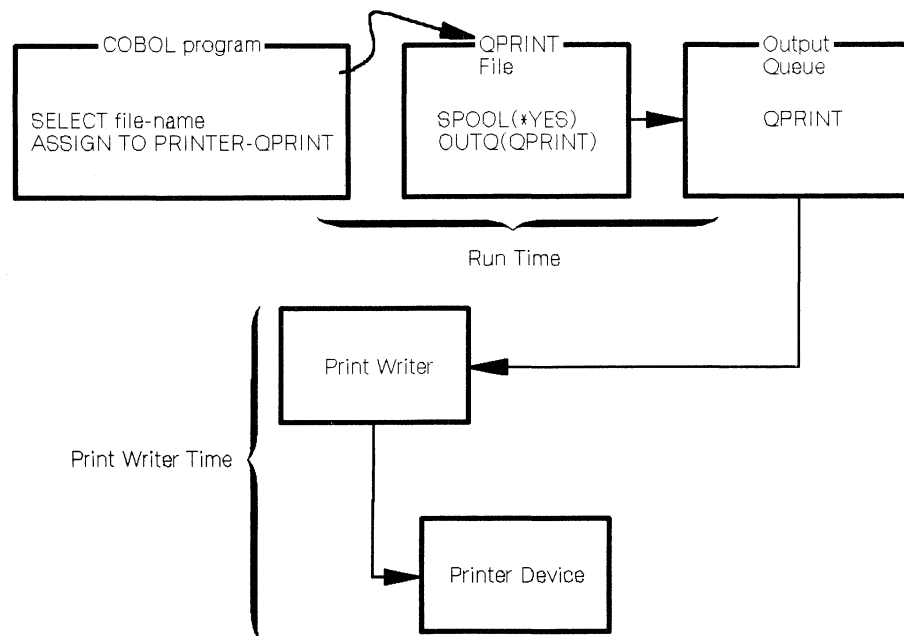
See "System Override Considerations" on page 90 for more detailed information on valid file redirections and file overrides.

Spooling

The AS/400 system provides for the use of input and output spooling functions. Each AS/400 file description contains a spool attribute that determines whether spooling is used for the file at run time. The COBOL program is not aware that spooling is being used. The actual physical device from which a file is read or to which a file is written is determined by the spool reader or the spool writer. See the *Data Management Guide* for more detailed information on spooling.

Output Spool

Output spooling is valid for batch and interactive jobs. The description of the file that is specified in COBOL by the system-name contains the specification for spooling as shown in the following example:

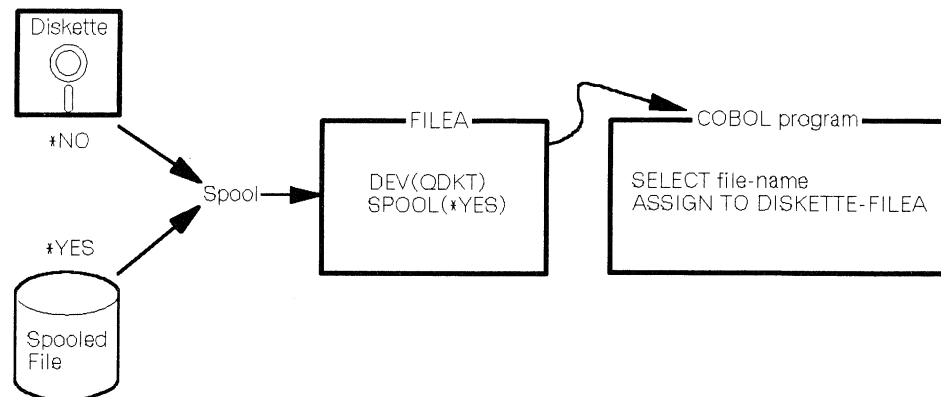


File override commands can be used at run time to override the spooling options that are specified in the file description, such as the number of copies to be printed. In addition, AS/400 spooling support allows you to redirect a file after the program has run. For example, you can direct the printer output to a different device, such as a diskette.

Input Spool

Input spooling is valid only for inline data files in batch jobs. If the input data read by COBOL comes from a spooled file, COBOL is not aware of which device the data was spooled in from.

The data is read from a spooled inline file:



See the *Data Management Guide* for more information on inline data files.

System Override Considerations

You must specify any overrides before the file is opened by the COBOL program. The system uses the file override command to determine the file to open and the attributes of the file.

The simplest form of overriding a file is to override some attributes of the file. For example, FILE(OUTPUT) with COPIES(2) is specified when a printer file is created. Then, before the COBOL program is run, the number of printed copies of output can be changed to 3. The override command is as follows:

```
OVRPRTF FILE(OUTPUT) COPIES(3)
```

Another form of file overriding is to redirect the COBOL program to access a different file. When the override redirects the program to a file of the *same* type (such as a printer file to another printer file), the file is processed in the same manner as the original file.

When the override redirects the program to a file of a *different* type, the overriding file is processed in the same manner as the original file would have been processed. Device-dependent specifications in the COBOL program are ignored, and the defaults are taken by the system.

Not all file redirections are valid. For example, an indexed file for a COBOL program can only be overridden to another indexed file with a keyed access path.

Multiple member processing can be accomplished for a database file by overriding a database file to process all members. Note the following exceptions:

- A database source file used for a COBOL program cannot be overridden to process all members. Specifying OVRDBF MBR(*ALL) will result in the termination of the compilation.

- A database file used for a COPY statement cannot be overridden to process all members. Specifying OVRDBF MBR(*ALL) will cause the COPY statement to be ignored.

The COBOL programmer must ensure that file overrides are applied properly. For more information on valid file redirections, the device dependent characteristics ignored, and the defaults assumed, see the *Data Management Guide*.

File and Record Locking by COBOL

The operating system allows a lock state (exclusive, exclusive allow read, shared-for-update, shared-no-update, or shared-for-read) to be placed on a file used during a job step. The file can be placed in a lock state with the Allocate Object (ALCOBJ) command.

By default, the operating system places the following lock states on database files when the files are opened by COBOL programs:

OPEN Type	Lock State
INPUT	Shared-for-read
I/O	Shared-for-update
EXTEND	Shared-for-update
OUTPUT	Shared-for-update

EXTEND mode is a method of adding records to the end of a sequential file when the file is opened.

The shared-for-read lock state allows another user to open the file with a lock state of shared-for-read, shared-for-update, shared-no-update, or exclusive-allow-read, but the user cannot specify the exclusive use of the file. The shared-for-update lock state allows another user to open the file with a shared-for-read or shared-for-update lock state.

The operating system places the shared-for-read lock on the device file and an exclusive-allow-read lock state on the device. Another user can open the file but cannot use the same device.

Note: When a COBOL program opens a physical file for OUTPUT, that file will be subject to an exclusive lock for the period of time necessary to clear the member.

For more information on allocating resources and the lock states, see the *Data Management Guide*.

Locking and Releasing Records

When a database record is read by COBOL and the file is opened for I/O, a lock is placed on that record so that another program cannot update it. That is, the record can be read by another program if it opens a file for input, but not if it opens the file for I/O.

For information about the duration of record lock with and without commitment control, refer to Table 2 on page 94.

To prevent the READ statement from locking records on files opened in I/O (update) mode, you can use the NO LOCK phrase. The READ WITH NO LOCK statement unlocks records locked by a previous READ statement. For more

information about this phrase, refer to the section on the READ statement in the *COBOL/400* Reference*.

For a logical file based on one physical file, the lock is placed on the record in the physical file. If a logical file is based on more than one physical file, a lock is placed on one record in each physical file.

This lock applies not only to other programs, but also to the original program if it attempts to update the same underlying physical record through a second file.

Note: When a file with indexed or relative organization is opened for I/O, using random or dynamic access, a failed I/O operation on any of the I/O verbs except WRITE also unlocks the record. A WRITE operation is not considered an update operation; therefore, the record lock is not released.

For more information about releasing database records read for update, see the *Data Management Guide*.

Sharing an Open Data Path

If you have already opened a file through another program in your routing step, your COBOL program can use the same Open Data Path (ODP) to access the file.

Note: Routing steps are described in the *Programming: Work Management Guide*; a job usually contains only one routing step.

The following rules apply to shared ODPs:

1. You must specify SHARE(*YES) in the command that creates the file, in a change command, or in an override command for the file.
2. Once a file with a shared ODP has been opened for the first time by a program and remains open, subsequent OPEN operations within the same routing step run faster than standard OPEN operations. The speed of I/O operations other than opens is not affected.
3. Your use of the file within your different programs should be consistent. For example, if a non-COBOL program performs a READ PREVIOUS operation using blocked I/O, the COBOL READ statement might retrieve the record preceding the current file position rather than the record following the current file position.

Commitment Control Considerations

Commitment control is a function that allows:

- Synchronization of changes to database files within the same job
- Cancellation of changes that should not be permanently entered into the database
- Locking of records being changed until changes are complete
- Techniques for recovering from job or system failure.

In some applications, it is desirable to synchronize changes to database records. If the program determines the changes are valid, the changes are then permanently made to the database (a COMMIT statement is processed). If the changes are not valid, or if a problem occurs during processing, the changes can be canceled (a ROLLBACK statement is processed). (When a file is cleared after being opened for OUTPUT, processing of a ROLLBACK does not restore cleared

records to the file.) Changes made to records in a file that is *not* under commitment control are always permanent. Such changes are never affected by subsequent COMMIT or ROLLBACK statements.

Each point where a COMMIT or ROLLBACK is successfully processed is a commitment boundary. (If no COMMIT or ROLLBACK has yet been issued in a program, a commitment boundary is created by the first open of any file under commitment control.) The committing or rolling back of changes only affects changes made since the previous commitment boundary.

The synchronizing of changes at commitment boundaries makes restart or recovery procedures after a failure easier. For more information, see "Recovery After a Failure" on page 80.

When commitment control is used for database files, records in those files are subject to either a high lock level LCKLVL (*ALL) or a low lock level LCKLVL(*CHG). With a low lock level (*CHG), all records that are changed (rewritten, deleted, or added) in files under commitment control are locked until a COMMIT or ROLLBACK statement is successfully processed. With a high lock level (*ALL), *all* records accessed, whether for input or output, are locked until a COMMIT or ROLLBACK is successfully processed. For both record locking levels, no other job can modify data in locked records until the COMMIT or ROLLBACK has been successfully completed. (A locked record can only be modified within the same job and through the same physical or logical file.)

The lock level also governs whether locked records can be read. With a high lock level (*ALL), you cannot read locked records in a database file.

With a low lock level (*CHG), you can read locked records in a database file, provided the file is opened as INPUT in your job, or opened as I/O and READ WITH NO LOCK is used.

Other jobs, where files are *not* under commitment control, can always read locked records, regardless of the lock level used, provided the files are opened as INPUT. Because it is possible in some cases for other jobs to read locked records, data can be accessed *before it is permanently committed to a database*. If a ROLLBACK statement is processed *after* another job has read locked records, the data accessed will not reflect the contents of the database.

Table 2 shows record locking considerations for files with and without commitment control.

Table 2. Record Locking Considerations with and without Commitment Control

VERB	OPEN MODE	LOCK LEVEL	DURATION OF RECORD LOCK	
			Next I/O Operation	COMMIT or ROLLBACK
DELETE	I-O	Without Commitment Control	DELETE	→
		With Commitment Control	→	→
READ	INPUT	Without Commitment Control	READ	→
		With Commitment Control	→	→
READ WITH NO LOCK	I-O	Without Commitment Control	READ	→
		With Commitment Control	→	→
READ	I-O	Without Commitment Control	READ	→
		With Commitment Control	→	→
REWRITE	I-O	Without Commitment Control	REWRITE	→
		With Commitment Control	→	→
START	INPUT	Without Commitment Control	START	→
		With Commitment Control	→	→
START	I-O	Without Commitment Control	START	→
		With Commitment Control	→	→
WRITE	I-O	Without Commitment Control	WRITE	→
		With Commitment Control	→	→
WRITE	OUTPUT	Without Commitment Control	WRITE	→
		With Commitment Control	→	→

A file under commitment control can be closed or opened without affecting the status of changes made since the last commitment boundary. A COMMIT must still be issued to make the changes permanent, or a ROLLBACK issued to cancel the changes. A COMMIT statement, when processed, leaves files in the same open or closed state as before processing.

All files under commitment control within the same job must be journaled to the same journal. For more information about journal management and its related functions, and for more information about commitment control, refer to the *Advanced Backup and Recovery Guide*.

Commitment control must also be specified outside the COBOL language through the OS/400 control language (CL). The Start Commitment Control (STRCMTCTL) command establishes the capability for commitment control and sets the level of record locking at the high level (*ALL), or the low level (*CHG). The STRCMTCTL command does not automatically initiate commitment control for a file. That file must also be specified in the COMMITMENT CONTROL clause of the I-O-CONTROL paragraph within the COBOL program. The commitment control environment is normally ended by using the End Commitment Control (ENDCMTCTL) command. This causes any uncommitted changes for database

files under commitment control to be canceled. (An implicit ROLLBACK is processed.) Refer to the *CL Reference* for more information on the STRCMTCTL and ENDCMTCTL commands.

For more information about commitment control, see the *Advanced Backup and Recovery Guide*.

Note: The ability to prevent reading of uncommitted data that has been changed is a function of commitment control and is only available if you are running under commitment control. Normal (noncommitted) database support is not changed by the commitment control extension, and allows reading of locked records when a file that is opened only for input is read. Try to use files consistently. Typically, files should always be run under commitment control or never be run under commitment control.

Figure 29 on page 96 illustrates a possible usage of commitment control in a banking environment. The program processes transactions for transferring funds from one account to another. If no problems occur during the transaction, the changes are committed to the database file. If the transfer cannot take place because of improper account number or insufficient funds, a ROLLBACK is issued to cancel the changes.

File Programmer	Date	Keying Instruction	Graphic Key	Description	Page of
--------------------	------	-----------------------	----------------	-------------	------------

Sequence Number	Form Type ACCT/DS/Comment A/D/*	Indicator	Indicator	Indicator	Indicator	Indicator	Name	Length	Reference (R)	Data Type/Keyboard Shift Data Position Usage (W/D/L/B/N/M/N/P)	Location		Functions	
											Line	Pos		
1	A						PROMPT SCREEN FILE NAME 'ACCTFMST'							
2	A											1	INDARA	
3	A						R ACCTPMT						TEXT('CUSTOMER ACCOUNT PROMPT')	
4	A												CA01(15 'END OF PROGRAM')	
5	A												PURETAIN OVERLAY	
6	A											1	3'ACCOUNT MASTER UPDATE'	
7	A											3	3'FROM ACCOUNT NUMBER'	
8	A						ACCTFROM	5Y 01				3	23CHECK(ME)	
9	A	98											ERRMSG('INVALID FROM ACCOUNT + NUMBER' 99)	
10	A	98											ERRMSG('INSUFFICIENT FUNDS IN FROM + ACCOUNT' 98)	
11	A											4	3'TO ACCOUNT NUMBER'	
12	A						ACCTTO	5Y 01				4	23CHECK(ME)	
13	A	97											ERRMSG('INVALID TO ACCOUNT + NUMBER' 97)	
14	A											5	3'AMOUNT TRANSFERRED'	
15	A						TRANSAMT	10Y02I				5	23	
16	A						R ERRFMT						8	5'INVALID FILE STATUS'
17	A	96											7	5'INVALID KEY IN REWRITE'
18	A	96												

Figure 29. Example of Use of Commitment Control --DDS

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B.+.....2.....3.....4.....5.....6.....7..IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                02/01/89
 2 000200 PROGRAM-ID. ACCOUNT.                                    02/04/89
 3 000300 AUTHOR. PROGRAMMER NAME.                               01/27/89
 4 000400 INSTALLATION. COBOL DEVELOPMENT CENTRE.                01/27/89
 5 000500 DATE-WRITTEN. 02/02/88.                                02/04/89
 8 000080 DATE-COMPILED. 05/24/91 14:02:39 .                    03/01/90
 7 000700 ENVIRONMENT DIVISION.                                  01/27/89
 8 000800 CONFIGURATION SECTION.                                  01/27/89
 9 000900 SOURCE-COMPUTER. IBM-AS400.                             01/27/89
10 001000 OBJECT-COMPUTER. IBM-AS400.                             01/27/89
11 001100 INPUT-OUTPUT SECTION.                                   01/27/89
12 001200 FILE-CONTROL.                                          01/27/89
13 001300 SELECT ACCOUNT-FILE ASSIGN TO DATABASE-ACCTMST        02/04/89
14 001400 ORGANIZATION IS INDEXED                                02/04/89
15 001500 ACCESS IS DYNAMIC                                       02/04/89
16 001600 RECORD IS EXTERNALLY-DESCRIBED-KEY                   02/04/89
17 001700 FILE STATUS IS ACCOUNT-FILE-STATUS.                   02/04/89
18 001800 SELECT DISPLAY-FILE ASSIGN TO WORKSTATION-ACCTFMSTS-SI 1 02/04/89
19 001900 ORGANIZATION IS TRANSACTION.                           02/04/89
002000*****
20 002100 I-O-CONTROL.                                           02/04/89
21 002200 COMMITMENT CONTROL FOR ACCOUNT-FILE. 2                 02/04/89
002300*****
22 002400 DATA DIVISION.                                         02/04/89
23 002500 FILE SECTION.                                           02/04/89
24 002600 FD ACCOUNT-FILE                                         02/04/89
25 002700 LABEL RECORDS ARE STANDARD.                             02/04/89
26 002800 01 ACCOUNT-RECORD.                                       02/04/89
27 002900 COPY DDS-ALL-FORMATS OF ACCTMST.                       02/04/89
28 +000001 05 ACCTMST-RECORD PIC X(82).                           <-ALL-FMTS
+000002* I-O FORMAT:ACCNTREC FROM FILE ACCTMST OF LIBRARY XMPLIB <-ALL-FMTS
+000003* <-ALL-FMTS
+000004*THE KEY DEFINITIONS FOR RECORD FORMAT ACCNTREC <-ALL-FMTS
+000005* NUMBER NAME RETRIEVAL TYPE ALTSEQ <-ALL-FMTS
+000006* 0001 ACCNKEY ASCENDING SIGNED NO <-ALL-FMTS
29 +000007 05 ACCNTREC REDEFINES ACCTMST-RECORD. <-ALL-FMTS
30 +000008 06 ACCNKEY PIC S9(5). <-ALL-FMTS
31 +000009 06 NAME PIC X(20). <-ALL-FMTS
32 +000010 06 ADDR PIC X(20). <-ALL-FMTS
33 +000011 06 CITY PIC X(20). <-ALL-FMTS
34 +000012 06 STATE PIC X(2). <-ALL-FMTS
35 +000013 06 ZIP PIC S9(5). <-ALL-FMTS
36 +000014 06 BALANCE PIC S9(8)V9(2). <-ALL-FMTS
003000
37 003100 FD DISPLAY-FILE
38 003200 LABEL RECORDS ARE STANDARD.
39 003300 01 DISPLAY-REC.
40 003400 COPY DDS-ALL-FORMATS OF ACCTFMSTS.
41 +000001 05 ACCTFMSTS-RECORD PIC X(20). <-ALL-FMTS
+000002* INPUT FORMAT:ACCTPMT FROM FILE ACCTFMSTS OF LIBRARY XMPLIB <-ALL-FMTS
+000003* CUSTOMER ACCOUNT PROMPT <-ALL-FMTS
42 +000004 05 ACCTPMT-I REDEFINES ACCTFMSTS-RECORD. <-ALL-FMTS
43 +000005 06 ACCTFROM PIC S9(5). <-ALL-FMTS
44 +000006 06 ACCTTO PIC S9(5). <-ALL-FMTS
45 +000007 06 TRANSAMT PIC S9(8)V9(2). <-ALL-FMTS
+000008* OUTPUT FORMAT:ACCTPMT FROM FILE ACCTFMSTS OF LIBRARY XMPLIB <-ALL-FMTS
+000009* CUSTOMER ACCOUNT PROMPT <-ALL-FMTS
+000010* 05 ACCTPMT-0 REDEFINES ACCTFMSTS-RECORD. <-ALL-FMTS
+000011* INPUT FORMAT:ERRFMT FROM FILE ACCTFMSTS OF LIBRARY XMPLIB <-ALL-FMTS
+000012* <-ALL-FMTS
+000013* 05 ERRFMT-I REDEFINES ACCTFMSTS-RECORD. <-ALL-FMTS
+000014* OUTPUT FORMAT:ERRFMT FROM FILE ACCTFMSTS OF LIBRARY XMPLIB <-ALL-FMTS
+000015* <-ALL-FMTS
+000016* 05 ERRFMT-0 REDEFINES ACCTFMSTS-RECORD. <-ALL-FMTS
46 003500 WORKING-STORAGE SECTION.
47 003600 77 ACCOUNT-FILE-STATUS PIC X(2).

```

Figure 30 (Part 1 of 3). Example of Use of Commitment Control

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B.....2.....3.....4.....5.....6.....7..IDENTFCN S COPYNAME  CHG DATE
48 003700 77 IND-ON                PIC 1  VALUE B"1".
49 003800 77 IND-OFF              PIC 1  VALUE B"0".
50 003900 01 DISPFILE-INDICS.
51 004000 COPY DDS-ALL-FORMATS-INDIC OF ACCTFMTS. 3
52 +000001 05 ACCTFMTS-RECORD.                                <-ALL-FMTS
+000002* INPUT FORMAT:ACCTPMT FROM FILE ACCTFMTS OF LIBRARY XMPLIB <-ALL-FMTS
+000003* CUSTOMER ACCOUNT PROMPT                                <-ALL-FMTS
53 +000004 06 ACCTPMT-I-INDIC.                                  <-ALL-FMTS
54 +000005 07 IN15 PIC 1 INDIC 15.                             <-ALL-FMTS
+000006* END OF PROGRAM                                        <-ALL-FMTS
55 +000007 07 IN97 PIC 1 INDIC 97.                             <-ALL-FMTS
+000008* INVALID TO ACCOUNT NUMBER                          <-ALL-FMTS
56 +000009 07 IN98 PIC 1 INDIC 98.                             <-ALL-FMTS
+000010* INSUFFICIENT FUNDS IN FROM ACCOUNT                <-ALL-FMTS
57 +000011 07 IN99 PIC 1 INDIC 99.                             <-ALL-FMTS
+000012* INVALID FROM ACCOUNT NUMBER                      <-ALL-FMTS
+000013* OUTPUT FORMAT:ACCTPMT FROM FILE ACCTFMTS OF LIBRARY XMPLIB <-ALL-FMTS
+000014* CUSTOMER ACCOUNT PROMPT                          <-ALL-FMTS
58 +000015 06 ACCTPMT-0-INDIC.                                  <-ALL-FMTS
59 +000016 07 IN97 PIC 1 INDIC 97.                             <-ALL-FMTS
+000017* INVALID TO ACCOUNT NUMBER                          <-ALL-FMTS
60 +000018 07 IN98 PIC 1 INDIC 98.                             <-ALL-FMTS
+000019* INSUFFICIENT FUNDS IN FROM ACCOUNT                <-ALL-FMTS
61 +000020 07 IN99 PIC 1 INDIC 99.                             <-ALL-FMTS
+000021* INVALID FROM ACCOUNT NUMBER                      <-ALL-FMTS
+000022* INPUT FORMAT:ERRFMT FROM FILE ACCTFMTS OF LIBRARY XMPLIB <-ALL-FMTS
+000023*                                                    <-ALL-FMTS
+000024* 06 ERRFMT-I-INDIC.                                  <-ALL-FMTS
+000025* OUTPUT FORMAT:ERRFMT FROM FILE ACCTFMTS OF LIBRARY XMPLIB <-ALL-FMTS
+000026*                                                    <-ALL-FMTS
62 +000027 06 ERRFMT-0-INDIC.                                  <-ALL-FMTS
63 +000028 07 IN95 PIC 1 INDIC 95.                             <-ALL-FMTS
64 +000029 07 IN96 PIC 1 INDIC 96.                             <-ALL-FMTS
004100
65 004200 PROCEDURE DIVISION.
004300 DECLARATIVES.
004400 ERROR-SECTION SECTION.
004500 USE AFTER STANDARD EXCEPTION PROCEDURE ON ACCOUNT-FILE.
004600 ERROR-PARAGRAPH.
66 004700 IF ACCOUNT-FILE-STATUS IS NOT EQUAL "23" THEN
67 004800 MOVE IND-ON TO IN96 OF ERRFMT-0-INDIC 4
004900 ELSE
68 005000 MOVE IND-ON TO IN95 OF ERRFMT-0-INDIC. 5
69 005100 WRITE DISPLAY-REC FORMAT IS "ERRFMT"
005200 INDICATORS ARE ERRFMT-0-INDIC.
70 005300 READ DISPLAY-FILE.
71 005400 CLOSE DISPLAY-FILE
005500 ACCOUNT-FILE.
72 005600 STOP RUN.
005700 END DECLARATIVES.
005800 MAIN-PROGRAM SECTION.
005900 MAINLINE.
73 006000 OPEN I-0 DISPLAY-FILE
006100 I-0 ACCOUNT-FILE.
74 006200 MOVE ZEROS TO ACCTPMT-I-INDIC
006300 ACCTPMT-0-INDIC.
75 006400 PERFORM WRITE-READ-DISPLAY.
76 006500 PERFORM VERIFY-ACCOUNT-NO UNTIL IN15 EQUAL IND-ON.
77 006600 CLOSE DISPLAY-FILE
006700 ACCOUNT-FILE.
78 006800 STOP RUN.
006900 VERIFY-ACCOUNT-NO.
79 007000 PERFORM VERIFY-TO-ACCOUNT.
80 007100 IF IN97 OF ACCTPMT-0-INDIC EQUAL IND-OFF THEN
81 007200 PERFORM VERIFY-FROM-ACCOUNT.
82 007300 PERFORM WRITE-READ-DISPLAY.
007400 VERIFY-FROM-ACCOUNT.
83 007500 MOVE ACCTFROM TO ACCNTKEY.
84 007600 READ ACCOUNT-FILE
85 007700 INVALID KEY MOVE IND-ON TO IN99 OF ACCTPMT-0-INDIC.
86 007800 IF IN99 OF ACCTPMT-0-INDIC EQUAL IND-ON THEN 6
007900* *
008000 ROLLBACK
008100* *
87 008200 ELSE
88 008300 PERFORM UPDATE-FROM-ACCOUNT.

```

Figure 30 (Part 2 of 3). Example of Use of Commitment Control

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B.+....2.....3.....4.....5.....6.....7..IDENTFCN S COPYNAME  CHG DATE
008400 VERIFY-TO-ACCOUNT.
89 008500 MOVE ACCTTO TO ACCNTKEY.
90 008600 READ ACCOUNT-FILE
91 008700 INVALID KEY MOVE IND-ON TO IN97 OF ACCTPMT-0-INDIC. 7
92 008800 IF IN97 OF ACCTPMT-0-INDIC EQUAL IND-ON THEN
008900* *
009000 ROLLBACK 8 *
009100* *
93 009200 ELSE
94 009300 PERFORM UPDATE-TO-ACCOUNT.
009400 UPDATE-TO-ACCOUNT.
95 009500 ADD TRANSAMT TO BALANCE.
96 009600 REWRITE ACCOUNT-RECORD.
009700 UPDATE-FROM-ACCOUNT.
97 009800 SUBTRACT TRANSAMT FROM BALANCE.
98 009900 REWRITE ACCOUNT-RECORD.
99 010000 IF BALANCE IS LESS THAN 0 THEN
100 010100 MOVE IND-ON TO IN98 OF ACCTPMT-0-INDIC *
010200* *
010300 ROLLBACK 9 *
010400* *
101 010500 ELSE *
010600* *
010700 COMMIT. 10 *
010800* *
102 010900 WRITE-READ-DISPLAY.
103 011000 WRITE DISPLAY-REC FORMAT IS "ACCTPMT"
011100 INDICATORS ARE ACCTPMT-0-INDIC. 11
104 011200 MOVE ZEROS TO ACCTPMT-I-INDIC
011300 ACCTPMT-0-INDIC.
105 011400 READ DISPLAY-FILE RECORD
011500 INDICATORS ARE ACCTPMT-I-INDIC.
011600
***** END OF SOURCE *****

```

Figure 30 (Part 3 of 3). Example of Use of Commitment Control

- 1 A separate indicator area is provided for the program.
- 2 The COMMITMENT CONTROL clause specifies files to be placed under commitment control. Any files named in this clause are affected by the COMMIT and ROLLBACK verbs.
- 3 The Format 2 COPY statement with the indicator attribute INDIC, defines data description entries in WORKING-STORAGE for the indicators to be used in the program.
- 4 IN96 is set if there is an invalid file status.
- 5 IN95 is set if there is an INVALID KEY condition on the REWRITE operation.
- 6 IN99 is set if the entered account number is invalid for the account to which money is being transferred.
- 7 IN97 is set if the entered account number is invalid for the account to which money is being transferred.
- 8 If an INVALID KEY condition occurs on the READ, a ROLLBACK is used and the record lock placed on the record after the first READ is released.
- 9 If the transfer of funds is not allowed (an indicator has been set), the ROLLBACK statement is processed. All changes made to database files under commitment control are canceled.
- 10 If the transfer of funds was valid (no indicators have been set), the COMMIT statement is processed, and all changes made to database files under commitment control become permanent.
- 11 The INDICATORS phrase is required for options on the work station display that are controlled by indicators.

Unblocking Input Records and Blocking Output Records

A **block** contains more than one record. In the interest of improving the performance of input and output operations, the COBOL compiler generates code to unblock input records and block output records in either of the following conditions:

1. *NOBLK is specified (with or without a BLOCK CONTAINS clause) and **all** of the following conditions are met:
 - a. ACCESS IS SEQUENTIAL is specified for the file.
 - b. The file is opened only for INPUT or OUTPUT in that program.
 - c. The file is assigned to DISK, DATABASE, DISKETTE, or TAPEFILE.
 - d. No START statements are specified for the file.

For RELATIVE organization, blocking is not performed for OPEN OUTPUT.

If you specify BLOCK CONTAINS, it is ignored except for tape files. For tape files, the BLOCK CONTAINS clause controls the number of records to be blocked. If you do not specify BLOCK CONTAINS, the system determines the number of records to be blocked. In the case of DISKETTE files, the system always determines the number of records to be blocked.

2. *BLK is specified with BLOCK CONTAINS and **all** of the following conditions are met:
 - a. ACCESS IS SEQUENTIAL or ACCESS IS DYNAMIC is specified for the file.
 - b. The file is opened only for INPUT or OUTPUT in that program.
 - c. The file is assigned to DISK, DATABASE, DISKETTE, or TAPEFILE.

For RELATIVE organization, blocking is not performed for OPEN OUTPUT.

The BLOCK CONTAINS clause controls the number of records to be blocked. In the case of DISKETTE files, the system always determines the number of records to be blocked.

Even when all of the above conditions are met, certain OS/400 restrictions can cause blocking and unblocking to not be processed. In these cases, performance improvements will not be realized.

If you are using dynamically accessed and indexed organization files, you can use READ PRIOR and READ NEXT to perform blocking. When using READ PRIOR and READ NEXT to perform blocking, you cannot change direction while there are records remaining in the block. To clear the records from a block, specify a random operation, such as a random READ or a random START, or use a sequential READ FIRST or READ LAST.

If an illegal change of direction takes place, file status 9U results. No further I/O is possible until the file is closed and reopened.

You can override blocking at run time by specifying SEQONLY(*NO) for the OVRDBF command.

For disk and database files, when you use BLOCK CONTAINS, and if the blocking factor of zero is specified or calculated, the system determines the blocking factor.

There are certain instances in which the blocking factor you specify may be changed. See the *Database Guide* for more information about these situations.

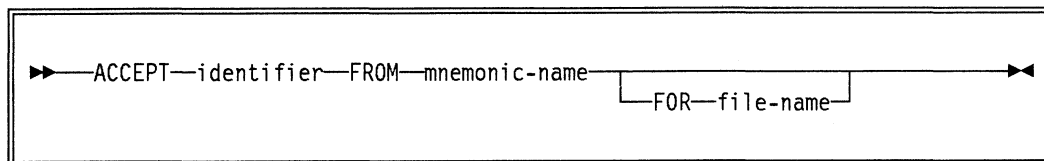
Where a block of records is written or read, the I/O feedback area contains the number of records in that block. The I/O-FEEDBACK area is not updated after each read or write for files where multiple records are blocked and unblocked by COBOL. It is updated when the next block is read or written. See "I/O FEEDBACK" in the *COBOL/400* Reference* for more information.

For database files, you may not see all changes as they occur, if the changes are made in different programs. For a description of the effect of blocking on changes to database files, see the discussion on sequential-only processing in the *Database Guide*.

File Status and Feedback Areas

To transfer data (OPEN-FEEDBACK or I-O-FEEDBACK areas) associated with an open file to an identifier use the following format:

ACCEPT Statement – Format 3 – Feedback



See the "ACCEPT Statement" section of the *COBOL/400* Reference* for more information on specifying this statement. See the "Attribute Data Formats" section of the *COBOL/400* Reference* for information on the OPEN-FEEDBACK and the I-O-FEEDBACK areas.

Refer to the *Data Management Guide* for information on OPEN-FEEDBACK and I-O-FEEDBACK and the layout and description of the data areas contained in the feedback areas.

When the FILE STATUS clause is specified, the system moves a value into the status key data item after each input/output request that explicitly or implicitly refers to this file. This 2-character value indicates the run status of the statement. When the compiler generates code to block output records or unblock input records, file status values that are caused by OS/400 exceptions are set only when a block is processed. For more information about blocking records, refer to "Unblocking Input Records and Blocking Output Records" on page 100.

The I-O-FEEDBACK area is not updated after each read or write for files in which multiple records are blocked and unblocked by COBOL.

For database files, you may not see all changes as they occur, if the changes are made in different programs. For a description of the effect of blocking on

changes to database files, see the discussion on Sequential-Only Processing in the *Database Guide*.

File Descriptions

All files on the AS/400 system are defined to the OS/400 operating system. The extent to which files can be defined differs:

- A **program-described file** is described at the field level within the COBOL program in the Data Division. The description of the file to the operating system includes information about the type of file and the length of the records in the file.
- An **externally described file** is described at the field level to the operating system through IDDU, SQL/400* commands, or DDS. If you create a file (for instance, by using the CRTPF command) without specifying DDS for it, the file still has a field description. The single field has the same name as the file, and has the record length you specified in the create command.

The description includes information about the type of file, such as database or a device, and a description of each field and its attributes. The file must be created before you compile the program.

Both externally described files and program-described files must be defined in the COBOL program within the INPUT-OUTPUT SECTION and the FILE SECTION. Record descriptions in the FILE SECTION for externally described files can be defined with the Format 2 COPY statement.

Device-dependent functions such as forms control are not extracted by the Format 2 COPY operation. Only field-level descriptions are extracted.

When EXTERNALLY-DESCRIBED-KEY is specified as RECORD KEY, the fields that make up RECORD KEY are also extracted from DDS.

For more information on the Format 2 COPY statement, see Figure 37 on page 110 and the accompanying text.

Note: Actual file processing within the Procedure Division is the same, if the file is externally described or program-described.

Program-Described Files

Records and fields for a program-described file are described by coding record descriptions in the File Section of the COBOL program instead of using the Format 2 COPY statement.

The file must exist on the system before the program can run, except when you use dynamic file creation, by specifying GENOPT(*CRTF) on the CRTCLPGM command. For more information, refer to the description of the GENOPT parameter on page 22, or the OPEN statement in the *COBOL/400* Reference*. To create a file, use one of the Create File commands, which can be found in the *CL Reference*.

DDS can be used with the Create File commands. For a COBOL indexed file, a keyed access path must be created. Specify a key in DDS when the file is created. The record key in COBOL must match the key defined when the file was created.

Externally Described Files

Externally described files offer the following advantages over program-described files:

- Less coding in COBOL programs. If the same file is used by many programs, the fields can be defined once to the operating system, and then used by all the programs. This eliminates the need to code a separate record description for each program that uses the file.
- Less maintenance activity when the file's record format is changed. You can often update programs by changing the file's record format and then recompiling the programs that use the file without changing any coding in the program.
- Improved documentation. Programs using the same files use consistent record format and field names.
- Any editing to be processed on externally described output files can be specified in DDS.

The external description for a file includes:

- The record format specifications that contain a description of the fields in a record
- Access path specifications that describe how the records are to be retrieved.

These specifications come from the external file description and from the OS/400 command you use to create the file.

You can use an externally described file within a program by making it a program-described file (by coding the record description in the source). In this case, the compiler does not copy the external field-level description of the file at compilation time. You may find this useful during conversions, since an existing program can use a program-described file while a new program uses an externally described file to refer to the same file.

Figure 31 shows how COBOL programs can relate to files on the AS/400 system, making use of external file descriptions from DDS.

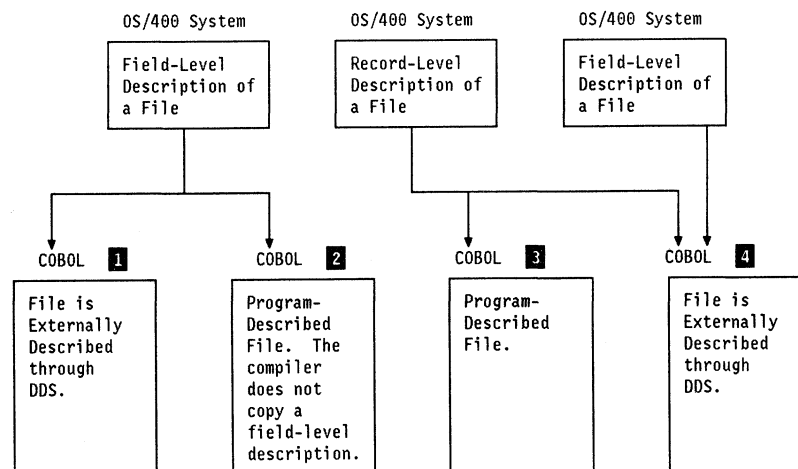


Figure 31. Example showing how COBOL can relate to AS/400 files

- 1** The COBOL program uses the field level description of a file that is defined to the operating system. The COBOL user coded a Format 2 COPY statement for the record description. At compilation time, the compiler copies in the external field-level description and translates it into a syntactically correct COBOL record description. The file must exist at compilation time.
- 2** An externally described file is used as a program-described file in the COBOL program. The entire record description for the file is coded in the COBOL program. This file does not have to exist at compilation time.
- 3** A file is described to the operating system as far as the record level only. The entire record description must be coded in the COBOL program. This file does not have to exist at compilation time.
- 4** A file name can be specified for compilation time, and a different file name can be specified for run time. A COBOL Format 2 COPY statement generates the record description for the file at compilation time. At run time, a different library list or a file override command can be used so that a different file is accessed by the program. The file description copied in at compilation time is used to describe the input records used at run time.

Note: For externally described files, the two file formats must be the same. Otherwise, a level check error will occur.

Data Description Specifications (DDS)

You can use Data Description Specifications (DDS) to describe files at the field level to the operating system. In DDS, each record format in an externally described file is identified by a unique record format name.

The record format specifications describe the fields in a record and the location of the fields in a record. The fields are located in the record in the order specified in DDS. The field description generally includes the field name, the field type (character, binary, external decimal, or internal decimal), and the field length (including the number of decimal positions in a numeric field). Instead of being specified in the record format for a physical or logical file, the field attributes can be defined in a field reference file. (See Figure 32 on page 105.)

The keys for a record format are specified in DDS. When you use a Format 2 COPY statement, a table of comments is generated in the source program listing showing how the keys for the format are defined in DDS.

In addition, DDS keywords can be used to:

- Specify edit codes for a field (EDTCDE)
- Specify edit words for a field (EDTWRD)
- Specify that duplicate key values are not allowed for the file (UNIQUE)
- Specify a text description for a record format or a field (TEXT).

See the *DDS Reference* for a complete list of the DDS keywords that are valid for a database file.

File		Keying Instruction		Graphic		Description		Page of	
Programmer	Date								

Sequence Number	Form Type	Firm Type	Amd/Chg/Comment	Conditioning				Name	Length	References	Line	Pos	Functions		
				Indicator	Ind. (N)	Ind. (N)	Ind. (N)								
A*													FLDREF	DSTREF	DISTRIBUTION APPLICATION FIELD REFERENCE
A*													R	DSTREF	TEXT('DISTRIBUTION FIELD REF')
A*													COMMON FIELDS USED AS REFERENCE		
1	A								6	0			EDTCDE(Y)		
A*													FIELDS USED BY CUSTOMER MASTER FILE		
2	A								5				CHECK(MF)		
A*													CUST		
A*													NAME		
A*													ADDR		
3	A												REFFLD(NAME)		
A*													CITY		
A*													STATE		
2	A								2				CHECK(MF)		
A*													SRHCOD		
A*													ZIP		
2	A								5	0			CHECK(MF)		
A*													CUSTYP		
4	A								1	0			RANGE(1 5)		
A*													ARBAL		
5	A								8	2			COLHGD('ACCTS REC' 'BALANCE')		
A*													DRDBAL		
6	A												REFFLD(ARBAL)		
A*													LSTAMT		
7	A												COLHGD('LAST' 'AMOUNT' 'PAID')		
A*													EDTCDE(J)		
A*													REFFLD(ARBAL)		
A*													COLHGD('A/R AMT IN' 'ORDER FILE')		
A*													REFFLD(ARBAL)		
A*													COLHGD('LAST' 'AMOUNT' 'PAID')		
A*													TEXT('LAST AMOUNT PAID IN A/R')		

Figure 32. Example of a Field Reference File

This example of a field reference file shows the definitions of the fields that are used by the CUSMSTL (customer master logical) file, which is shown in Figure 33 on page 107. The field reference file normally contains the definitions of fields that are used by other files. The following text describes some of the entries for this field reference file.

- 1 The BASDAT field is edited by the Y edit code, as indicated by the keyword EDTCDE (Y). If this field is used in an externally described output file for a COBOL program, the COBOL-generated field is compatible with the data type specified in the DDS. The field is edited when the record is written. When the field is used in a program-described output file, compatibility with the DDS fields in the file is the user's responsibility. When DDS is not used to create the file, appropriate editing of the field in the COBOL program is also the user's responsibility.
- 2 The CHECK(MF) entry specifies that the field is a mandatory fill field when it is entered from a display work station. Mandatory fill means that all characters for the field must be entered from the display work station.
- 3 The ADDR and CITY fields share the same attributes that are specified for the NAME field, as indicated by the REFFLD keyword.

- 4** The RANGE keyword, which is specified for the CUSTYP field, ensures that the only valid numbers that can be entered into this field from a display work station are 1 through 5.
- 5** The COLHDG keyword provides a column head for the field if it is used by the Application Development Tools (Appl Dev Tools).
- 6** The ARBAL field is edited by the J edit code, as indicated by the keyword EDTCDE(J).
- 7** A text description (TEXT keyword) is provided for some fields. The TEXT keyword is used for documentation purposes and appears in various listings.

COBOL Specifications for Files Described Externally Using DDS

You can incorporate the file description in your program by coding a Format 2 COPY statement. The information from the external description is then retrieved by the COBOL compiler, and a COBOL data structure is generated.

The following pages provide examples of DDS usage and the COBOL code that would result from the use of a Format 2 COPY statement. (See “Format 2 COPY Statement (DD, DDR, DDS, or DDSR Option)” on page 110 for a detailed description of the Format 2 COPY statement.)

- Figure 33 on page 107 shows the DDS for a logical file and Figure 34 on page 108 shows the COBOL code generated.
- Figure 35 on page 109 describes the same file but includes the ALIAS keyword, and Figure 36 on page 110 shows the COBOL code generated.

Actual file processing within the Procedure Division is the same for both program-described and externally described files.

File		Keying Instruction	Graphic							Description	Page	of
Programmer	Date		Key									

Sequence Number	Form Type A=Add/Chg/Comment D=Delete R=Record K=Key	Conditioning				Name	Length	Reference (R)	Data Type/Keyboard Shift B=Blank C=Character D=Decimal F=Float I=Integer N=Numeric S=Sign Z=Zero	Location		Functions
		Indicator	Not (N)	Indicator	Not (N)					Line	Pos	
1	A	LOGICAL				CUSMSTL						CUSTOMER MASTER FILE
												2 UNIQUE
												3 R CUSREC
												PFILE(CUSMSTP)
												TEXT('CUSTOMER MASTER RECORD')
						CUST						
						NAME						
						ADDR						
						CITY						
						STATE						
						ZIP						
						SRHCOB						
						CUSTYP						
						ARBAL						
						ORDBAL						
						LSTAMT						
						LSTDAT						
						CRDLMT						
						SLSYR						
						SLSLYR		5				
						4 K CUST						

Figure 33. Example of Data Description Specifications for a Logical File

- 1 A logical file for processing the customer master physical file (CUSMSTP) is defined and named CUSMSTL.
- 2 The UNIQUE keyword indicates that duplicate key values for this file are not allowed.
- 3 One record format (CUSREC) is defined for the CUSMSTL file, which is to be based upon the physical file CUSMSTP.
- 4 The CUST field is identified as the key field for this file.
- 5 If field attributes (such as length, data type, and decimal positions) are not specified in the DDS for a logical file, the attributes are obtained from the corresponding field in the physical file. Any field attributes specified in the DDS for the logical file override the attributes for the corresponding field in the physical file. The definition of the fields in the physical file could refer to a field reference file. A field reference file is a data description file consisting of field names and their definitions, such as size and type. When a field reference file is used, the same fields that are used in multiple record formats have to be defined only once in the field reference file. For more information on a field reference file, see the *Database Guide*.

Figure 32 on page 105 shows an example of a field reference file that defines the attributes of the fields used in the database file. See the *Database Guide* for more information regarding field reference files.

```

01 CUS-MASTER.
   COPY DDS-CUSREC OF CUSLIB-CUSTMAST.
*I-0 FORMAT: CUSREC FROM FILE CUSTMAST OF LIBRARY CUSLIB      CUSREC
*
*   CUSTOMER MASTER RECORD                                CUSREC
*THE KEY DEFINITIONS FOR THE RECORD FORMAT CUSREC           CUSREC
*NUMBER  NAME  RETRIEVAL  TYPE  ALTSEQ                    CUSREC
*0001   CUST  ASCENDING  AN    NO                        CUSREC
05 CUSREC.
06 CUST      PIC X(5).                                     CUSREC
*           CUSTOMER NUMBER                               CUSREC
06 NAME      PIC X(20).                                    CUSREC
*           CUSTOMER NAME                                 CUSREC
06 ADDR      PIC X(20).                                    CUSREC
*           CUSTOMER ADDRESS                             CUSREC
06 CITY      PIC X(20).                                    CUSREC
*           CUSTOMER CITY                                CUSREC
06 STATE     PIC X(2).                                     CUSREC
*           STATE ABBREVIATION                           CUSREC
06 ZIP       PIC S9(5) COMP-3.                             CUSREC
*           ZIP CODE                                      CUSREC
06 SHRCOD    PIC X(6).                                     CUSREC
*           CUSTOMER NAME SEARCH CODE                    CUSREC
06 CUSTYP    PIC 9(1).                                     CUSREC
*           CUSTOMER TYPE                                 CUSREC
06 ARBAL     PIC S9(6)V9(2) COMP-3.                       CUSREC
*           ACCT/REC BALANCE                              CUSREC

```

Figure 34. Example of the Results of the Format 2 COPY Statement (DDS)

File	Keying Instruction	Graphic		Description	Page	of
Programmer	Date	Key				

Sequence Number	Form Type AA/ZZ/Comment (A/D/*) Not 00	Conditioning	Condition Name	Name	Length	Reference (R)	Data Type/Keyboard Shift	Position	Usage (B/D/U/B/N/N/N/P)	Location		Functions
										Line	Pos	
A*			LOGICAL	CUSMSTL								CUSTOMER MASTER FILE
A*			R	CUSREC								UNIQUE
A*				CUST								PFILE(CUSMSTP)
A*				NAME								TEXT('CUSTOMER MASTER RECORD')
A*				ADDR								ALIAS(CUSTOMER_NUMBER)
A*				CITY								ALIAS(CUSTOMER_NAME)
A*				STATE								ALIAS(ADDRESS)
A*				ZIP								
A*				SRHCOD								ALIAS(SEARCH_CODE)
A*				CUSTYP								ALIAS(CUSTOMER_TYPE)
A*				ARBAL								ALIAS(ACCT_REC_BALANCE)
A*				ORBAL								
A*				LSTAMT								
A*				LSTDAT								
A*				CRDLMT								
A*				SLSYR								
A*				SLSLYR								
A*			K	CUST								

Figure 35. Example of Data Description Specifications with ALIAS

- 1 This is the name associated with the ALIAS keyword, which will be included in the program. Available through the DDS ALIAS option, an alias is an alternative name that allows a data name of up to 30 characters to be included in a COBOL/400 program.

```

01 CUS-MASTER.
   COPY DD-CUSREC OF CUSLIB-CUSTMAST.
*I-0 FORMAT: CUSREC FROM FILE CUSTMAST OF LIBRARY CUSLIB      CUSREC
*      CUSTOMER MASTER RECORD                                CUSREC
*THE KEY DEFINITIONS FOR THE RECORD FORMAT CUSREC            CUSREC
*NUMBER  NAME          RETRIEVAL TYPE          ALTSEQ        CUSREC
*0001   CUSTOMER-NUMBER ASCENDING AN          NO              CUSREC
                                         CUSREC
05 CUSREC.                                                  CUSREC
06 CUSTOMER-NUMBER PIC X(5).                                CUSREC
*      CUSTOMER NUMBER                                       CUSREC
06 CUSTOMER-NAME PIC X(20).                                CUSREC
*      CUSTOMER NAME                                         CUSREC
06 ADDRESS PIC X(20).                                      CUSREC
*      CUSTOMER ADDRESS                                       CUSREC
06 CITY PIC X(20).                                        CUSREC
*      CUSTOMER CITY                                          CUSREC
06 STATE PIC X(2).                                        CUSREC
*      STATE ABBREVIATION                                     CUSREC
06 ZIP PIC S9(5) COMP-3.                                   CUSREC
*      ZIP CODE                                              CUSREC
06 SEARCH-CODE PIC X(6).                                   CUSREC
*      CUSTOMER NAME SEARCH CODE                             CUSREC
06 CUSTOMER-TYPE PIC 9(1).                                 CUSREC
*      CUSTOMER TYPE                                         CUSREC
06 ACCT-REC-BALANCE PIC S9(6)V9(2) COMP-3.                CUSREC
*      ACCT/REC BALANCE                                       CUSREC

```

Figure 36. Example of the Results of the Format 2 COPY Statement (DD) with the ALIAS Keyword

IBM Extension

Format 2 COPY Statement (DD, DDR, DDS, or DDSR Option)

For general information about both formats of the COPY statement, see the *COBOL/400* Reference*.

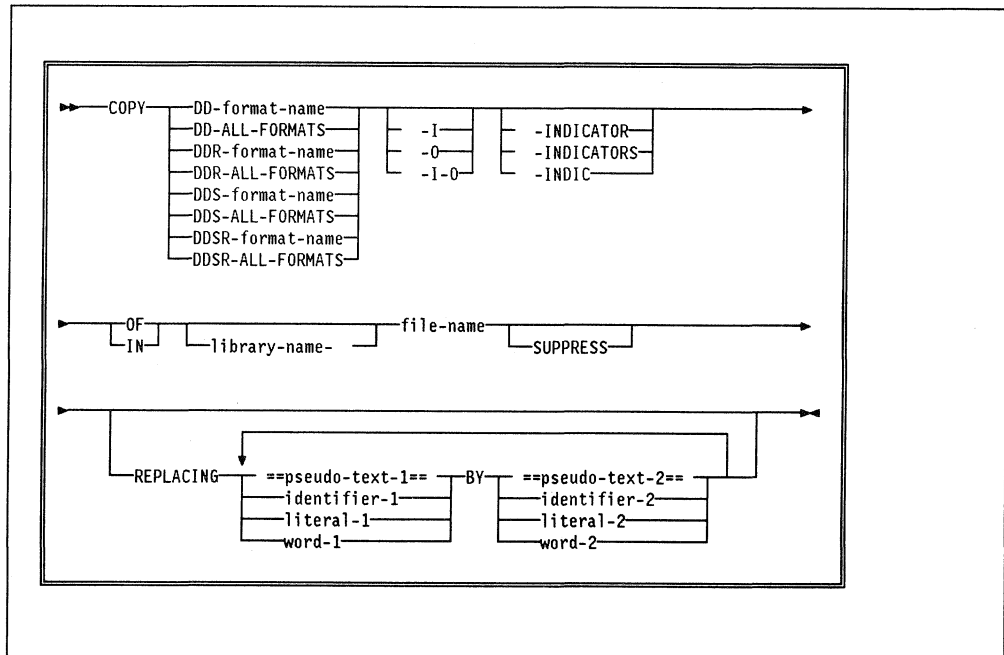


Figure 37. COPY Statement – Format 2 – DDS Translate

You can use the Format 2 COPY statement (DD, DDR, DDS, or DDSR option) to create COBOL Data Division statements to describe a file that exists on the

system. These descriptions are based on the version of the file in existence at compilation time. They do not make use of any DDS source statements for the file. Refer to the "COPY Statement" section of the *COBOL/400* Reference* for more information about the COPY statement.

Note: The Format 2 COPY statement (DD, DDR, DDS, or DDSR option) will be denoted by the term *Format 2 COPY statement* throughout this manual.

The Format 2 COPY statement can be used only in the Data Division. You should ensure that a group level item that has a level-number less than 05 precedes the statement.

The DD option is used to reference ALIAS (alternative) names. The specification of an ALIAS name in DDS allows a data name of up to 30 characters to be included in the COBOL program.

When the DD option is used, any ALIAS names present replace the corresponding DDS field names. All underscores in the ALIAS names are translated into hyphens before any replacing occurs.

The DDR option does everything that the DD option does. It also copies the internal DDS format field names, replacing the invalid COBOL characters @, #, \$, and _ with the valid COBOL characters A, N, D, and - accordingly. This option also removes any underscores from the ends of the field names.

The DDS option copies in the internal DDS format field names. For examples of keys and key names that can be generated when you use the DDS option of the Format 2 COPY statement, see pages 119 through 123.

The DDSR option does everything that the DDS option does. It also copies the internal DDS format field names, replacing the invalid COBOL characters @, #, \$, and _ with the valid COBOL characters A, N, D, and - accordingly. This option also removes any underscores from the ends of the field names.

The following shows the effect of the DDR or DDSR option on invalid COBOL field names:

Original Field Name	Modified Field Name
FLD_A	FLD-A
NUMBER#1	NUMBERN1
POINT@7	POINTA7
BALANCE\$	BALANCED

When the RECORD KEY clause specifies EXTERNALLY-DESCRIBED-KEY, a format can be copied only once under an FD. For example, if all of the formats of a file are copied under an FD, no other Format 2 COPY statement can be specified for the same file under that FD.

The format-name is the name of the DDS record format definition that is to be translated into COBOL data description entries. The format-name must follow the rules for formation of any COBOL/400 name.

If neither -I nor -O is specified, -I-O is assumed.

If format-name is specified without the Indicator attribute, and both -I and -O formats are to be generated, each record format is generated as a redefinition of an 05 elementary item defined as:

- The size of the largest record format that will be generated.

If ALL-FORMATS is specified (without the Indicator attribute) each record format is generated as a redefinition of an 05 elementary item defined as either:

- The size of the largest record format in the file, if the COPY statement appears in the File Section
- The size of the largest record format that will be generated, if the COPY statement appears outside of the File Section.

When the Indicator attribute is specified, no redefinition takes place. Instead, each of the formats generates a separate data structure.

More information can be found about the Indicator attribute in the section, "Indicator Attribute of the Format 2 COPY Statement" on page 116.

Library-name is optional. If it is not specified, the current job library list is used as the default value.

File-name is the name of an AS/400 system file. The generated DDS entries represent the record format defined in the file. The file must be created before the program is compiled.

If the file is a database file, a single I/O format is generated.

For all other file types, the description generated varies as follows:

- If -I is specified, the generated data description entries contain either:
 - The input and input/output fields for a nonsubfile format
 - The input, output, and input/output fields for a subfile format.
- If -O is specified, the generated data description entries contain either:
 - The output and input/output fields for a nonsubfile format
 - The input, output, and input/output fields for a subfile format.

Note: Subfile records with only output or input/output fields, and no field indicators specified, generate I/O formats.

If a separate storage area is needed in WORKING-STORAGE for each format, an individual COPY statement must be specified for each format.

For example, if you assume that the file CUSTMASTER contains two formats CUSADR and CUSTDETL, the following COPY statements could be specified:

```
SELECT FILE-X
ASSIGN TO DATABASE-CUSTMASTER.
:
FD FILE-X
  LABEL RECORDS ARE STANDARD.
01 FILE-X-RECS.
  COPY DDS-ALL-FORMATS OF
    CUSTMASTER-QGPL. (See Note 1.)
:
WORKING-STORAGE SECTION.
01 ADR-REC.
  COPY DDS-CUSTADR OF
    CUSTMASTER. (See Note 2.)
01 DETAIL-REC.
  COPY DDS-CUSTDETL OF
    CUSTMASTER. (See Note 2.)
```

Notes:

1. This COPY statement generates only one storage area for all formats.
2. These COPY statements generate separate storage areas.

Indicators

Indicators are Boolean data items that can have the values B"0" or B"1".

When you define a record format for a file using DDS, you can condition the options using indicators; indicators can also be used to reflect particular responses. These indicators are known as OPTION and RESPONSE, respectively. Option indicators provide options such as spacing, underlining, and allowing or requesting data transfer from a program to a printer or display device. Response indicators provide response information to a program from a device, such as function keys pressed by a work station user, and whether data has been entered.

Indicators can be used with TRANSACTION files and FORMATFILE files, but never with database files.

Data Structures Generated

Different DDS keywords influence the creation of various types of data structures.

Format (Record) Level Structures

At the beginning of each format, a table of comments is generated in the source program listing. These comments provide details of the files used during compilation of the program. If there are record keys for the file, comments are also generated to show how the keys are defined in DDS. The record key entries that may appear in the table and the table heading are listed below.

Heading	Possible Entry
NUMBER	key field number
NAME	key field name
RETRIEVAL	ASCENDING, DESCENDING
TYPE	ZONE, DIGIT, SIGNED, ABSVAL, AN (alphanumeric), N (numeric) J (DBCS item), DDS - L (date), DDS - T (time), DDS - Z (timestamp), DDS - G (fixed-length graphic), VARLEN (variable-length character or bracketed DBCS item), G VARLEN (variable-length DBCS-graphic)
ALTSEQ	NO, YES

If redefinition is required to allow for the generation of multiple formats, a group level name is generated as follows:

```
05 file-name-RECORD
   PIC X(size of largest record).
```

For each format, a group level name is assigned as follows:

- INPUT


```
05 format-name-I
```
- OUTPUT


```
05 format-name-O
```
- I/O Format


```
05 format-name
```

Data Field Structures

Field names, PICTURE definitions, and numeric usage clauses are derived directly from the internal DDS format field names (or ALIAS names in the case of the DD or DDR option) and data type representations. Field names and PICTURE definitions are constructed as follows:

```
06 field-name PIC
```

Note: See Figure 38 on page 115 for the appropriate COBOL PICTURE definition.

Table 3. Data Field Structures			
DDS		COBOL DATA DIVISION n=total field length (DDS pos. 30-34) m=number of decimals (DDS pos. 36 & 37)	
Data Type (pos.35)	Formats	If DDS pos. 36 & 37 are blank	If DDS pos. 36 & 37 are not blank
PHYSICAL, LOGICAL, PRINTER, AND COMMUNICATIONS FILES			
b(Blank)	Default	PIC X(n) ²	PIC S9(n-m)V9(m)
P	Packed decimal	PIC S9(n) COMP-3	PIC S9(n-m)V9(m) COMP-3
S	Zoned decimal/signed numeric	PIC S9(n)	PIC S9(n-m)V9(m)
B	Binary	PIC S9(n) COMP-4	PIC S9(n-m)V9(m) COMP-4
F	Floating point ¹		
	single precision	PIC 9(5) COMP-4	PIC 9(5) COMP-4
	double precision	PIC 9(10) COMP-4	PIC 9(10) COMP-4
A	Character	PIC X(n) ²	—
H	Hexadecimal data	PIC X(n)	—
L	Date ³	PIC X(n)	—
T	Time ³	PIC X(n)	—
Z	Timestamp ³	PIC X(n)	—
J	DBCS-Only data	PIC X(n)	—
E	DBCS-Either data	PIC X(n)	—
O	DBCS-Open data	PIC X(n)	—
G	DBCS-Graphic data	PIC X(2n) ²	—
DISPLAY FILES			
b(Blank)	Default	PIC X(n)	PIC S9(n-m)V9(m)
X	Alphabetic Only	PIC X(n)	—
N	Numeric Shift	PIC X(n)	PIC S9(n-m)V9(m)
Y	Numeric Only	—	PIC S9(n-m)V9(m)
I	Inhibit Keyboard entry	PIC X(n)	PIC S9(n-m)V9(m)
W	Katakana	PIC X(n)	—
A	Alphanumeric Shift	PIC X(n)	—
D	Digits only	PIC X(n)	PIC S9(n)
F	Floating point ¹		
	single precision	PIC 9(5) COMP-4	PIC 9(5) COMP-4
	double precision	PIC 9(10) COMP-4	PIC 9(10) COMP-4
M	Numeric-only character	PIC X(n)	—
S	Signed-numeric shift	—	PIC S9(n-m)V9(m)
E	DBCS-either	PIC X(n)	—
J	DBCS-only	PIC X(n)	—
O	DBCS-open	PIC X(n)	—
G	DBCS-graphic	PIC X(2n)	—
¹ COBOL treats floating point fields as FILLER. See 'Floating Point Fields'. ² In DDS, if the field has an attribute of VARLEN, the result is two additional bytes at the beginning of the field. ³ FILLER items by default. See 'Date, Time, and Timestamp Fields'.			

Figure 38. Data Field Structures

Indicator Structures

If indicators are requested, and exist in the format, an additional group name (06 level) is generated at the beginning of the structure, followed by entries (07 level) for the relevant individual indicators.

```
06 format-name(-I or -O)-INDIC.
07 INxx PIC 1 INDIC xx.
```

where xx is the indicator number.

For example:

```
06 SAMPLE1-I-INDIC.  
07 IN01 PIC 1 INDIC 01.  
07 IN04 PIC 1 INDIC 04.  
07 IN05 PIC 1 INDIC 05.  
07 IN07 PIC 1 INDIC 07.  
06 FLD1 PIC ... .  
06 FLD2 PIC ... .
```

Indicator Attribute of the Format 2 COPY Statement

The Indicator attribute specifies if data description entries are generated for indicators.

If the Indicator attribute is specified, data description entries are generated for indicators, but not for data fields. A 05 group level entry is generated as follows:

- If the COPY is for a single structure (for example, COPY DDS-format-name-INDIC)
05 format-name-I. (or -O as appropriate)
- If the COPY is for multiple structures
(for example, COPY DDS-ALL-FORMATS-INDIC)
05 file-name-RECORD.

The data description entries that are generated are determined by which one of the usage attributes (I, O, or I-O) is specified or assumed in the COPY statement.

- If ...I-INDICATOR... is specified, data description entries for input (response) indicators are generated for indicators used in the input record area.
- If ...O-INDICATOR... is specified, data description entries for output (option) indicators are generated for indicators used in the output record area.
- If ...I-O-INDICATOR... is specified or assumed, separate data description entries for both input and output (response and option) indicators are generated for indicators used in the input and output record areas.

If the Indicator attribute is not specified, generation of data description entries for indicators depends on if the file had the keyword INDARA specified in the DDS at the time it was created.

- If INDARA was not specified, data description entries are generated for both data fields and indicators.
- If INDARA was specified, data description entries are generated for data fields only, not for indicators.

Generation of I/O Formats

When all field descriptions are identical, and you have requested INPUT or OUTPUT fields implicitly or explicitly, only one set of field descriptions is generated. This type of description is annotated with a comment line reading, "I-O FORMAT: format-name". Neither -I nor -O is appended to the record format name.

Note: This always happens for database files because all field descriptions within a database file are identical.

For example:

```
01 RCUSREC.
  COPY DDS-CUSREC-I OF CUSFILE.
* I-O FORMAT: CUSREC FROM FILE CUSFILE OF LIBRARY CUSLIB      CUSREC
* THE KEY DEFINITIONS FOR RECORD FORMAT CUSREC
* NUMBER NAME RETRIEVAL TYPE ALTSEQ
* 0001 ARBAL ASCENDING SIGNED NO
* 0002 AREACD DESCENDING ABSVAL NO
  05 CUSREC.
  06 ARBAL          PIC S9(7)V9(2)      COMP-3      CUSREC
  06 AREACD        PIC S9(3)           COMP-3.     CUSREC
  06 BOSTAZ        PIC X(1).           CUSREC
  06 CNTCT         PIC X(15).          CUSREC
  06 CRCHKZ        PIC S9(2).          CUSREC
  06 CSTAT         PIC X(1).           CUSREC
  06 CUSTNZ        PIC S9(6).          CUSREC
  06 DLORD         PIC S9(6).          CUSREC
  06 DSCPCZ        PIC S9(2)V9(3)      COMP-3.     CUSREC
  06 INDUS        PIC S9(2).          CUSREC
  06 NAME1         PIC X(25).          CUSREC
  06 NAME2         PIC X(25).          CUSREC
  06 NAME3         PIC X(25).          CUSREC
  06 NAME4         PIC X(25).          CUSREC
  06 PHONE        PIC S9(7)           COMP-3.     CUSREC
  06 PRICIZ        PIC S9(2).          CUSREC
  06 SHPINZ        PIC X(25).          CUSREC
  06 SLSMAZ        PIC X(3).           CUSREC
  06 TAXCDZ        PIC S9(2).          CUSREC
  06 TERMSZ        PIC S9(2).          CUSREC
```

Figure 39. Example of Copy DDS Showing I/O Formats

Redefinition of Formats

Pay particular attention to the REDEFINES clause that may be generated for the ALL-FORMATS or -I-O phrases. Because all formats are redefined on the same area (generally a buffer area), several field names can describe the same area of storage, and unpredictable results can occur if the entire format area is not reinitialized prior to each output operation.

Data items that are subordinate to the data item specified in a MOVE CORRESPONDING statement do not correspond and are not moved when they contain a REDEFINES clause or are subordinate to a redefining item.

To avoid reinitialization, multiple Format 2 COPY statements using -I and -O suffixes can be used to create separate areas of storage in the Working-Storage section for each format or format type (input or output). READ INTO and WRITE FROM statements can be used with these record formats.

For example:

```
FD ORDER-ENTRY-SCREEN ...
01 ORDER-ENTRY-RECORD ...
:
WORKING-STORAGE SECTION.
01 ORDSFL-I-FORMAT.
   COPY DDS-ORDSFL-I OF DOESCR.
01 ORDSFL-O-FORMAT.
   COPY DDS-ORDSFL-O OF DOESCR.
:
PROCEDURE DIVISION.
:
READ SUBFILE ORDER-ENTRY-SCREEN NEXT MODIFIED RECORD
   INTO ORDSFL-I-FORMAT FORMAT IS "ORDSFL"
   AT END SET NO-MODIFIED-SUBFILE-RCD TO TRUE.
:
MOVE CORR ORDSFL-I TO ORDSFL-O.
REWRITE SUBFILE ORDER-ENTRY-RECORD FROM ORDSFL-O-FORMAT
   FORMAT IS "ORDSFL" ...
:
```

Key Generation Examples



AS/400 DATA DESCRIPTION SPECIFICATIONS

GX21-9891-0 LM/050+
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic		Description	Page	of
Programmer	Date		Key				
A							
Sequence Number	Form Type	Conditioning			Location	Line	Pos
		Condition Name					
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80
PHYSICAL FILE PF1 FOR KEY GENERATION EXAMPLES							
R PFRECORD							
MTH 2							
DAY 2							
YEAR 4							
ITEM 8							
K MTH							
K DAY							

Figure 40. Data Description Specifications for a Physical File

The physical file described by Figure 40 forms a basis for the examples that follow. Each example refers to a logical file (derived from the physical file) that specifies EXTERNALLY-DESCRIBED-KEY in its SELECT clause.

Example Using CONCAT Keyword

File		Keying Instruction	Graphic						Description		Page		of
Programmer	Date		Key										
A	Sequence Number	Conditioning	Name	Length	Location	Functions							
	1	Condition Name					21	22	23	24	25	26	27
2	Indicator	Indicator											
3	Not (N)	Not (N)											
4	Indicator	Indicator											
5	Not (N)	Not (N)											
6	Indicator	Indicator											
7	Not (N)	Not (N)											
8	Type of Name of Spec. (W/R/H/J/K/S/D)	Reserve											
9	Comment (A/D/O)												
10	Line	Post											
11	Line	Post											
12	Line	Post											
13	Line	Post											
14	Line	Post											
15	Line	Post											
16	Line	Post											
17	Line	Post											
18	Line	Post											
19	Line	Post											
20	Line	Post											
21	Line	Post											
22	Line	Post											
23	Line	Post											
24	Line	Post											
25	Line	Post											
26	Line	Post											
27	Line	Post											
28	Line	Post											
29	Line	Post											
30	Line	Post											
31	Line	Post											
32	Line	Post											
33	Line	Post											
34	Line	Post											
35	Line	Post											
36	Line	Post											
37	Line	Post											
38	Line	Post											
39	Line	Post											
40	Line	Post											
41	Line	Post											
42	Line	Post											
43	Line	Post											
44	Line	Post											
45	Line	Post											
46	Line	Post											
47	Line	Post											
48	Line	Post											
49	Line	Post											
50	Line	Post											
51	Line	Post											
52	Line	Post											
53	Line	Post											
54	Line	Post											
55	Line	Post											
56	Line	Post											
57	Line	Post											
58	Line	Post											
59	Line	Post											
60	Line	Post											
61	Line	Post											
62	Line	Post											
63	Line	Post											
64	Line	Post											
65	Line	Post											
66	Line	Post											
67	Line	Post											
68	Line	Post											
69	Line	Post											
70	Line	Post											
71	Line	Post											
72	Line	Post											
73	Line	Post											
74	Line	Post											
75	Line	Post											
76	Line	Post											
77	Line	Post											
78	Line	Post											
79	Line	Post											
80	Line	Post											
81	Line	Post											
82	Line	Post											
83	Line	Post											
84	Line	Post											
85	Line	Post											
86	Line	Post											
87	Line	Post											
88	Line	Post											
89	Line	Post											
90	Line	Post											
91	Line	Post											
92	Line	Post											
93	Line	Post											
94	Line	Post											
95	Line	Post											
96	Line	Post											
97	Line	Post											
98	Line	Post											
99	Line	Post											
100	Line	Post											

Figure 41. Data Description Specifications Using the CONCAT Keyword

For the logical file described by Figure 41, COPY DDS generates keys and key names derived from the physical file:

```

FD LF1 LABEL RECORDS ARE STANDARD.
01 LOG-RECORD.
    COPY DDS-ALL-FORMATS OF LF1.
    05 LF1-RECORD PIC X(8).
*   I-O FORMAT:RECORD1 FROM FILE LF1 OF LIBRARY COPYDDS
*
*THE KEY DEFINITIONS FOR RECORD FORMAT RECORD1
* NUMBER NAME RETRIEVAL TYPE ALTSEQ
* 0001 MTH-DDS ASCENDING AN NO
* KEY NAME ORIGINATES FROM PHYSICAL FILE
* 0002 DAY-DDS-DDS ASCENDING AN NO
* KEY NAME ORIGINATES FROM PHYSICAL FILE
    05 RECORD1 REDEFINES LF1-RECORD.
    06 DATE-DDS PIC X(8).
    06 FILLER REDEFINES DATE-DDS.
    07 MTH-DDS PIC X(2).
    07 DAY-DDS-DDS PIC X(2).
    07 FILLER PIC X(4).

```

Figure 42. Example Using the CONCAT Keyword

The COPY statement adds the suffix -DDS to the field names MTH and DATE because MTH is a key that originates from the physical file, and DATE is a COBOL reserved word. The COPY statement adds the suffix -DDS twice to the field name DAY because DAY is both a key that originates from the physical file and a COBOL reserved word.

Note that if you move your COPY statement from the File Section to the Working-Storage Section or to the Linkage Section, the fields subordinate to DATE-DDS are no longer available:

```

WORKING-STORAGE SECTION.
01 WRK-RECORD.
    COPY DDS-ALL-FORMATS OF LF1.
    05 LF1-RECORD PIC X(8).
*   I-O FORMAT:RECORD1 FROM FILE LF1 OF LIBRARY COPYDDS
*
    05 RECORD1 REDEFINES LF1-RECORD.
    06 DATE-DDS PIC X(8).

```

Figure 43. Example Using the CONCAT Keyword-- Working-Storage Section

Example Using RENAME Keyword

File		Keying Instruction		Graphic		Description		Page		of	
Programmer		Date		Key							
A	Conditioning	Condition Name	Name	Length	Location	Functions					
Sequence Number	Form Type	Indicator	Indicator	Indicator	Indicator	Indicator	Indicator	Indicator	Indicator	Indicator	Indicator
1	2	3	4	5	6	7	8	9	10	11	12
1	A*										
2	A*		R RECORD2								
3	A*		MONTH								
4	A*		K MTH								
5	A*										
6	A*										
7	A*										
8	A*										
9	A*										
10	A*										
11	A*										
12	A*										
13	A*										
14	A*										
15	A*										
16	A*										
17	A*										
18	A*										
19	A*										
20	A*										
21	A*										
22	A*										
23	A*										
24	A*										
25	A*										
26	A*										
27	A*										
28	A*										
29	A*										
30	A*										
31	A*										
32	A*										
33	A*										
34	A*										
35	A*										
36	A*										
37	A*										
38	A*										
39	A*										
40	A*										
41	A*										
42	A*										
43	A*										
44	A*										
45	A*										
46	A*										
47	A*										
48	A*										
49	A*										
50	A*										
51	A*										
52	A*										
53	A*										
54	A*										
55	A*										
56	A*										
57	A*										
58	A*										
59	A*										
60	A*										
61	A*										
62	A*										
63	A*										
64	A*										
65	A*										
66	A*										
67	A*										
68	A*										
69	A*										
70	A*										
71	A*										
72	A*										
73	A*										
74	A*										
75	A*										
76	A*										
77	A*										
78	A*										
79	A*										
80	A*										
81	A*										
82	A*										

Figure 44. Data Description Specifications Using the RENAME Keyword

For the logical file described by Figure 44, COPY DDS generates a key and key name derived from the physical file:

```

FD LF2 LABEL RECORDS ARE STANDARD.
01 LOG-RECORD.
    COPY DDS-ALL-FORMATS OF LF2.
    05 LF2-RECORD PIC X(2).
* I-O FORMAT:RECORD2 FROM FILE LF2 OF LIBRARY COPYDDS
*
*THE KEY DEFINITIONS FOR RECORD FORMAT RECORD2
* NUMBER NAME RETRIEVAL TYPE ALTSEQ
* 0001 MTH-DDS ASCENDING AN NO
* KEY NAME ORIGINATES FROM PHYSICAL FILE
05 RECORD2 REDEFINES LF2-RECORD.
06 MONTH PIC X(2).
06 MTH-DDS REDEFINES MONTH PIC X(2).
    
```

Figure 45. Using the RENAME Keyword

The COPY statement adds the suffix -DDS to the field name MTH because MTH is a key that originates from the physical file.

Example Using SST Keyword

File		Keying Instruction		Graphic		Description		Page of	
Programmer	Date			Key					
A									
Sequence Number	Form Type	Conditioning			Name	Length	Location		Functions
		Indicator	Indicator	Indicator			Line	Pos	
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
LOGICAL FILE LF3 FOR SST KEYWORD EXAMPLES									
					R	RECORD3			PFILE(PF1)
					YY			I	SST(YEAR 2 2)
					K	YY			

Figure 46. Data Description Specifications Using the SST Keyword

For the logical file described by Figure 46, COPY DDS generates the following specifications:

```

FD LF3 LABEL RECORDS ARE STANDARD.
01 LOG-RECORD.
   COPY DDS-ALL-FORMATS OF LF3.
   05 LF3-RECORD PIC X(2).
*   I-O FORMAT:RECORD3 FROM FILE LF3 OF LIBRARY COPYDDS
*
*THE KEY DEFINITIONS FOR RECORD FORMAT RECORD3
* NUMBER NAME RETRIEVAL TYPE ALTSEQ
* 0001 YY ASCENDING AN NO
   05 RECORD3 REDEFINES LF3-RECORD.
   06 YY PIC X(2).
    
```

Figure 47. Using the SST Keyword

The COPY statement does not add a suffix to the field name YY because YY is neither a key that originates from the physical file nor a COBOL reserved word.

Additional Notes on Field and Format Names

If the generated field name is a COBOL reserved word, the suffix -DDS is added to the field name.

The REPLACING phrase cannot be used to change the name of a key field when EXTERNALLY-DESCRIBED-KEY is used.

Floating-Point Fields

COBOL treats floating-point fields as FILLER. The fields can contain floating-point values set outside of COBOL. A COMP-4 definition is generated to maintain proper alignment in the record, but the data is *not* in binary format. No attempt must be made to use floating-point data for processing in the COBOL program.

Floating-point key fields are not allowed. In cases where some formats exist with a floating-point key field and other formats do not, use one or more Format 2 COPY statements with specific format names, rather than the ALL-FORMATS option.

Note: If you have not specified your own program collating sequence, you can create a record containing floating-point fields in your COBOL program by moving LOW-VALUES to the entire record before moving in the values of the non-floating-point fields. This will give the floating-point fields in the record a value of zero. Note that the above method is only recommended if valid floating-point fields with a value of zero are desirable for your particular application.

REPLACING Phrase in Format 2 COPY Statement

The REPLACING phrase can be used to replace any of the generated COBOL source, including the level numbers and the format-name. Note the following exception:

- When RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified, the REPLACING phrase cannot change the name of a field that is a key.
For example:

```
5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B.+...2....+...3....+...4....+...5....+...6....+...7..IDENTFCN S COPYNAME  CHG DATE
001500*
001600* COPY DDS W I T H O U T REPLACING OPTION
001700*
14 001800 COPY DDS-CUSMST OF CUSMSTP.
+000001* I-0 FORMAT:CUSMST FROM FILE CUSMSTP OF LIBRARY COBNATEX CUSMST
+000002* CUSTOMER MASTER RECORD CUSMST
15 +000003 05 CUSMST. CUSMST
16 +000004 06 CUST PIC X(5). CUSMST
+000005* CUSTOMER NUMBER CUSMST
17 +000006 06 NAME PIC X(25). CUSMST
+000007* CUSTOMER NAME CUSMST
18 +000008 06 ADDR PIC X(20). CUSMST
+000009* CUSTOMER ADDRESS CUSMST
19 +000010 06 CITY PIC X(20). CUSMST
+000011* CUSTOMER CITY CUSMST
20 +000012 06 STATE PIC X(2). CUSMST
+000013* STATE CUSMST
21 +000014 06 ZIP PIC S9(5) COMP-3. CUSMST
+000015* ZIP CODE CUSMST
```

Figure 48. COPY DDS without the REPLACING Option

5738CB1 V2R2M0		AS/400 COBOL Source						
STMT	SEQNBR	-A 1 B...	2...3...4...5...6...7...	IDENTFCN	S	COPYNAME	CHG DATE	
	001900*						03/25/89	
	002000*	COPY DDS WITH REPLACING OPTION						03/25/89
	002100*						03/25/89	
31	002200	COPY	DDS-CUSMST	OF	CUSMSTP		03/25/89	
32	002300	REPLACING	NAME BY	ADDR-LINE-1			03/25/89	
33	002400	ADDR BY	ADDR-LINE-2				03/25/89	
34	002500	CITY BY	ADDR-LINE-3.				03/25/89	
	+000001*	I-O	FORMAT:CUSMST	FROM	FILE CUSMSTP	OF	LIBRARY COBNATEX	
	+000002*				CUSTOMER	MASTER	RECORD	
35	+000003	05	CUSMST.				CUSMST	
36	+000004	06	CUST		PIC X(5).		CUSMST	
	+000005*				CUSTOMER	NUMBER	CUSMST	
37	+000006	06	ADDR-LINE-1		PIC X(25).		CUSMST	
	+000007*				CUSTOMER	NAME	CUSMST	
38	+000008	06	ADDR-LINE-2		PIC X(20).		CUSMST	
	+000009*				CUSTOMER	ADDRESS	CUSMST	
39	+000010	06	ADDR-LINE-3		PIC X(20).		CUSMST	
	+000011*				CUSTOMER	CITY	CUSMST	
40	+000012	06	STATE		PIC X(2).		CUSMST	
	+000013*				STATE		CUSMST	
41	+000014	06	ZIP		PIC S9(5)	COMP-3.	CUSMST	
	+000015*				ZIP	CODE	CUSMST	

Figure 49. COPY DDS with the REPLACING Option

End of IBM Extension

Access Path

The description of an externally described file contains the access path that describes how records are to be retrieved from the file. Records can be retrieved based on an arrival sequence (nonkeyed) access path or on a keyed sequence access path.

The arrival sequence access path is based on the order in which the records are stored in the file. Records are added only to the end of the file.

For the keyed sequence access path, the sequence in which records are retrieved from the file is based on the contents of the key fields defined in the DDS for the file. For example, in the DDS shown in Figure 33 on page 107, CUST is defined as the key field. The keyed sequence access path is updated whenever records are added, deleted, or when the contents of a key field change.

See the *Database Guide* for a complete description of the access paths for an externally described database file.

Record Keys and Common Keys

For a keyed sequence access path, one or more fields can be defined in the DDS to be used as the key fields for a record format. All record types in a file do not have to have the same key fields. For example, an order header record can have the ORDER field defined as the key field, and the order detail records can have the ORDER and LINE fields defined as the key fields.

The key for a file is determined by the valid keys for the record types in that file. The file's key is determined in the following manner:

- If all record types in a file have the same number of key fields defined in DDS that are identical in attributes, the *key for the file* consists of all fields in

the key for the record types. (The corresponding fields do not have to have the same name.) For example, if the file has three record types and the key for each record type consists of fields A, B, and C, the file's key consists of fields A, B, and C. That is, the file's key is the same as the records' key.

- If all record types in the file do not have the same key fields, the key for the file consists of the key fields *common* to all record types. For example, a file has three record types and the key fields are defined as follows:
 - REC1 contains key field A.
 - REC2 contains key fields A and B.
 - REC3 contains key fields A, B, and C.

Then the file's key is field A, the key field common to all record types.

- If no key field is common to all record types, any keyed reference to the file will always return the first record in the file.

In COBOL, you must specify a RECORD KEY for an indexed file to identify the record you want to process. COBOL compares the key value with the key of the file or record, and processes the specified operation on the record whose key matches the RECORD KEY value.

When RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified:

- If the FORMAT phrase is specified, the compiler builds the search argument from the key fields in the record area for the specified format
- If the FORMAT phrase is not specified, the compiler builds the search argument from the key fields in the record area for the first record format defined in the program for that file.

Note: For a file containing multiple key fields to be processed in COBOL, the key fields must be contiguous in the record format used by the COBOL program, except when RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified.

Overriding or Adding COBOL Functions to the External Description

In addition to placing the external file description in the program through the use of the Format 2 COPY statement, you can also use standard record definition and redefinition to describe external files or to provide a group definition for a series of fields. It is the programmer's responsibility to ensure that program-described definitions are compatible with the external definitions of the file.

Level Checking

When a COBOL/400 program uses an externally described file, the operating system provides a level check function (LVLCHK). This function ensures that the format has not changed since compilation time.

The compiler always provides the information required by level checking when an externally described file is used (that is, when a record description was defined for the file by using the Format 2 COPY statement). Only those formats that were copied by the Format 2 COPY statement under the FD for a file are level checked. The level check function will be initiated at run time based on the selection made on the create, change, or override file commands. The default on the create file command is to request level checking. If level checking was requested, level checking occurs on a record format basis when the file is opened. If a level check error occurs, COBOL sets a file status of 39 at OPEN time.

When no level checking was requested, and the file is re-created using an existing format, existing COBOL programs that use that format may not work without recompilation, depending on the changes to the format. For instance,

- A change of keys will certainly cause a failure of the program on any I/O statement
- A change in the record length will cause any REWRITE to fail
- A change in the record layout can cause various errors in the processing of such a record.

You should use extreme caution when using COBOL programs without level checking or recompiling the programs.

Note: The compiler does not provide level checking for program-described files.

For more information on level checking, see the *Data Management Guide*.

Declaring Data Items Using CVTOPT Data Types

The COBOL/400 compiler allows you to convert variable-length fields from externally described files and SAA database data types to standard COBOL data items. The SAA data types you can convert are date, time, timestamp, and DBCS-graphic. COBOL/400 provides limited support for these data types.

Variable-length Fields

You can bring a variable-length field into your program if you specify *VARCHAR on the CVTOPT parameter of the CRTCLPGM command, or the VARCHAR option of the PROCESS statement. When *VARCHAR is specified, your COBOL/400 program will convert a variable-length field from an externally described file into a COBOL/400 group item.

An example of such a group item is:

```
06 ITEM1.  
   49 ITEM1-LENGTH    PIC S9(4) COMP-4.  
   49 ITEM1-DATA      PIC X(n).
```

where n represents the maximum length of the variable-length field. Within the program, the PIC S9(4) COMP-4 is treated like any other declaration of this type, and the PIC X(n) is treated as standard alphanumeric.

Since the maximum value that ITEM1-LENGTH can hold is 9 999, this is the length of the longest variable-length field you can write from a COBOL program.

When *VARCHAR is not specified, variable-length fields are ignored and declared as FILLER fields in COBOL/400 programs. If *NOVARCHAR is specified, the item is declared as follows:

```
06 FILLER    PIC x(n+2).
```

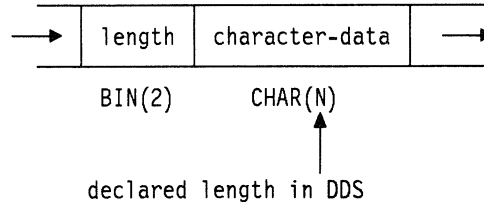
For syntax information, see the CVTOPT parameter on page 23.

Your program can perform any valid character operations on the generated data portion; however, because of the structure of the field, the length portion must be valid binary data. This data is not valid if it is negative, or greater than the maximum field length.

If the first two bytes of the field do not contain a valid binary number, an error will occur if you try to WRITE or REWRITE a record containing the field (or UPDATE or PUT the field in a database), and file status 90 is returned.

The following conditions apply when you specify variable-length fields:

- If a variable-length field is encountered when a field is extracted for an externally described file or an externally described data structure, it is declared in a COBOL/400 program as a fixed-length character field.
- For single-byte character fields, the length of the declared COBOL/400 field is the length of the DDS field plus 2 bytes.
- For DBCS-graphic data fields, the length of the declared COBOL/400 field is two times the length of the DDS field plus 2 bytes. For more information on graphic data types, see "DBCS-Graphic Fields" on page 129. The two extra bytes in the COBOL/400 field contain a binary number that represents the current length of the variable-length field. Figure 50 shows the COBOL/400 field length of variable-length fields.



For single-byte character fields: $2 + N = \text{COBOL/400 field length}$

For DBCS-graphic data type fields: $2 + 2(N) = \text{COBOL/400 field length}$

Figure 50. COBOL/400 Field Length of a Variable-Length Field

- Your COBOL/400 program can perform any valid character calculation operations on the declared fixed-length field. However, because of the structure of the field, the first two bytes of the field must contain valid binary data (invalid current field-length data is non-numeric, less than 0, or greater than the DDS field length.) An error occurs for an input or output operation if the first two bytes of the field contain invalid field-length data; file status 90 is returned.
- If you do not specify *VARCHAR, you can encounter problems performing WRITE operations on variable-length fields, because you cannot assign a value to FILLER. The two-byte field may have a value (for example X'4040') which gives a length beyond the range allowed for the field. This causes an I/O error.

To see an example of a program using variable-length fields, refer to "Examples:" on page 131.

Date, Time, and Timestamp Fields

Date, time, and timestamp fields are brought into your program only if you specify the *DATETIME option of the CRTCLPGM CVTOPT parameter, or the DATETIME option of the PROCESS statement. For a description and the syntax of the CVTOPT parameter, see page 23. If *DATETIME is not specified, date, time, and timestamp fields are ignored and are declared as FILLER fields in your COBOL/400 program.

Date, time or timestamp fields are brought into a COBOL/400 program as fixed-length character fields. Your COBOL/400 program can perform any valid character operations on the fixed-length fields. These operations will follow the standard COBOL rules for alphanumeric data items.

The date, time, and timestamp data types each have their own format.

If a field containing date, time, or timestamp information is updated by your program, and the updated information is to be passed back to your database, the format of the field must be exactly the same as it was when the field was retrieved from the database. If you do not use the same format, an error will occur. For information on valid formats for each data type, see the *DDS Reference*.

If you try to WRITE a record before moving an appropriate value to a date, time, or timestamp field, the WRITE operation will fail, and file status 90 will be returned.

If you declare date, time or timestamp items in your program as FILLER, do not attempt to WRITE records containing these fields, since you will not be able to set them to values that will be accepted by the system.

Null-capable Fields

Although your program can process null-capable fields, null values are not supported. READ, SORT, and MERGE operations can be performed on null-capable fields, but if the fields actually contain null values, errors occur.

DBCS-Graphic Fields

The DBCS-graphic data type is a character string in which each character is represented by 2 bytes. The DBCS-graphic data type does not contain shift-out (SO) or shift-in (SI) characters. The difference between single-byte and DBCS-graphic data is shown in the following figure:

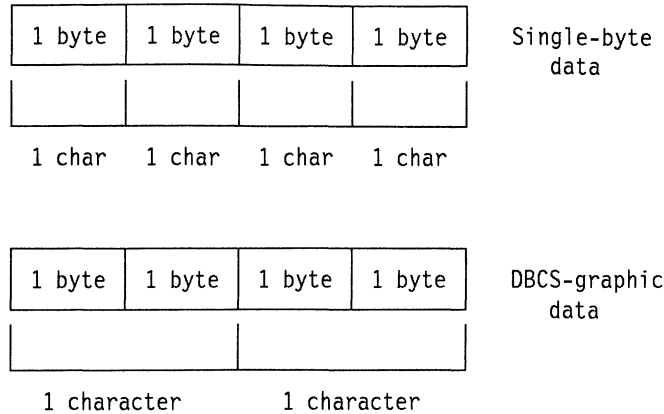


Figure 51. Comparing Single-byte and Graphic Data

DBCS-graphic data is brought into your COBOL/400 program only if you specify the *GRAPHIC value on the CVTOPT parameter of the CRTCLPGM command, or the CVTGRAPHIC option of the PROCESS statement. If you do not specify DBCS-graphic data, graphic data is ignored and declared as FILLER fields in your COBOL/400 program. For a description and the syntax of the CVTOPT parameter, see page 23.

The following conditions apply when DBCS-graphic data is specified:

- DBCS-graphic data is copied into a COBOL/400 program as a fixed-length alphanumeric field.
- Every DBCS-graphic data *character* has a length of 2 bytes.
- Every fixed-length DBCS-graphic data *field* has a length of 2 bytes times the number of characters in the field. For a description of the field length of variable-length graphic data fields, see “Variable-length Fields” on page 127.
- Your COBOL/400 program can perform any valid character operations on the fixed-length fields.

Variable-length DBCS-Graphic Fields

You can use variable-length fields in combination with DBCS-graphic data types, to specify variable-length DBCS-graphic data. To specify variable-length DBCS-graphic data, specify *VARCHAR and *GRAPHIC for the CVTOPT parameter of the CRTCLPGM command, or the VARCHAR and CVTGRAPHIC options for the PROCESS statement.

If you specify either of the following: CVTOPT(*NOVARCHAR *NOGRAPHIC) or CVTOPT(*NOVARCHAR *GRAPHIC) and the compiler encounters a variable-length DBCS-graphic data item, the resulting program contains the following:

```
*          06 FILLER          PIC X(2n+2).
          (Variable-length field)
```

where n is the number of characters in the DDS field.

If you specify CVTOPT(*VARCHAR *NOGRAPHIC), and the compiler encounters a variable-length DBCS-graphic data item, the resulting program contains the following:

```

06 NAME
*      (Variable-length field)
49 NAME-LENGTH      PIC S9(4) COMP-4.
*      (Number of 2-byte characters)
49 FILLER           PIC X(2n).
*      (Graphic field)

```

where n is the number of characters in the DDS field.

If you specify CVTOPT(*VARCHAR *GRAPHIC), and the compiler encounters a variable-length DBCS-graphic data item, the resulting program contains the following:

```

06 NAME
*      (Variable-length field)
49 NAME-LENGTH      PIC S9(4) COMP-4.
*      (Number of 2-byte characters)
49 NAME-DATA        PIC X(2n).
*      (Graphic field)

```

where n is the number of characters in the DDS field.

Examples:

Figure 52 shows an example of a DDS file that defines a variable-length DBCS-graphic data item. Figure 53 on page 132 shows the COBOL/400 program using a COPY DDS statement, and the resulting listing when the program is compiled.

```

A      R SAMPLEFILE
A*
A      VARITEM      100      VARLEN
A*
A      TIMEITEM     T       TIMFMT(*HMS)
A      DATEITEM     L       DATFMT(*YMD)
A      TIMESTAMP    Z
A*
A      GRAPHITEM    100G
A      VGRAPHITEM   100G      VARLEN

```

Figure 52. DDS File Defining a Variable-Length Graphic Data Field

```

5738C01 V2R2M0 001000      IBM SAA COBOL/400      TESTER/PGM1      AS400SYS 04/24/92 08:55:54      Page 1
Program . . . . . : PGM1
Library . . . . . : TESTER
Source file . . . . . : QLBSLRC
Library . . . . . : TESTER
Source member . . . . . : PGM1 04/24/92 08:23:06
Generation severity level . . . . . : 29
Text 'description' . . . . . : Data types example
Source listing options . . . . . : *NONE
Generation options . . . . . : *NONE
Conversion options . . . . . : *VARCHAR *DATETIME *GRAPHIC
Message limit:
  Number of messages . . . . . : *NOMAX
  Message limit severity . . . . . : 29
Print file . . . . . : QSYSPT
Library . . . . . : *LIBL
FIPS flagging . . . . . : *NOFIPS *NOSEG *NODEB *NOOBSOLETE
SAA flagging . . . . . : *NOFLAG
Extended display options . . . . . :
Flagging severity . . . . . : 0
Replace program . . . . . : *YES
Target release . . . . . : *CURRENT
User profile . . . . . : *USER
Authority . . . . . : *LIBCRTAUT
Compiler . . . . . : IBM SAA COBOL/400
5738C01 V2R2M0 001000      AS/400 COBOL Source      TESTER/PGM1      AS400SYS 04/24/92 08:55:54      Page 2
STMT SEQNBR -A 1 B.+.2.+.3.+.4.+.5.+.6.+.7..IDENTFCN S COPYNAME CHG DATE
 1 000100 Identification division. 01/02/91
 2 000200 Program-id. pgml. 02/13/92
 3 000300 Environment division. 01/02/91
 4 000400 Configuration section. 01/02/91
 5 000500 Source-computer. ibm-as400. 01/02/91
 6 000600 Object-computer. ibm-as400. 01/02/91
 7 000700 Input-output section. 01/02/91
 8 000800 File-control. 01/02/91
 9 000900 Select file1 04/23/91
10 001000 assign to database-samplefile 02/13/92
11 001100 organization is sequential 04/23/91
12 001200 access is sequential 04/23/91
13 001300 file status is fsl. 04/23/91
14 001400 Data division. 01/02/91
15 001500 File division. 01/02/91
16 001600 fd file1. 01/02/91
17 001700 01 record1. 01/02/91
18 001800 copy dds-all-formats of samplefile. 02/13/92
19 +000001 05 SAMPLEFILE-RECORD PIC X(546). <-ALL-FMTS
+000002* I-O FORMAT:SAMPLEFILE FROM FILE SAMPLEFILE OF LIBRARY TESTER <-ALL-FMTS
+000003* <-ALL-FMTS
20 +000004 05 SAMPLEFILE REDEFINES SAMPLEFILE-RECORD. <-ALL-FMTS
21 +000005 06 VARITEM. <-ALL-FMTS
+000006* (Variable length field) <-ALL-FMTS
22 +000007 49 VARITEM-LENGTH PIC S9(4) COMP-4. <-ALL-FMTS
23 +000008 49 VARITEM-DATA PIC X(100). <-ALL-FMTS
24 +000009 06 TIMEITEM PIC X(8). <-ALL-FMTS
+000010* (Time field) <-ALL-FMTS
25 +000011 06 DATEITEM PIC X(8). <-ALL-FMTS
+000012* (Date field) <-ALL-FMTS
26 +000013 06 TIMESTAMP PIC X(26). <-ALL-FMTS
+000014* (Timestamp field) <-ALL-FMTS
27 +000015 06 GRAPHITEM PIC X(200). <-ALL-FMTS
+000016* (Graphic field) <-ALL-FMTS
28 +000017 06 VGRAPHITEM. <-ALL-FMTS
+000018* (Variable length field) <-ALL-FMTS
29 +000019 49 VGRAPHITEM-LENGTH PIC S9(4) COMP-4. <-ALL-FMTS
+000020* (Number of 2-byte characters) <-ALL-FMTS
30 +000021 49 VGRAPHITEM-DATA PIC X(200). <-ALL-FMTS
+000022* (Graphic field) <-ALL-FMTS
31 001900 working-storage section. 04/22/91
32 002000 77 fsl pic x(2). 04/23/91
33 002100 Procedure division. 01/09/91
 002200 Mainline. 01/02/91
34 002300 stop run. 01/02/91
***** END OF SOURCE *****
5738C01 V2R2M0 001000      AS/400 COBOL Messages      TESTER/PGM1      AS400SYS 04/24/92 08:55:54      Page 3
STMT
* 16 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 001000
  Message . . . . . : Blocking/Deblocking for file 'FILE1' will be
  performed by compiler-generated code.
  ***** END OF MESSAGES *****
  Message Summary
  Total  Info(0-4)  Warning(5-19)  Error(20-29)  Severe(30-39)  Terminal(40-99)
  1 1 0 0 0 0
Source records read . . . . . : 23
Copy records read . . . . . : 22
Copy members processed . . . . . : 1
Sequence errors . . . . . : 0
Highest severity message issued . . . : 0
  LBL0901 00 Program PGM1 created in library TESTER.
  ***** END OF COMPILATION *****

```

Figure 53. COBOL/400 Program Using Variable-Length DBCS-Graphic Data Items

Cross-system Data Considerations

Coded character set identifiers (CCSIDs) can help you to maintain the integrity of character data across systems.

Character Data Representation Architecture (CDRA) defines CCSID values to identify the code points used to represent characters, and to convert these codes as needed to preserve their meanings.

As a consequence of CDRA conversion, you might have substitution characters (X'3F') in your data. If you write these characters to a display, the results will not be predictable.

For more information about CCSIDs and CDRA, see *System Concepts*, GC41-9802 and the *Data Management Guide*.

Chapter 8. Transaction Files

IBM Extension

This chapter describes the COBOL/400 language extensions that support work stations and program-to-program communication.

The TRANSACTION file organization allows a COBOL program to communicate interactively with:

- One or more work station users
- One or more programs on a remote system
- One or more devices on a remote system.

The AS/400 system permits you to communicate with a program or device (such as Asynchronous communication types) on a remote system. For a detailed discussion of these devices, see the *ICF Programmer's Guide*.

Program-Described Transaction Files

COBOL TRANSACTION files are usually externally described. If these files are program-described, only simple display formatting can be performed. All field-level descriptions are defined in the COBOL program.

Do not send internal (packed) or binary data (COMP, COMP-3, or COMP-4) to a display station as output data. Such data can contain display station control characters that can cause unpredictable results.

See the *Data Management Guide* for more information about using program-described display files.

Externally Described Transaction Files

A COBOL TRANSACTION file uses an externally described file that contains file information and a description of the fields in the records. The records in this file can be described to the COBOL program by the Format 2 COPY statement.

The Format 2 COPY Statement

Format 2 COPY statements are used to generate COBOL Data Division statements within source programs to describe files that exist on the system.

Note: The term *Format 2 COPY statement* is used throughout this manual to describe the COPY statement (DD, DDR, DDS, or DDSR option).

For more information about the Format 2 COPY statement, see "Format 2 COPY Statement (DD, DDR, DDS, or DDSR Option)" on page 110.

Data Description Specifications

Data description specifications (DDS) are a description of the user's database or device files that are entered into the system in a fixed form. The description is then used to create files.

In addition to the field descriptions (such as field names and attributes), the data description specifications (DDS) for a display device file:

- Specify the line number and position number entries for each field and constant to format the placement of the record on the display.
- Specify attention functions such as underlining and highlighting fields, reverse image, or a blinking cursor.
- Specify validity checking for data entered at the display work station. Validity checking functions include:
 - Detecting fields where data is required
 - Detecting mandatory fill fields
 - Detecting incorrect data types
 - Detecting data for a specific range
 - Checking data for a valid entry
 - Performing modules 10 or 11 check digit verification.
- Control display management functions such as when fields are to be erased, overlaid, or retained when new data is displayed.
- Associate indicators 01 through 99 with function keys designated as type CA or CF. If a function key is designated as CF, both the modified data record and the response indicator are returned to the program. If a function key is designated as CA, the response indicator is returned to the program, but the data record usually contains default values for input-only fields and values written to the format for hidden output/input fields. For more information about type CF and CA function keys, see the *DDS Reference*.
- Assign an edit code (EDTCDE keyword) or edit word (EDTWRD keyword) to a field to specify how the field's values are to be displayed.
- Specify subfiles.

Display format data defines or describes a display. A display device record format contains three types of fields:

- *Input Fields*: Input fields pass from the device to the program when the program reads a record. Input fields can be initialized with a default value; if the default value is not changed, the default value passes to the program. Uninitialized input fields are displayed as blanks where the work station user can enter data.
- *Output Fields*: Output fields pass from the program to the device when the program writes a record to a display. The program or the record format in the device file can provide output fields.
- *Output/Input (both) Fields*: An output/input field is an output field that can be changed to become an input field. Output/input fields pass *from* the program when the program writes a record to a display and pass *to* the program when the program reads a record from the display. Output/input fields are used when the user is to change or update the data that is written to the display from the program.

For a detailed description of a data communications file, see the *ICF Programmer's Guide*. For more information on externally defined display files, see the *Data Management Guide*. For a list of the valid data description specifications (DDS) keywords, see the *DDS Reference*.

Figure 54 shows an example of the DDS for a display device file:

File		Keying Instruction	Graphic	Description		Page	of
Programmer	Date		Key				

Sequence Number	Form Type	And/Or Comment (A/D/+)	Conditioning				Name	Reference (R)	Length	Data Type/Keyboard Shift	Line	Pos	Location	Functions
			Indicator	Indicator	Indicator	Indicator								
1	A					CUSTOMER MASTER INQUIRY FILE								
2	A													
3	A					R CUSPMT							REF(CUSMSTP) 1	
4	A												TEXT('CUSTOMER PROMPT')	
5	A												CA03(15 'END OF PROGRAM') 2	
6	A										1	3	'CUSTOMER MASTER INQUIRY'	
7	A										3	3	'CUSTOMER NUMBER'	
8	A					CUST	R				1	3	20	
9	A	99											ERRMSG('CUSTOMER NUMBER NOT FOUND+ 3	
10	A												PRESS RESET, THEN ENTER VALID NUMBE+ 3	
11	A												R' 99)	
12	A												5 3'USE F3 TO END PROGRAM, USE ENTER+ 3	
13	A					R CUSFLDS							TO RETURN TO PROMPT SCREEN'	
14	A												TEXT('CUSTOMER DISPLAY')	
15	A												CA03(15 'END OF PROGRAM')	
16	A												OVERLAY 4	
17	A										8	3	'NAME'	
18	A					NAME	R				8	11		
19	A					ADDR	R				9	3	'ADDRESS'	
20	A										10	3	'CITY' 5	
21	A					CITY	R				10	11		
22	A										11	3	'STATE'	
23	A					STATE	R				11	11		
24	A										11	21	'ZIP CODE'	
25	A					ZIP	R				11	31		
26	A										12	3	'A/R BALANCE'	
27	A					ARBAL	R				12	17		

Figure 54. Example of the Data Description Specifications for a Display Device File

This display device file contains two record formats: CUSPMT and CUSFLDS.

- 1 The attributes for the fields in this file are defined in the CUSMSTP field reference file. For example, EDTCDE(J) is defined in CUSMSTP for the field ARBAL.
- 2 The F3 key is associated with indicator 15, with which the user ends the program.
- 3 The ERRMSG keyword identifies the error message that is displayed if indicator 99 is set on in the program that uses this record format.
- 4 The OVERLAY keyword is used for the record format CUSFLDS so that the CUSPMT record on the display will not be erased when the CUSFLDS record is written to the display.
- 5 The constants such as 'Name', 'Address', and 'City' describe the fields that are written out by the program.

- 6 The line and position entries identify where the fields or constants are written on the display.

Processing an Externally Described Transaction File

When an externally described TRANSACTION file is processed, the operating system transforms data from the program to the format specified for the file and displays the data. When data passes to the program, the data is transformed to the format used by the program.

The operating system provides device control information for performing input/output operations for the device. When an input record is requested from the device, the operating system issues the request, and then removes device control information from the data before passing the data to the program. In addition, the operating system can pass indicators to the program indicating which, if any, fields in the record have changed.

When the program requests an output operation, it passes the output record to the operating system. The operating system provides the necessary device control information to display the record. It also adds any constant information specified for the record format when the record is displayed.

When a record passes to a program, the fields are arranged in the order in which they are specified in the DDS. The order in which the fields are displayed is based on the display positions (line numbers and positions) assigned to the fields in the DDS. Therefore, the order in which the fields are specified in the DDS and the order in which they appear on the display need not be the same.

Using Indicators with Transaction Files

Indicators are Boolean data items that can have the values B"0" or B"1".

When you define a record format for a file using DDS, you can condition the options using indicators; indicators can also be used to reflect particular responses. These indicators are known as OPTION and RESPONSE, respectively.

Option indicators provide options such as spacing, underlining, and allowing or requesting data transfer from a program to a printer or display device. Response indicators provide response information to a program from a device, such as function keys pressed by a work station user, and whether data has been entered.

Indicators can be passed with data records in a record area, or outside the record area in a separate indicator area.

Indicators in a Separate Indicator Area

If you specify the file level keyword INDARA in the DDS, all indicators defined in the record format or formats for that file are passed to and from the program in a separate indicator area, not in the record area. For information on how to specify the INDARA keyword, see the *DDS Reference*.

The file control entry for a file that has INDARA specified in its DDS must have the separate indicator area attribute, SI, as part of the assignment-name.

The advantages of using a separate indicator area are as follows:

- The number and order of indicators used in an I/O statement for any record format in a file need not match the number and order of indicators specified in the DDS for that record format.
- The program associates the indicator number in a data description entry with the appropriate indicator.

Indicators in the Record Area

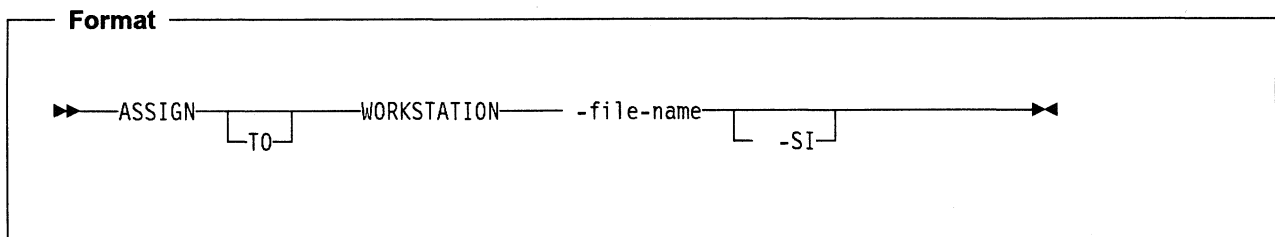
If the keyword INDARA is not used in the DDS of the file, indicators are created in the record area. When indicators are defined in a record format for a file, they are read, rewritten, and written with the data in the record area.

The number and order of indicators defined in the DDS for a record format for a file determines the number and order in which the data description entries for the indicators in the record format must be coded in the program.

The file control entry for a file that does not have the INDARA keyword specified in the DDS associated with it must *not* have the separate indicator area attribute, SI, as part of the assignment-name.

If a Format 2 COPY statement is used to copy indicators into a source program, the indicators are defined in the order in which they are specified in the DDS for the file.

ASSIGN Clause and the Separate Indicator Area Attribute



The rules for the ASSIGN clause are as follows:

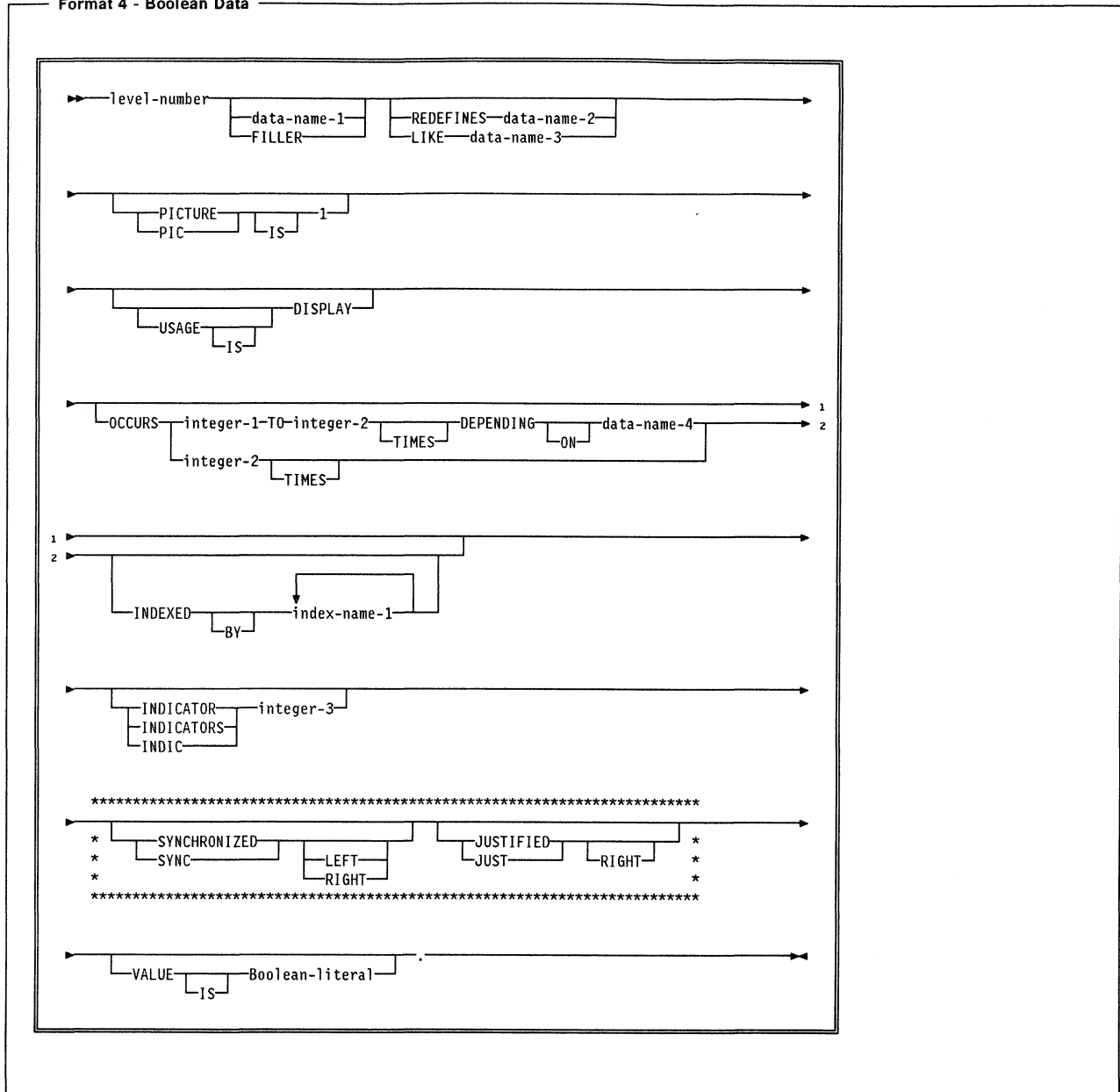
- Device must be WORKSTATION
- If -SI is coded, *file-name* must refer to a file that has the file level keyword INDARA specified in its DDS.

For more information about the ASSIGN clause, see “ASSIGN Clause” on page 168.

Data Description Entry – Boolean Data

When you use indicators in a COBOL program, you must describe them as Boolean data items using the data description entry for Boolean data.

Format 4 - Boolean Data



Special Considerations

The special considerations for the clauses used with the Boolean data follow. All other rules for clauses are the same as those for other data as described in the "COBOL Program Structure" section of the *COBOL/400* Reference*.

PICTURE Clause: An elementary Boolean data name is defined by a PICTURE containing a single 1.

USAGE Clause: USAGE must be defined implicitly or explicitly as DISPLAY.

OCCURS Clause: When the OCCURS clause and the INDICATOR clause are both specified at an elementary level, a table of Boolean data items is defined with each element in the table corresponding to an external indicator. The first element in the table corresponds to the indicator number specified in the INDICATOR clause; the second element corresponds to the indicator that sequentially follows the indicator specified by the INDICATOR clause.

For example, if the following is coded, SWITCHES (1) corresponds to indicator 16, SWITCHES (2) corresponds to indicator 17,..., and SWITCHES (10) corresponds to indicator 25:

```
07    SWITCHES    PIC 1
                        OCCURS 10 TIMES
                        INDICATOR 16.
```

INDICATOR Clause: If indicator fields are in a separate indicator area, the INDICATOR clause associates an indicator defined in DDS with a Boolean data item. If indicator fields are in the record area, the INDICATOR clause is syntax-checked, but is treated as a comment.

Integer-3 must have a value of 1 through 99.

The INDICATOR clause must be specified at an elementary level only.

VALUE Clause: The VALUE clause specifies the initial content of a Boolean data item. The allowable values for Boolean literals are B"0", B"1", and ZERO.

LIKE Clause: You cannot use this clause to change the length of the data item.

INDICATORS Phrase

When the INDICATORS phrase is used in READ, REWRITE, and WRITE statements (see Figure 57 on page 147), it specifies which indicators are to be read, rewritten, and written.

The identifier specified in the INDICATORS phrase can be either of the following:

- An elementary Boolean data item
- A group item with elementary Boolean data items subordinate to it. (The Boolean data items can be anywhere in the group, but they are the only items you can read, write, or rewrite.)

The identifier cannot be subordinate to an item that is subject to an OCCURS clause.

Indicators in a Separate Indicator Area

If INDARA is specified in the DDS for the file, the use of the indicators referenced in the INDICATORS phrase is based on indicator number.

- In a READ statement, only the response indicator numbers referenced by the INDICATORS phrase are updated. Indicators specified in the DDS for the format but not referenced by the INDICATORS phrase are ignored. Indicators referenced by the INDICATORS phrase but not specified in the DDS are not modified.
- In a WRITE or REWRITE statement, only the option indicators referenced by the INDICATORS phrase are used. Indicators specified in the DDS for the format but *not* referenced by the INDICATORS phrase are assumed to be **OFF**. Indicators referenced by the INDICATORS phrase but not used in the DDS for the format are ignored.

If the INDICATORS phrase is not specified, the following occurs:

- In the READ statement, indicators are not updated.
- In a WRITE or REWRITE statement, indicators are treated as though they are set to **OFF**.

Indicators in the Record Area

If INDARA is not specified in the DDS for the file, the size of the identifier in the INDICATORS phrase of an I/O statement (see Figure 57 on page 147) should be equal to the number of option or response indicators defined in the DDS for that format.

- In a READ statement, the identifier size should be equal to the number of response indicators.
- In a REWRITE or WRITE statement, the identifier size should be equal to the number of option indicators.

The contents of the identifier are not checked, but are copied to or from the beginning of the record, on a byte-by-byte basis; indicator numbers are ignored.

If the INDICATORS phrase is omitted, the data in the indicator fields in the record are still passed in the record area. The INDICATORS phrase is only used to copy indicators into the record area before a WRITE or REWRITE statement, or out of the record area after a READ statement.

Indicators Example Programs

This section contains examples of COBOL/400 programs that illustrate the use of indicators in either a record area or a separate indicator area.

All the programs do the following:

1. Determine the current date.
2. If it is the first day of the month, turn on an option indicator that causes an output field to appear and blink.
3. Allow you to press function keys to terminate the program, or turn on response indicators and call programs to write daily or monthly reports.

Figure 56 on page 145 shows a program that uses indicators in the record area but does not use the INDICATORS phrase in any I/O statement. Figure 55 on page 144 shows the associated DDS for the file.

Figure 57 on page 147 shows a program that uses indicators in the record area and the INDICATORS phrase in the I/O statements. The associated DDS for Figure 57 is Figure 55 on page 144.

Figure 59 on page 150 shows a program that uses indicators in a separate indicator area, defined in WORKING-STORAGE by using the Format 2 COPY statement. Figure 58 on page 149 shows the associated DDS for the file.

Figure 60 on page 152 shows a program that uses indicators in a separate indicator area, defined in a table in WORKING-STORAGE. The associated DDS for the file is the same as Figure 58 on page 149.

File		Keying Instruction		Graphic		Description		Page of																																														
Programmer	Date			Key																																																		
A																																																						
1	2	3	4	5	6	7	8	9	10																																													
<table border="1"> <thead> <tr> <th>Form Type Indicator</th> <th>Word/Comment (A/D/+)</th> <th>Condition Name</th> <th>Name</th> <th>Length</th> <th>Location</th> <th>Functions</th> </tr> </thead> <tbody> <tr> <td></td><td></td><td></td><td colspan="7">DISPLAY FILE DDS FOR INDICATOR EXAMPLES</td> </tr> <tr> <td></td><td></td><td>2</td><td>R</td><td>FORMAT1</td><td></td><td>3CF03(99 'END OF PROGRAM') 4CF05(51 'DAILY REPORT') 5CF09(52 'MONTHLY REPORT')</td> </tr> <tr> <td></td><td></td><td></td><td></td><td></td><td>4</td><td>10 10'DEPARTMENT NUMBER: '</td> </tr> <tr> <td></td><td>5</td><td>01</td><td>DEPTNO</td><td>5</td><td>I</td><td>10 32 20 26'PRODUCE MONTHLY REPORTS' DSPATR(BL)</td> </tr> <tr> <td></td><td></td><td></td><td></td><td></td><td>6</td><td>24 01'F5 - DAILY REPORT' 24 26'F9 - MONTHLY REPORT' 24 53'F3 - TERMINATE'</td> </tr> </tbody> </table>										Form Type Indicator	Word/Comment (A/D/+)	Condition Name	Name	Length	Location	Functions				DISPLAY FILE DDS FOR INDICATOR EXAMPLES									2	R	FORMAT1		3 CF03(99 'END OF PROGRAM') 4 CF05(51 'DAILY REPORT') 5 CF09(52 'MONTHLY REPORT')						4	10 10'DEPARTMENT NUMBER: '		5	01	DEPTNO	5	I	10 32 20 26'PRODUCE MONTHLY REPORTS' DSPATR(BL)						6	24 01'F5 - DAILY REPORT' 24 26'F9 - MONTHLY REPORT' 24 53'F3 - TERMINATE'
Form Type Indicator	Word/Comment (A/D/+)	Condition Name	Name	Length	Location	Functions																																																
			DISPLAY FILE DDS FOR INDICATOR EXAMPLES																																																			
		2	R	FORMAT1		3 CF03(99 'END OF PROGRAM') 4 CF05(51 'DAILY REPORT') 5 CF09(52 'MONTHLY REPORT')																																																
					4	10 10'DEPARTMENT NUMBER: '																																																
	5	01	DEPTNO	5	I	10 32 20 26'PRODUCE MONTHLY REPORTS' DSPATR(BL)																																																
					6	24 01'F5 - DAILY REPORT' 24 26'F9 - MONTHLY REPORT' 24 53'F3 - TERMINATE'																																																

Figure 55. Example of a Program Using Indicators in the Record Area without Using the INDICATORS Phrase in the I/O Statement—Data Description Specifications

- 1** The INDARA keyword is not used; indicators are stored in the record area with the data fields.
- 2** One record format, FORMAT1, is specified.
- 3** Three indicators are associated with three function keys. Indicator 99 will be set on when you press F3, and so on.
- 4** One field is defined for input.
- 5** Indicator 01 is defined to cause the associated constant field to blink if the indicator is on.
- 6** The function (F) key definitions are documented on the work station display.

```

5738CBI V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7..IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                03/09/89
 2 000200 PROGRAM-ID. XMPLE71.                                    03/22/89
 000300* PROGRAM EXAMPLE WITH INDICATORS IN RECORD AREA.         03/09/89
 3 000400 AUTHOR. PROGRAMMER NAME.                               03/09/89
 4 000500 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.       03/09/89
 5 000600 DATE-WRITTEN. 12/08/88.                                03/09/89
 6 000700 DATE-COMPILED. 05/24/91 11:02:36 .
 7 000800 ENVIRONMENT DIVISION.                                  03/09/89
 8 000900 CONFIGURATION SECTION.                                 03/09/89
 9 001000 SOURCE-COMPUTER. IBM-AS400.                            03/25/89
10 001100 OBJECT-COMPUTER. IBM-AS400.                            03/25/89
11 001200 INPUT-OUTPUT SECTION.                                  03/09/89
12 001300 FILE-CONTROL.                                          03/09/89
13 001400 SELECT DISPFILE                                        03/09/89
14 001500 ASSIGN TO WORKSTATION-DSPFILEX 1
15 001600 ORGANIZATION IS TRANSACTION                            03/09/89
16 001700 ACCESS IS SEQUENTIAL.                                  03/09/89
17 001800 DATA DIVISION.                                        03/09/89
18 001900 FILE SECTION.                                          03/09/89
19 002000 FD DISPFILE                                           03/09/89
20 002100 LABEL RECORDS ARE OMITTED                              03/09/89
21 002200 DATA RECORD IS DISP-REC.                              03/09/89
22 002300 01 DISP-REC.
23 002400 COPY DDS-ALL-FORMATS OF DSPFILEX. 2
24 +000001 05 DSPFILEX-RECORD PIC X(8). <-ALL-FMTS
+000002* INPUT FORMAT:FORMAT1 FROM FILE DSPFILEX OF LIBRARY XMPLIB <-ALL-FMTS
+000003* <-ALL-FMTS
25 +000004 05 FORMAT1-I REDEFINES DSPFILEX-RECORD. <-ALL-FMTS
26 +000005 06 FORMAT1-I-INDIC. <-ALL-FMTS
27 +000006 07 IN99 PIC 1 INDIC 99. 3 <-ALL-FMTS
+000007* END OF PROGRAM <-ALL-FMTS
28 +000008 07 IN51 PIC 1 INDIC 51. <-ALL-FMTS
+000009* DAILY REPORT <-ALL-FMTS
29 +000010 07 IN52 PIC 1 INDIC 52. <-ALL-FMTS
+000011* MONTHLY REPORT <-ALL-FMTS
30 +000012 06 DEPTNO PIC X(5). <-ALL-FMTS
+000013* OUTPUT FORMAT:FORMAT1 FROM FILE DSPFILEX OF LIBRARY XMPLIB <-ALL-FMTS
+000014* <-ALL-FMTS
31 +000015 05 FORMAT1-0 REDEFINES DSPFILEX-RECORD. <-ALL-FMTS
32 +000016 06 FORMAT1-0-INDIC. <-ALL-FMTS
33 +000017 07 IN01 PIC 1 INDIC 01. <-ALL-FMTS
002500
34 002600 WORKING-STORAGE SECTION.
35 002700 01 CURRENT-DATE.
36 002800 05 CURR-YEAR PIC 9(2).
37 002900 05 CURR-MONTH PIC 9(2).
38 003000 05 CURR-DAY PIC 9(2).
39 003100 01 INDIC-AREA. 4
40 003200 05 IN01 PIC 1.
41 003300 88 NEW-MONTH 5 VALUE B"1".
42 003400 05 IN51 PIC 1.
43 003500 88 WANT-DAILY VALUE B"1".
44 003600 05 IN52 PIC 1.
45 003700 88 WANT-MONTHLY VALUE B"1".
46 003800 05 IN99 PIC 1.
47 003900 88 NOT-END-OF-JOB VALUE B"0".
48 004000 88 END-OF-JOB VALUE B"1".
49 004100 PROCEDURE DIVISION.
004200 XAMPLE3-MAIN.
50 004300 OPEN I-O DISPFILE.
51 004400 ACCEPT CURRENT-DATE FROM DATE.
52 004500 SET NOT-END-OF-JOB TO TRUE.
53 004600 PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
004700 UNTIL END-OF-JOB.
54 004800 CLOSE DISPFILE.
55 004900 STOP RUN.
005000 DISPLAY-SCREEN.
56 005100 MOVE ZEROS TO INDIC-AREA. 6
57 005200 IF CURR-DAY = 01 THEN
58 005300 SET NEW-MONTH TO TRUE. 7
59 005400 MOVE CORR INDIC-AREA TO FORMAT1-0-INDIC. 8
60 005500 WRITE DISP-REC FORMAT IS "FORMAT1". 9
005600 READ-AND-PROCESS-SCREEN.
61 005700 MOVE ZEROS TO INDIC-AREA.
62 005800 READ DISPFILE FORMAT IS "FORMAT1". 10

```

Figure 56 (Part 1 of 2). Example of a Program Using Indicators in the Record Area without Using the INDICATORS Phrase in the I/O Statement—COBOL Source Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B.+...2....3....4....5....6....7..IDENTFCN S COPYNAME  CHG DATE
 63 005900 MOVE CORR FORMAT1-I-INDIC TO INDIC-AREA. 11
 64 006000 IF WANT-DAILY THEN
 65 006100 CALL "DAILY" USING DEPTNO 12
    006200 ELSE
 66 006300 IF WANT-MONTHLY THEN
 67 006400 CALL "MONTHLY" USING DEPTNO.
          ***** END OF SOURCE *****
5738CB1 V2R2M0          AS/400 COBOL Messages
STMT
          ***** END OF MESSAGES *****
Message Summary
Total   Info(0-4)   Warning(5-19)   Error(20-29)   Severe(30-39)   Terminal(40-99)
  0      0          0                0                0                0
Source records read . . . . . : 64
Copy records read . . . . . : 17
Copy members processed . . . . . : 1
Sequence errors . . . . . : 0
Highest severity message issued . . : 0
LBL0901 00 Program XMPLE71 created in library XMPLIB.
          ***** END OF COMPILATION *****

```

Figure 56 (Part 2 of 2). Example of a Program Using Indicators in the Record Area without Using the INDICATORS Phrase in the I/O Statement—COBOL Source Program

- 1 The separate indicator area attribute, SI, is not coded in the ASSIGN clause.
- 2 The Format 2 COPY statement defines data fields and indicators in the record area.
- 3 Because the file does not have a separate indicator area, response and option indicators are defined in the order in which they are used in the DDS, and the indicator numbers are treated as documentation.
- 4 All indicators used by the program are defined with meaningful names in data description entries in WORKING-STORAGE. Indicator numbers are omitted here because they have no effect.
- 5 For each indicator, a meaningful level-88 condition-name is associated with a value for that indicator.
- 6 Initialize group level to zeros.
- 7 IN01 in WORKING-STORAGE is set on if it is the first day of the month.
- 8 Indicators appropriate to the output of FORMAT1 are copied to the record area.
- 9 FORMAT1 is written to the work station display with both data and indicator values in the record area.
The INDICATORS phrase is not necessary because there is no separate indicator area and indicator values have been set in the record area through the previous MOVE CORRESPONDING statement.
- 10 FORMAT1, including both data and indicators, is read from the display.
- 11 The response indicators for FORMAT1 are copied from the record area to the data description entries in WORKING-STORAGE.
- 12 If F5 has been pressed, a program call is processed.

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7..IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                03/07/89
 2 000200 PROGRAM-ID. XMPLE713.                                  03/22/89
 000300* SAMPLE PROGRAM - FILE WITH INDICATORS IN RECORD AREA    03/07/89
 3 000400 AUTHOR. PROGRAMMER NAME.                               03/07/89
 4 000500 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.       03/07/89
 5 000600 DATE-WRITTEN. 12/10/88.                               03/07/89
 6 000700 DATE-COMPILED. 05/24/91 11:04:34 .
 7 000800 ENVIRONMENT DIVISION.                                  03/07/89
 8 000900 CONFIGURATION SECTION.                                03/07/89
 9 001000 SOURCE-COMPUTER. IBM-AS400.                           03/07/89
10 001100 OBJECT-COMPUTER. IBM-AS400.                           03/07/89
11 001200 INPUT-OUTPUT SECTION.                                  03/07/89
12 001300 FILE-CONTROL.                                         03/07/89
13 001400 SELECT DISPFILE                                        03/07/89
14 001500 ASSIGN TO WORKSTATION-DSPFILEX 1
15 001600 ORGANIZATION IS TRANSACTION                           03/07/89
16 001700 ACCESS IS SEQUENTIAL.                                03/07/89
17 001800 DATA DIVISION.                                       03/07/89
18 001900 FILE SECTION.                                         03/07/89
19 002000 FD DISPFILE                                           03/07/89
20 002100 LABEL RECORDS ARE OMITTED                             03/07/89
21 002200 DATA RECORD IS DISP-REC.                             03/07/89
22 002300 01 DISP-REC.                                          03/07/89
23 002400 COPY DDS-ALL-FORMATS OF DSPFILEX. 2
24 +000001 05 DSPFILEX-RECORD PIC X(8).
+000002* INPUT FORMAT:FORMAT1 FROM FILE DSPFILEX OF LIBRARY XMPLIB <-ALL-FMTS
+000003* <-ALL-FMTS
25 +000004 05 FORMAT1-I REDEFINES DSPFILEX-RECORD. <-ALL-FMTS
26 +000005 06 FORMAT1-I-INDIC. <-ALL-FMTS
27 +000006 07 IN99 PIC 1 INDIC 99. 3 <-ALL-FMTS
+000007* END OF PROGRAM <-ALL-FMTS
28 +000008 07 IN51 PIC 1 INDIC 51. <-ALL-FMTS
+000009* DAILY REPORT <-ALL-FMTS
29 +000010 07 IN52 PIC 1 INDIC 52. <-ALL-FMTS
+000011* MONTHLY REPORT <-ALL-FMTS
30 +000012 06 DEPTNO PIC X(5). <-ALL-FMTS
+000013* OUTPUT FORMAT:FORMAT1 FROM FILE DSPFILEX OF LIBRARY XMPLIB <-ALL-FMTS
+000014* <-ALL-FMTS
31 +000015 05 FORMAT1-O REDEFINES DSPFILEX-RECORD. <-ALL-FMTS
32 +000016 06 FORMAT1-O-INDIC. <-ALL-FMTS
33 +000017 07 IN01 PIC 1 INDIC 01. <-ALL-FMTS
002500
34 002600 WORKING-STORAGE SECTION.
35 002700 01 CURRENT-DATE.
36 002800 05 CURR-YEAR PIC 9(2).
37 002900 05 CURR-MONTH PIC 9(2).
38 003000 05 CURR-DAY PIC 9(2).
003100
39 003200 77 IND-OFF PIC 1 VALUE B"0".
40 003300 77 IND-ON PIC 1 VALUE B"1".
003400
41 003500 01 RESPONSE-INDICS.
42 003600 05 END-OF-PROGRAM PIC 1. 4
43 003700 05 DAILY-REPORT PIC 1.
44 003800 05 MONTHLY-REPORT PIC 1.
45 003900 01 OPTION-INDICS.
46 004000 05 NEW-MONTH PIC 1.
004100
47 004200 PROCEDURE DIVISION.
004300 XMPLE3-MAIN.
48 004400 OPEN I-O DISPFILE.
49 004500 ACCEPT CURRENT-DATE FROM DATE.
50 004600 MOVE IND-OFF TO END-OF-PROGRAM.
51 004700 PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
004800 UNTIL END-OF-PROGRAM = IND-ON.
52 004900 CLOSE DISPFILE.
53 005000 STOP RUN.
005100
005200 DISPLAY-SCREEN.
54 005300 MOVE ZEROS TO OPTION-INDICS.
55 005400 IF CURR-DAY = 01 THEN 5
56 005500 MOVE IND-ON TO NEW-MONTH.
57 005600 WRITE DISP-REC FORMAT IS "FORMAT1" 6
005700 INDICATORS ARE OPTION-INDICS.
005800

```

Figure 57 (Part 1 of 2). Example of a Program Using Indicators in the Record Area and the INDICATORS phrase in the I/O Statements – COBOL Source Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
005900 READ-AND-PROCESS-SCREEN.
58 006000 MOVE ZEROS TO RESPONSE-INDICS.
59 006100 READ DISPFILE FORMAT IS "FORMAT1" 7
006200          INDICATORS ARE RESPONSE-INDICS. 8
60 006300 IF DAILY-REPORT = IND-ON THEN
61 006400 CALL "DAILY" USING DEPTNO 9
006500 ELSE
62 006600 IF MONTHLY-REPORT = IND-ON THEN
63 006700 CALL "MONTHLY" USING DEPTNO.
          ***** END OF SOURCE *****
5738CB1 V2R2M0          AS/400 COBOL Messages
STMT
          ***** END OF MESSAGES *****
          Message Summary
Total      Info(0-4)      Warning(5-19)      Error(20-29)      Severe(30-39)      Terminal(40-99)
0          0          0          0          0          0
Source records read . . . . . : 67
Copy records read . . . . . : 17
Copy members processed . . . . . : 1
Sequence errors . . . . . : 0
Highest severity message issued . . : 0
LBL0901 00 Program XMPLE713 created in library XMPLIB.
          ***** END OF COMPILATION *****

```

Figure 57 (Part 2 of 2). Example of a Program Using Indicators in the Record Area and the INDICATORS phrase in the I/O Statements – COBOL Source Program

- 1 The separate indicator area attribute, SI, is not coded in the ASSIGN clause.
- 2 The Format 2 COPY statement defines data fields and indicators in the record area.
- 3 Because the file does not have a separate indicator area, response and option indicators are defined in the order in which they are used in the DDS, and the indicator numbers are treated as documentation.
- 4 All indicators used by the program are defined with meaningful names in data description entries in WORKING-STORAGE. Indicator numbers are omitted here because they have no effect. Indicators should be defined in the order needed by the display file.
- 5 IN01 in WORKING-STORAGE is set on if it is the first day of the month.
- 6 FORMAT1 is written to the work station display:
 - The INDICATORS phrase causes the contents of the variable OPTION-INDICS to be copied to the beginning of the record area.
 - Data and indicator values are written to the work station display.
- 7 FORMAT1, including both data and indicators, is read from the work station display.
- 8 The INDICATORS phrase causes bytes to be copied from the beginning of the record area to RESPONSE-INDICS.
- 9 If F5 has been pressed, a program call is processed.

File	Keying	Graphic						
Programmer	Instruction	Key						
Date								

Description	Page	of

Sequence Number	Type	Code	Comment	Conditioning			Name	Length	Reference	Data Type	Keyboard Shift	Decimal Position	Location		Functions
				Indicator	Indicator	Indicator							Line	Pos	
1	A														
2	A														
3	A														
4	A														
5	A														
6	A														
7	A														
8	A														
9	A														
10	A														
11	A														
12	A														
13	A														
14	A														
15	A														
16	A														
17	A														
18	A														
19	A														
20	A														
21	A														
22	A														
23	A														
24	A														
25	A														
26	A														
27	A														
28	A														
29	A														
30	A														
31	A														
32	A														
33	A														
34	A														
35	A														
36	A														
37	A														
38	A														
39	A														
40	A														
41	A														
42	A														
43	A														
44	A														
45	A														
46	A														
47	A														
48	A														
49	A														
50	A														
51	A														
52	A														
53	A														
54	A														
55	A														
56	A														
57	A														
58	A														
59	A														
60	A														
61	A														
62	A														
63	A														
64	A														
65	A														
66	A														
67	A														
68	A														
69	A														
70	A														
71	A														
72	A														
73	A														
74	A														
75	A														
76	A														
77	A														
78	A														
79	A														
80	A														

Figure 58. Example of a Program Using Indicators in a Separate Indicator Area, Defined in WORKING-STORAGE by Using the COPY Statement, DDS Format

- 1** The INDARA keyword is specified; indicators are stored in a separate indicator area, not in the record area. Except for this specification, the DDS for this file is the same as that shown in Figure 55 on page 144.

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B.+. . . . 2. . . . 3. . . . 4. . . . 5. . . . 6. . . . 7. . IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                03/09/89
 2 000200 PROGRAM-ID.      XMPLE717.                              03/22/89
   000300*   SAMPLE PROGRAM - FILE WITH SEPARATE INDICATORS AREA  03/09/89
 3 000400 AUTHOR.          PROGRAMMER NAME.                      03/09/89
 4 000500 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.       03/09/89
 5 000600 DATE-WRITTEN. 12/08/88.                                03/09/89
 6 000700 DATE-COMPILED. 05/24/91 12:53:17 .
 7 000800 ENVIRONMENT DIVISION.                                  03/09/89
 8 000900 CONFIGURATION SECTION.                                03/09/89
 9 001000 SOURCE-COMPUTER. IBM-AS400.                            03/09/89
10 001100 OBJECT-COMPUTER. IBM-AS400.                            03/09/89
11 001200 INPUT-OUTPUT SECTION.                                  03/09/89
12 001300 FILE-CONTROL.                                         03/09/89
13 001400   SELECT DISPFILE                                       03/09/89
14 001500   ASSIGN TO WORKSTATION-DSPFILE-SI 1
15 001600   ORGANIZATION IS TRANSACTION                          03/09/89
16 001700   ACCESS IS SEQUENTIAL.                                03/09/89
   001800
17 001900 DATA DIVISION.                                       03/09/89
18 002000 FILE SECTION.                                         03/09/89
19 002100 FD DISPFILE                                           03/09/89
20 002200   LABEL RECORDS ARE OMITTED                           03/09/89
21 002300   DATA RECORD IS DISP-REC.                           03/09/89
22 002400 01 DISP-REC.                                           03/09/89
23 002500   COPY DDS-ALL-FORMATS OF DSPFILE. 2
24 +000001   05 DSPFILE-RECORD PIC X(5).                         <-ALL-FMTS
   +000002* INPUT FORMAT:FORMAT1 FROM FILE DSPFILE OF LIBRARY XMPLIB <-ALL-FMTS
   +000003*                                         <-ALL-FMTS
25 +000004   05 FORMAT1-I REDEFINES DSPFILE-RECORD.              <-ALL-FMTS
26 +000005   06 DEPTNO PIC X(5).                                  <-ALL-FMTS
   +000006* OUTPUT FORMAT:FORMAT1 FROM FILE DSPFILE OF LIBRARY XMPLIB <-ALL-FMTS
   +000007*                                         <-ALL-FMTS
   +000008*   05 FORMAT1-O REDEFINES DSPFILE-RECORD.              <-ALL-FMTS
   002600
27 002700 WORKING-STORAGE SECTION.
28 002800 01 CURRENT-DATE.
29 002900   05 CURR-YEAR PIC 9(2).
30 003000   05 CURR-MONTH PIC 9(2).
31 003100   05 CURR-DAY PIC 9(2).
   003200
32 003300   77 IND-OFF PIC 1 VALUE B"0".
33 003400   77 IND-ON PIC 1 VALUE B"1".
34 003500 01 DISPFILE-INDICS.
35 003600   COPY DDS-ALL-FORMATS-INDIC OF DSPFILE. 3
36 +000001   05 DSPFILE-RECORD.                                  <-ALL-FMTS
   +000002* INPUT FORMAT:FORMAT1 FROM FILE DSPFILE OF LIBRARY XMPLIB <-ALL-FMTS
   +000003*                                         <-ALL-FMTS
37 +000004   06 FORMAT1-I-INDIC.                                  <-ALL-FMTS
38 +000005   07 IN51 PIC 1 INDIC 51. 4
   +000006*   DAILY REPORT                                       <-ALL-FMTS
39 +000007   07 IN52 PIC 1 INDIC 52.                               <-ALL-FMTS
   +000008*   MONTHLY REPORT                                       <-ALL-FMTS
40 +000009   07 IN99 PIC 1 INDIC 99.                               <-ALL-FMTS
   +000010*   END OF PROGRAM                                       <-ALL-FMTS
   +000011* OUTPUT FORMAT:FORMAT1 FROM FILE DSPFILE OF LIBRARY XMPLIB <-ALL-FMTS
   +000012*
41 +000013   06 FORMAT1-O-INDIC.
42 +000014   07 IN01 PIC 1 INDIC 01.
   003700
43 003800 PROCEDURE DIVISION.
   003900
   004000 MAIN-PROCESS.
   004100
44 004200   OPEN I-O DISPFILE.
45 004300   ACCEPT CURRENT-DATE FROM DATE.
46 004400   MOVE IND-OFF TO IN99 IN FORMAT1-I-INDIC.
47 004500   PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
   004600   UNTIL IN99 IN FORMAT1-I-INDIC = IND-ON.
48 004700   CLOSE DISPFILE.
49 004800   STOP RUN.
   004900
   005000 DISPLAY-SCREEN.
   005100
50 005200   MOVE ZEROS TO FORMAT1-O-INDIC.
51 005300   IF CURR-DAY = 01 THEN
52 005400   MOVE IND-ON TO IN01 IN FORMAT1-O-INDIC. 5

```

Figure 59 (Part 1 of 2). COBOL Listing Using Indicators in a Separate Indicator Area

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S COPYNAME  CHG DATE
 53 005500  WRITE DISP-REC FORMAT IS "FORMAT1"
      005600          INDICATORS ARE FORMAT1-0-INDIC. 6
      005700
      005800 READ-AND-PROCESS-SCREEN.
      005900
 54 006000  MOVE ZEROS TO FORMAT1-I-INDIC.
 55 006100  READ DISPFILE FORMAT IS "FORMAT1"
      006200          INDICATORS ARE FORMAT1-I-INDIC. 7
 56 006300  IF IN51 IN FORMAT1-I-INDIC = IND-ON THEN
 57 006400  CALL "DAILY" USING DEPTNO 8
      006500  ELSE
 58 006600  IF IN52 IN FORMAT1-I-INDIC = IND-ON THEN
 59 006700  CALL "MONTHLY" USING DEPTNO.
          ***** END OF SOURCE *****
5738CB1 V2R2M0          AS/400 COBOL Messages
STMT
* 23 MSGID: LBL0600 SEVERITY: 10 SEQNBR: 000250
  Message . . . . : No OUTPUT fields found for format FORMAT1.
          ***** END OF MESSAGES *****
          Message Summary
Total      Info(0-4)  Warning(5-19)  Error(20-29)  Severe(30-39)  Terminal(40-99)
  1          0          1          0          0          0
Source records read . . . . . : 67
Copy records read . . . . . : 22
Copy members processed . . . . . : 2
Sequence errors . . . . . : 0
Highest severity message issued . . : 10
  LBL0901 00 Program XMPLE717 created in library XMPLIB.
          ***** END OF COMPILATION *****

```

Figure 59 (Part 2 of 2). COBOL Listing Using Indicators in a Separate Indicator Area

- 1 The separate indicator area attribute, SI, is specified in the ASSIGN clause.
- 2 The Format 2 COPY statement generates data descriptions in the record area for data fields only. The data description entries for the indicators are not generated because a separate indicator area has been specified for the file.
- 3 The Format 2 COPY statement, with the INDICATOR attribute, INDIC, defines data description entries in WORKING-STORAGE for all indicators used in the DDS for the record format for the file.
- 4 Because the file has a separate indicator area, the indicator numbers used in the data description entries are not treated as documentation.
- 5 IN01 in the separate indicator area for FORMAT1 is set on if it is the first day of the month.
- 6 The INDICATORS phrase is required to send indicator values to the work station display.
- 7 The INDICATORS phrase is required to receive indicator values from the work station display. If you have pressed F5, IN51 is set on.
- 8 If IN51 has been set on, a program call is processed.

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7..IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                01/22/89
 2 000200 PROGRAM-ID.      XMPLE720.                              03/22/89
   000300* PROGRAM EXAMPLE                                        01/22/89
   000400* FILE WITH SEPARATE INDICATORS AREA IN WORKING STORAGE 01/22/89
 3 000500 AUTHOR.          PROGRAMMER NAME.                      01/22/89
 4 000600 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.       01/22/89
 5 000700 DATE-WRITTEN. 12/08/88.                                01/22/89
 6 000800 DATE-COMPILED. 05/24/91 12:46:00 .                    01/22/89
 7 000900 ENVIRONMENT DIVISION.                                  01/22/89
 8 001000 CONFIGURATION SECTION.                                 01/22/89
 9 001100 SOURCE-COMPUTER. IBM-AS400.                            01/22/89
10 001200 OBJECT-COMPUTER. IBM-AS400.                            01/22/89
11 001300 INPUT-OUTPUT SECTION.                                  01/22/89
12 001400 FILE-CONTROL.                                          01/22/89
13 001500     SELECT DSPFILE                                     01/22/89
14 001600     ASSIGN TO WORKSTATION-DSPFILE-SI 1                 03/22/89
15 001700     ORGANIZATION IS TRANSACTION                       01/22/89
16 001800     ACCESS IS SEQUENTIAL.                             01/22/89
   001900
17 002000 DATA DIVISION.                                        01/22/89
18 002100 FILE SECTION.                                          01/22/89
19 002200 FD DSPFILE                                           01/22/89
20 002300 LABEL RECORDS ARE OMITTED                             01/22/89
21 002400 DATA RECORD IS DISP-REC.                             01/22/89
22 002500 01 DISP-REC.                                          01/22/89
23 002600 COPY DDS-ALL-FORMATS OF DSPFILE. 2                     03/22/89
24 +000001 05 DSPFILE-RECORD PIC X(5).                           <-ALL-FMTS
   +000002* INPUT FORMAT:FORMAT1 FROM FILE DSPFILE OF LIBRARY XMPLIB <-ALL-FMTS
   +000003*                                     <-ALL-FMTS
25 +000004 05 FORMAT1-I REDEFINES DSPFILE-RECORD.                <-ALL-FMTS
26 +000005 06 DEPTNO PIC X(5).                                   <-ALL-FMTS
   +000006* OUTPUT FORMAT:FORMAT1 FROM FILE DSPFILE OF LIBRARY XMPLIB <-ALL-FMTS
   +000007*                                     <-ALL-FMTS
   +000008* 05 FORMAT1-O REDEFINES DSPFILE-RECORD.              <-ALL-FMTS
   002700
27 002800 WORKING-STORAGE SECTION.
28 002900 01 CURRENT-DATE.
29 003000 05 CURR-YEAR PIC 9(2).
30 003100 05 CURR-MONTH PIC 9(2).
31 003200 05 CURR-DAY PIC 9(2).
   003300
32 003400 01 INDIC-AREA.
33 003500 05 INDIC-TABLE OCCURS 99 PIC 1 INDICATOR 1. 3
34 003600 88 IND-OFF VALUE B"0".
35 003700 88 IND-ON VALUE B"1".
   003800
36 003900 01 DISPFILE-INDIC-USAGE.
37 004000 05 IND-NEW-MONTH PIC 9(2) VALUE 01.
38 004100 05 IND-DAILY PIC 9(2) VALUE 51. 4
39 004200 05 IND-MONTHLY PIC 9(2) VALUE 52.
40 004300 05 IND-EOJ PIC 9(2) VALUE 99.
   004400
41 004500 PROCEDURE DIVISION.
   004600
   004700 XMPLE-MAIN.
42 004800 OPEN I-O DISPFILE.
43 004900 ACCEPT CURRENT-DATE FROM DATE.
44 005000 SET IND-OFF (IND-EOJ) TO TRUE.
45 005100 PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
   005200 UNTIL IND-ON (IND-EOJ).
46 005300 CLOSE DISPFILE.
47 005400 STOP RUN.
   005500
   005600 DISPLAY-SCREEN.
   005700
48 005800 MOVE ZEROS TO INDIC-AREA.
49 005900 IF CURR-DAY = 01 THEN
50 006000 SET IND-ON (IND-NEW-MONTH) TO TRUE. 5
51 006100 WRITE DISP-REC FORMAT IS "FORMAT1"
   006200 INDICATORS ARE INDIC-TABLE. 6
   006300
   006400 READ-AND-PROCESS-SCREEN.
   006500
52 006600 READ DISPFILE FORMAT IS "FORMAT1"
   006700 INDICATORS ARE INDIC-TABLE. 7

```

Figure 60 (Part 1 of 2). Example of a Program Using Indicators in a Separate Indicator Area, Defined in a Table in WORKING-STORAGE

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
53 006800 IF IND-ON (IND-DAILY) THEN 8
54 006900 CALL "DAILY" USING DEPTNO
    007000 ELSE
55 007100 IF IND-ON (IND-MONTHLY) THEN
56 007200 CALL "MONTHLY" USING DEPTNO.
          ***** END OF SOURCE *****
5738CB1 V2R2M0          AS/400 COBOL Messages
STMT
* 23 MSGID: LBL0600 SEVERITY: 10 SEQNBR: 000260
  Message . . . . : No OUTPUT fields found for format FORMAT1.
          ***** END OF MESSAGES *****
                          Message Summary
Total      Info(0-4)   Warning(5-19)   Error(20-29)   Severe(30-39)   Terminal(40-99)
1          0          1                0                0                0
Source records read . . . . . : 72
Copy records read . . . . . : 8
Copy members processed . . . . . : 1
Sequence errors . . . . . : 0
Highest severity message issued . . : 10
LBL0901 00 Program XMPLE720 created in library XMPLIB.
          ***** END OF COMPILATION *****

```

Figure 60 (Part 2 of 2). Example of a Program Using Indicators in a Separate Indicator Area, Defined in a Table in WORKING-STORAGE

- 1 The separate indicator area attribute, SI, is specified in the ASSIGN clause.
- 2 The Format 2 COPY statement generates fields in the record area for data fields only.
- 3 A table of 99 Boolean data items is defined in WORKING-STORAGE. The INDICATOR clause for this data description entry causes these data items to be associated with indicators 1 through 99 respectively. The use of such a table may result in improved performance as compared to the use of a group item with multiple subordinate entries for individual indicators.
- 4 A series of data items is defined in WORKING-STORAGE to provide meaningful subscript names with which to refer to the table of indicators. The use of such data items is not required.
- 5 INDIC-TABLE (01) in the separate indicator area for FORMAT1 is set on if it is the first day of the month.
- 6 The INDICATOR phrase is required to send indicator values to the work station display.
- 7 The INDICATOR phrase is required to receive indicator values from the work station display. If F5 has been pressed, INDIC-TABLE (51) will be set on.
- 8 If INDIC-TABLE (51) has been set on, program DAILY is called.

Subfiles

Subfiles can be specified in the DDS for a display file to allow you to handle multiple records of the same type on a display. See Figure 61 on page 154 for an example of a subfile display. A **subfile** is a group of records that are read from or written to a display device. The program processes one record at a time, but the operating system and the work station send and receive blocks of records. If more records are transmitted than can be shown on the display at one time, the work station operator can page through the block of records without returning control to the program.

key is an unsigned number that can be used directly by the system to locate a record in a file.

The TRANSACTION file must be an externally defined file. In COBOL, all access to the subfile is done with a relative record number. If the SUBFILE phrases are used with a TRANSACTION file, the SELECT statement in the Environment Division must state that ACCESS MODE IS DYNAMIC and must specify the RELATIVE KEY to be used.

If more than one display device is acquired by a display file, there is a separate subfile for each individual display device. If a subfile has been created for a particular display device acquired by a TRANSACTION file, all input operations that refer to a record format for the subfile are performed against the subfile belonging to that device. See the discussion on the TERMINAL phrase on page 178 of this chapter for information about how to determine which device is used. Any operations that reference a record format name that is not designated as a subfile are processed as an input/output operation directly to the display device.

Use of Subfiles

Some typical uses of subfiles include:

Use	Meaning
Display Only	The work station user reviews the display.
Display With Selection	The user requests more information about one of the items on display.
Modification	The user modifies one or more of the records.
Input Only (with no validity checking)	A subfile is used for a data-entry function.
Input Only (with validity checking)	A subfile is used for a data-entry function, and the records are checked as well.
Combination of Tasks	A subfile can be used as a display with modification.

File		Keying Instruction		Graphic		Description		Page	
Programmer	Date	Instruction	Key	Key	Key			of	
A		Conditioning							
Sequence Number	Form Type	Condition Name	Name	Length	Location	Line	Pos	Functions	
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
1	A*	DDS FOR THE DISPLAY DEVICE FILE PAYUPDT							
2	A*	ACCOUNTS RECEIVABLE INTERACTIVE PAYMENT UPDATE							
3	A	R SUBFILE1				1		SFL	
4	A*							TEXT('SUBFILE FOR CUSTOMER PAYMENT')	
5	A	ACPPMT		4A	I	5	4	TEXT('ACCEPT PAYMENT')	
6	A					2		VALUES('*YES' '*NO') 3	
7	A							DSPATR(RI MDT)	
8	A							DSPATR(ND PR)	
9	A*								
10	A	CUST		5	B	5	15	TEXT('CUSTOMER NUMBER')	
11	A					4		DSPATR(RI)	
12	A							DSPATR(ND)	
13	A							DSPATR(PR)	
14	A*								
15	A	AMPAID		8	O2B	5	24	TEXT('AMOUNT PAID')	
16	A							CHECK(FE) 3	
17	A							AUTO(RAB) 3	
18	A							CMP(GT 0) 7	
19	A							DSPATR(RI)	
20	A							DSPATR(ND)	
21	A							DSPATR(PR)	
22	A*								
23	A	ECPMSG		31A	O	5	37	TEXT('EXCEPTION MESSAGE')	
24	A							DSPATR(RI)	
25	A							DSPATR(ND)	
26	A							DSPATR(BL)	
27	A*								
28	A	OVRPMT		BY 20		5	70	TEXT('OVERPAYMENT')	
29	A							EDTCDE(1)	
30	A							DSPATR(BL)	
31	A							DSPATR(ND)	
32	A*								
33	A	STSCDE		1A	H			TEXT('STATUS CODE')	

Figure 62. Data Description Specifications for a Subfile Record Format

The data description specifications (DDS) for a subfile record format describe the records in the subfile:

- 1 The SFL keyword identifies the record format as a subfile.
- 2 The line and position entries define the location of the fields on the display.
- 3 The VALUES keyword specifies that the user can only specify *YES or *NO as values for the ACPMT field.
- 4 The usage entries define whether the named field is to be an output (O), input (I), output/input (B), or hidden (H) field.
- 5 The entry CHECK(FE) specifies that the user cannot skip to the next input field without pressing one of the field exit keys.
- 6 The entry AUTO(RAB) specifies that data entered into the field AMPAID is to be automatically right-justified, and the leading characters are to be filled with blanks.
- 7 The entry CMP(GT 0) specifies that the data entered for the field AMPAID is to be compared to zero to ensure that the value is greater than zero.

8 The EDTCDE keyword specifies the desired editing for output field OVRPMT. EDTCDE(1) indicates that the field OVRPMT is to be printed with commas, decimal point, and no sign. Also, a zero balance will be printed, and leading zeros will be suppressed.

9 The DSPATR keyword is used to specify the display attributes for the named field when the corresponding indicator status is true. The attributes specified are:

- BL (blink)
- RI (reverse image)
- PR (protect)
- MDT (set modified data tag)
- ND (nondisplay).

File	Keying Instruction	Graphic	Description	Page	of
Programmer	Date	Key			

Sequence Number	Conditioning					Name	Length	Reference ID	Data Type/Keyboard Shift	Decimals	Positions	Location		Function
	Form Type	Indic	Indic	Indic	Indic							Line	Pos	
					R	CONTROL1								TEXT('SUBFILE CONTROL')
														SFLCTL(SUBFILE1) 1
														SFLSIZ(17) 2
														SFLPAG(17) 3
			61											SFLCLR 4
			62											SFLDSP 5
			62											SFLDSPCTL 6
														OVERLAY
														LOCK 7
														HELP(99 'HELP KEY') 8
														CA12(98 'END PAYMENT UPDATE')
														CA11(97 'IGNORE INPUT')
			99											9 SFLMSG('F11 - IGNORE INVALID INPUT + F12 - END PAYMENT + UPDATE')
														1 2'CUSTOMER PAYMENT UPDATE PROMPT'
														1 65'DATE'
														1 78DATE EDTCDE(Y)
			63											3 2'ACCEPT'
			63											4 2'PAYMENT'
														3 14'CUSTOMER'
														3 26'PAYMENT'
			64											3 37'EXCEPTION MESSAGE'

Figure 63. Data Description Specifications for a Subfile Control Record Format

The subfile control record format defines the attributes of the subfile, the search input field, constants, and command keys. The keywords used indicate the following:

- 1 SFLCTL identifies this record as a subfile control record and names the associated subfile record (SUBFILE1).

- 2** SFLSIZ indicates the total number of records to be included in the subfile (17).
- 3** SFLPAG indicates the total number of records in a page (17).
- 4** SFLCLR indicates when the subfile should be cleared (when indicator 61 is on).
- 5** SFLDSP indicates when to display the subfile (when indicator 62 is on).
- 6** SFLDSPCTL indicates when to display the subfile control record (when indicator 62 is on).
- 7** The LOCK keyword prevents the work station user from using the keyboard when the CONTROL1 record format is initially displayed.
- 8** HELP allows the user to press the Help key and sets indicator 99 on.
- 9** SFLMSG identifies the constant as a message that is displayed if indicator 99 is on.

In addition to the control information, the subfile control record format defines the constants to be used as column headings for the subfile record format. Refer to Figure 63 on page 157 for an example of the subfile control record format.

Multiple Device Files and Single Device Files

A **multiple device file** is either a display file or an intersystem communications function (ICF) file. A multiple device file can acquire more than one program device. For an example of the use of multiple device files, see Figure 64 on page 160.

A **single device file** is a device file created with only one program device defined for it. Printer files, diskette files and tape files are single device files. Display files and intersystem communication function (ICF) files created with a maximum number of one program device are also single device files.

A display file can have multiple program devices when the MAXDEV parameter of the CRTDSPF command is greater than 1. If you specify *NONE for the DEV parameter of this command, you must supply the name of a display device *before* you use any fields that are related to the file.

For more information about how to create and use a display file, see the *Data Management Guide*.

ICF files can have multiple program devices when the MAXPGMDEV parameter of the CRTICFF command is greater than 1. For more information about how to create and use ICF files, see the *ICF Programmer's Guide*.

COBOL determines at run time whether a file is a single device file or a multiple device file, based on whether the file is *capable* of having multiple devices. The actual number of devices acquired does not affect whether a file is considered a single or multiple device file. Whether a file is a single or a multiple device file is *not* determined at compilation time; this determination is based on the current description of the display or ICF file.

For multiple device files, if a particular program device is to be used in an I/O statement, that device is specified by the TERMINAL phrase. The TERMINAL phrase can also be specified for a single device file.

The following pages contain an example illustrating the use of multiple device files. The program uses a display file, and is intended to be run in batch mode. The program acquires terminals and invites those terminals using a sign-on display. After the terminals are invited, they are polled. If nobody signs on before the wait time expires, the program ends. If you enter a valid password, you are allowed to update an employee file by calling another COBOL program. Once the update is complete, the device is invited again and the terminals are polled again.

File Programmer	Date	Keying Instruction	Graphic Key	Description	Page of
--------------------	------	-----------------------	----------------	-------------	------------

Sequence Number	ESDS Type ADD/DEL/Comment (A/D/C)	Indicator Indic (0)	Indicator Indic (0)	Indicator Indic (0)	Indicator Indic (0)	Type of Name of Spec (M/R/N/J/N/S/O) Reserved	Name	Length	Reference (R)	Data Type/Keyboard Shift Data Decimal Positions	Usage (B/O//B//M//N//P)	Location		Functions	
												Line	Pos		
A*							DDS FOR THE MULTIPLE DEVICE DISPLAY FILE								
A*							R SIGNON							INVITE	
A												0	5	20	'bbbbbbbbbbbbbbbbbbbb'
A												0	6	20	'bb'
A												0	6	38	'bb'
A												0	7	20	'bb'
A												0	7	27	'M D F'
A												0	7	38	'bb'
A												0	8	20	'bb'
A												0	8	38	'bb'
A												0	9	20	'bbbbbbbbbbbbbbbbbbbb'
A												0	20	20	'PLEASE LOGON'
A							PASSWORD WRONG	10A	I			20	43	DSPATR(PC ND)	
A							R UPDATE	20A	O			21	43	DSPATR(HI)	
A												0	3	5	'UPDATE OF PERSONNEL FILE'
A												0	7	5	'TYPE IN EMPLOYEE NUMBER + TO BE UPDATED'
A							NUM	7A	I			7	44	DSPATR(RI PC)	
A							R EMPLOYEE					0	3	5	'EMPLOYEE NUMBER'
A							NUM	7A	B			3	25	DSPATR(PC)	
A							NAME	30A	B			5	25	DSPATR(PC)	
A												0	7	5	'EMPLOYEE ADDRESS'
A							STREET	30A	B			9	25	DSPATR(PC)	
A							APTNO	5A	B			11	25	DSPATR(PC)	
A							CITY	20A	B			13	25	DSPATR(PC)	
A							PROV	20A	B			15	25	DSPATR(PC)	
A							R RECOVERY					0	3	5	'THE EMPLOYEE NUMBER YOU + HAVE GIVEN IS INVALID'
A												0	6	5	'TYPE Y TO RETRY'
A							ANSWER	1X	I			8	5	DSPATR(RI PC)	
A												0	8	5	'TYPE N TO EXIT'
A												0	10	5	DSPATR(RI PC)
A															VALUES('Y' 'N')

1 The format SIGNON has the keyword INVITE associated with it. This means that, if format SIGNON is used in a WRITE statement, the device to which it is writing will be invited.

Figure 64 (Part 1 of 3). Example of the Use of Multiple Device Files


```

5738CB1 V2R2M0 001000          AS/400 COBOL Source          TESTER/SAMPMDF          AS400SYS 03/31/92 13:58:05      Page 2
STMT SEQNBR -A 1 B..+...2....+...3....+...4....+...5....+...6....+...7..IDENTFCN S COPYNAME  CHG DATE
 1 000010 IDENTIFICATION DIVISION.
 2 000020 PROGRAM-ID.          SAMPMDF.
 3 000030 AUTHOR.              PROGRAMMER NAME.
 000040
 000050*****
 000060* THE FOLLOWING PROGRAM DEMONSTRATES SOME OF THE FUNCTIONS *
 000070* AVAILABLE WITH MULTIPLE DEVICE FILE SUPPORT.          *
 000080*****
 000090
 4 000100 INSTALLATION. COBOL DEVELOPMENT CENTRE.
 5 000110 DATE-WRITTEN. 02/02/87.
 6 000120 DATE-COMPILED. 03/31/92 13:58:05
 7 000130 ENVIRONMENT DIVISION.
 8 000140 CONFIGURATION SECTION.
 9 000150 SOURCE-COMPUTER. IBM-AS400.
10 000160 OBJECT-COMPUTER. IBM-AS400.
11 000170 SPECIAL-NAMES. ATTRIBUTE-DATA IS ATTR. 1
12 000180 INPUT-OUTPUT SECTION.
13 000190 FILE-CONTROL.
14 000200     SELECT MULTIPLE-FILE
15 000210     ASSIGN TO WORKSTATION-MULT
16 000220     ORGANIZATION IS TRANSACTION 2
17 000230     ACCESS MODE IS SEQUENTIAL
18 000240     FILE STATUS IS MULTIPLE-FS1, MULTIPLE-FS2 3
19 000250     CONTROL-AREA IS MULTIPLE-CONTROL-AREA.
 000260     4
20 000270     SELECT TERMINAL-FILE
21 000280     ASSIGN TO DATABASE-TERM
22 000290     ORGANIZATION IS SEQUENTIAL
23 000300     ACCESS IS SEQUENTIAL
24 000310     FILE STATUS IS TERMINAL-FS1.
 000320
25 000330     SELECT PASSWORD-FILE
26 000340     ASSIGN TO DATABASE-PASSWORD
27 000350     ORGANIZATION IS INDEXED
28 000360     RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
29 000370     ACCESS MODE IS RANDOM
30 000380     FILE STATUS IS PASSWORD-FS1.
 000390
31 000400     SELECT PRINTER-FILE
32 000410     ASSIGN TO PRINTER-QPRINT.
33 000420 DATA DIVISION.
34 000430 FILE SECTION.
35 000440 FD MULTIPLE-FILE.
36 000450 01 MULTIPLE-REC. COPY DDS-SIGNON OF MULT. 5
37 +000001     05 MULT-RECORD PIC X(20).                                SIGNON
+000002* INPUT FORMAT:SIGNON FROM FILE MULT OF LIBRARY TESTER          SIGNON
+000003*                                SIGNON
38 +000004     05 SIGNON-I REDEFINES MULT-RECORD.                                SIGNON
39 +000005     06 PASSWORD PIC X(10). 6                                SIGNON
+000006* OUTPUT FORMAT:SIGNON FROM FILE MULT OF LIBRARY TESTER          SIGNON
+000007*                                SIGNON
40 +000008     05 SIGNON-0 REDEFINES MULT-RECORD.                                SIGNON
41 +000009     06 WRONG PIC X(20).                                SIGNON
 000460
42 000470 FD TERMINAL-FILE.
43 000480 01 TERMINAL-REC. COPY DDS-ALL-FORMATS OF TERM.
44 +000001     05 TERM-RECORD PIC X(10).                                <-ALL-FMTS
+000002* I-O FORMAT:TERM FROM FILE TERM OF LIBRARY TESTER          <-ALL-FMTS
+000003*                                <-ALL-FMTS
45 +000004     05 TERM REDEFINES TERM-RECORD.                                <-ALL-FMTS
46 +000005     06 TERM PIC X(10).                                <-ALL-FMTS
 000490
47 000500 FD PASSWORD-FILE.
48 000510 01 PASSWORD-REC. COPY DDS-ALL-FORMATS OF PASSWORD.
49 +000001     05 PASSWORD-RECORD PIC X(20).                                <-ALL-FMTS
+000002* I-O FORMAT:PASSWORDS FROM FILE PASSWORD OF LIBRARY TESTER          <-ALL-FMTS
+000003*                                <-ALL-FMTS
+000004*THE KEY DEFINITIONS FOR RECORD FORMAT PASSWORDS                                <-ALL-FMTS
+000005* NUMBER NAME RETRIEVAL TYPE ALTSEQ                                <-ALL-FMTS
+000006* 0001 PASSKEY ASCENDING AN NO                                <-ALL-FMTS
50 +000007     05 PASSWORDS REDEFINES PASSWORD-RECORD.                                <-ALL-FMTS
51 +000008     06 PASSKEY PIC X(10).                                <-ALL-FMTS
52 +000009     06 PASSWORD PIC X(10).                                <-ALL-FMTS
 000520

```

Figure 65 (Part 1 of 4). COBOL Source Listing for Multiple Device File Support

```

5738CB1 V2R2M0 001000          AS/400 COBOL Source          TESTER/SAMPMDF          AS400SYS 03/31/92 13:58:05          Page 2
STMT SEQNBR -A 1 B..+...2....+...3....+...4....+...5....+...6....+...7..IDENTFCN S COPYNAME CHG DATE
53 000530 FD PRINTER-FILE.
54 000540 01 PRINTER-REC.
55 000550 05 PRINTER-RECORD          PIC X(132).
000560
56 000570 WORKING-STORAGE SECTION.
000580
000590*****
000600*          DECLARE THE FILE STATUS FOR EACH FILE          *
000610*****
000620
57 000630 01 MULTIPLE-FS1          PIC X(2)          VALUE SPACES.
58 000640 01 MULTIPLE-FS2.          7
59 000650 05 MULTIPLE-MAJOR          PIC X(2)          VALUE SPACES.
60 000660 05 MULTIPLE-MINOR          PIC X(2)          VALUE SPACES.
61 000670 01 TERMINAL-FS1          PIC X(2)          VALUE SPACES.
62 000680 01 PASSWORD-FS1          PIC X(2)          VALUE SPACES.
000690
000700*****
000710*          DECLARE STRUCTURE FOR HOLDING FILE ATTRIBUTES          *
000720*****
000730
63 000740 01 STATION-ATTR.
64 000750 05 STATION-TYPE          PIC X(1).          8
65 000760 05 STATION-SIZE          PIC X(1).
66 000770 05 STATION-LOC          PIC X(1).
67 000780 05 FILLER          PIC X(1).
68 000790 05 STATION-ACQUIRE          PIC X(1).
69 000800 05 STATION-INVITE          PIC X(1).
70 000810 05 STATION-DATA          PIC X(1).
71 000820 05 STATION-STATUS          PIC X(1).
72 000830 05 STATION-DISPLAY          PIC X(1).
73 000840 05 STATION-KEYBOARD          PIC X(1).
74 000850 05 STATION-SIGNON          PIC X(1).
75 000860 05 FILLER          PIC X(5).
000870
000880*****
000890*          DECLARE THE CONTROL AREA FOR MULTIPLE-FILE          *
000900*****
000910
76 000920 01 MULTIPLE-CONTROL-AREA.
77 000930 05 MULTIPLE-KEY-FEEDBACK PIC X(2)          VALUE SPACES.
78 000940 05 MULTIPLE-DEVICE-NAME PIC X(10)         VALUE SPACES.
79 000950 05 MULTIPLE-FORMAT-NAME PIC X(10)         VALUE SPACES.
000960
000970*****
000980*          DECLARE ERROR REPORT VARIABLES          *
000990*****
001000
80 001010 01 HEADER-LINE.
81 001020 05 FILLER          PIC X(60)          VALUE SPACES.
82 001030 05 FILLER          PIC X(72)
83 001040          VALUE "MDF ERROR REPORT".
84 001050 01 DETAIL-LINE.
85 001060 05 FILLER          PIC X(15)          VALUE SPACES.
86 001070 05 DESCRIPTION          PIC X(25)          VALUE SPACES.
87 001080 05 DETAIL-VALUE          PIC X(92)          VALUE SPACES.
001090
001100*****
001110*          DECLARE COUNTERS, FLAGS AND STORAGE VARIABLES          *
001120*****
001130
88 001140 01 CURRENT-TERMINAL          PIC X(10)          VALUE SPACES.
89 001150 01 TERMINAL-ARRAY.
90 001160 05 LIST-OF-TERMINALS OCCURS 250 TIMES.
91 001170 07 DEVICE-NAME          PIC X(10).
92 001180 01 COUNTER          PIC 9(3)          VALUE IS 1.
93 001190 01 NO-OF-TERMINALS          PIC 9(3)          VALUE IS 1.
94 001200 01 TERMINAL-LIST-FLAG          PIC 1.
95 001210 88 END-OF-TERMINAL-LIST          VALUE IS B"1".
96 001220 88 NOT-END-OF-TERMINAL-LIST          VALUE IS B"0".
97 001230 01 NO-DATA-FLAG          PIC 1.
98 001240 88 NO-DATA-AVAILABLE          VALUE IS B"1".
99 001250 88 DATA-AVAILABLE          VALUE IS B"0".
001260

```

Figure 65 (Part 2 of 4). COBOL Source Listing for Multiple Device File Support

```

5738CB1 V2R2M0 001000          AS/400 COBOL Source          TESTER/SAMPMDF          AS400SYS 03/31/92 13:58:05      Page 2
STMT SEQNBR -A 1 B.+. . . . 2. . . . 3. . . . 4. . . . 5. . . . 6. . . . 7. . . . IDENTFCN S COPYNAME CHG DATE
100 001270 PROCEDURE DIVISION.
    001280
    001290 DECLARATIVES.
    001300
    001310 MULTIPLE-SECTION SECTION.
    001320     USE AFTER STANDARD EXCEPTION PROCEDURE ON MULTIPLE-FILE.
    001330
    001340 MULTIPLE-PARAGRAPH.
101 001350     WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
102 001360     MOVE "FILE NAME IS:" TO DESCRIPTION OF DETAIL-LINE.
103 001370     MOVE "MULTIPLE FILE" TO DETAIL-VALUE OF DETAIL-LINE.
104 001380     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
105 001390     MOVE "FILE STATUS IS:" TO DESCRIPTION OF DETAIL-LINE.
106 001400     MOVE MULTIPLE-FS1 TO DETAIL-VALUE OF DETAIL-LINE.
107 001410     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
108 001420     MOVE "EXTENDED STATUS IS:" TO DESCRIPTION OF DETAIL-LINE. 9
109 001430     MOVE MULTIPLE-FS2 TO DETAIL-VALUE OF DETAIL-LINE.
110 001440     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
111 001450     ACCEPT STATION-ATTR FROM ATTR. 9A
112 001460     MOVE "FILE ATTRIBUTES ARE:" TO DESCRIPTION OF DETAIL-LINE.
113 001470     MOVE STATION-ATTR TO DETAIL-VALUE OF DETAIL-LINE.
114 001480     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
115 001490     STOP RUN.
    001500
    001510 TERMINAL-SECTION SECTION.
    001520     USE AFTER STANDARD EXCEPTION PROCEDURE ON TERMINAL-FILE.
    001530 TERMINAL-PARAGRAPH.
116 001540     WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
117 001550     MOVE "FILE NAME IS:" TO DESCRIPTION OF DETAIL-LINE.
118 001560     MOVE "TERMINAL FILE" TO DETAIL-VALUE OF DETAIL-LINE.
119 001570     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
120 001580     MOVE "FILE STATUS IS:" TO DESCRIPTION OF DETAIL-LINE.
121 001590     MOVE TERMINAL-FS1 TO DETAIL-VALUE OF DETAIL-LINE.
122 001600     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
123 001610     STOP RUN.
    001620
    001630 PASSWORD-SECTION SECTION.
    001640     USE AFTER STANDARD EXCEPTION PROCEDURE ON PASSWORD-FILE.
    001650 PASSWORD-PARAGRAPH.
124 001660     WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
125 001670     MOVE "FILE NAME IS:" TO DESCRIPTION OF DETAIL-LINE.
126 001680     MOVE "PASSWORD FILE" TO DETAIL-VALUE OF DETAIL-LINE.
127 001690     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
128 001700     MOVE "FILE STATUS IS:" TO DESCRIPTION OF DETAIL-LINE.
129 001710     MOVE PASSWORD-FS1 TO DETAIL-VALUE OF DETAIL-LINE.
130 001720     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
131 001730     STOP RUN.
    001740
    001750 END DECLARATIVES.
    001760
    001770*****
    001780*          MAIN PROGRAM LOGIC BEGINS HERE          *
    001790*****
    001800
    001810 MAIN-LINE SECTION.
    001820 MAIN-LINE-PARAGRAPH.
132 001830     OPEN I-0 MULTIPLE-FILE 10
    001840         INPUT TERMINAL-FILE
    001850         I-0 PASSWORD-FILE
    001860         OUTPUT PRINTER-FILE.
    001870
133 001880     MOVE 1 TO COUNTER.
134 001890     SET NOT-END-OF-TERMINAL-LIST TO TRUE.
    001900     PERFORM
135 001910         FILL-TERMINAL-LIST UNTIL END-OF-TERMINAL-LIST.
    001920     PERFORM
136 001930         ACQUIRE-AND-INVITE-TERMINALS
    001940             VARYING COUNTER FROM 1 BY 1
    001950             UNTIL COUNTER GREATER THAN NO-OF-TERMINALS.
137 001960     MOVE 1 TO COUNTER.
138 001970     SET DATA-AVAILABLE TO TRUE.
    001980     PERFORM
139 001990         POLL-TERMINALS UNTIL NO-DATA-AVAILABLE.
    002000     PERFORM
140 002010         DROP-TERMINALS
    002020             VARYING COUNTER FROM 1 BY 1
    002030             UNTIL COUNTER GREATER THAN NO-OF-TERMINALS.

```

Figure 65 (Part 3 of 4). COBOL Source Listing for Multiple Device File Support


```

5738CB1 V2R2M0 001000 AS/400 COBOL Source TESTER/SAMPMDF AS400SYS 03/31/92 13:58:05 Page 2
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME CHG DATE
141 002040 CLOSE MULTIPLE-FILE
002050 TERMINAL-FILE
002060 PASSWORD-FILE
002070 PRINTER-FILE.
142 002080 STOP RUN.
002090
002100*****
002110* PROCEDURES *
002120*****
002130
002140 PROCEDURE-SECTION SECTION.
002150 FILL-TERMINAL-LIST.
143 002160 READ TERMINAL-FILE RECORD INTO LIST-OF-TERMINALS(COUNTER)
002170 AT END
144 002180 SET END-OF-TERMINAL-LIST TO TRUE
145 002190 SUBTRACT 1 FROM COUNTER
146 002200 MOVE COUNTER TO NO-OF-TERMINALS.
147 002210 ADD 1 TO COUNTER.
002220
002230 ACQUIRE-AND-INVITE-TERMINALS.
148 002240 ACQUIRE LIST-OF-TERMINALS(COUNTER) FOR MULTIPLE-FILE. 11
149 002250 WRITE MULTIPLE-REC 12
002260 FORMAT IS "SIGNON"
002270 TERMINAL IS LIST-OF-TERMINALS(COUNTER).
002280
002290 POLL-TERMINALS.
150 002300 READ MULTIPLE-FILE RECORD. 13
151 002310 IF MULTIPLE-FS2 EQUAL "310" THEN
152 002320 SET NO-DATA-AVAILABLE TO TRUE. 14
153 002330 IF DATA-AVAILABLE THEN
154 002340 MOVE MULTIPLE-DEVICE-NAME TO CURRENT-TERMINAL
155 002350 PERFORM PASSWORD-VALIDATION. 15
002360
002370 PASSWORD-VALIDATION.
156 002380 MOVE CURRENT-TERMINAL TO PASSKEY OF PASSWORD-REC.
157 002390 READ PASSWORD-FILE RECORD.
158 002400 IF PASSWORD OF SIGNON-I EQUAL PASSWORD OF PASSWORD-REC THEN
159 002410 CALL "UPDT" USING CURRENT-TERMINAL
160 002420 MOVE SPACES TO WRONG OF SIGNON-0
002430 ELSE
161 002440 MOVE "INVALID PASSWORD" TO WRONG OF SIGNON-0.
162 002450 WRITE MULTIPLE-REC
002460 FORMAT IS "SIGNON"
002470 TERMINAL IS CURRENT-TERMINAL.
002480
002490 DROP-TERMINALS.
163 002500 DROP LIST-OF-TERMINALS(COUNTER) FROM MULTIPLE-FILE. 16
***** END OF SOURCE *****

```

Figure 65 (Part 4 of 4). COBOL Source Listing for Multiple Device File Support

Device File Attributes

- 1 ATTR is the mnemonic-name associated with the function-name ATTRIBUTE-DATA. ATTR is used in the ACCEPT statement to obtain attribute data for the TRANSACTION file MULTIPLE-FILE. See item 9A.
- 2 File MULT must have been created using the CRTDSPF command, where the DEV parameter has a value of *NONE and the MAXDEV parameter has a value greater than 1. The WAITRCD parameter specifies the wait time for READ operations on the file. The WAITRCD parameter must have a value greater than 0.
- 3 MULTIPLE-FS2 is the extended file status for the TRANSACTION file MULTIPLE-FILE. This variable has been declared in the WORKING-STORAGE section of the program. See item 7.
- 4 MULTIPLE-CONTROL-AREA is the control area for the TRANSACTION file MULTIPLE-FILE. This variable is used to determine which program device was used to sign on. See item 15.

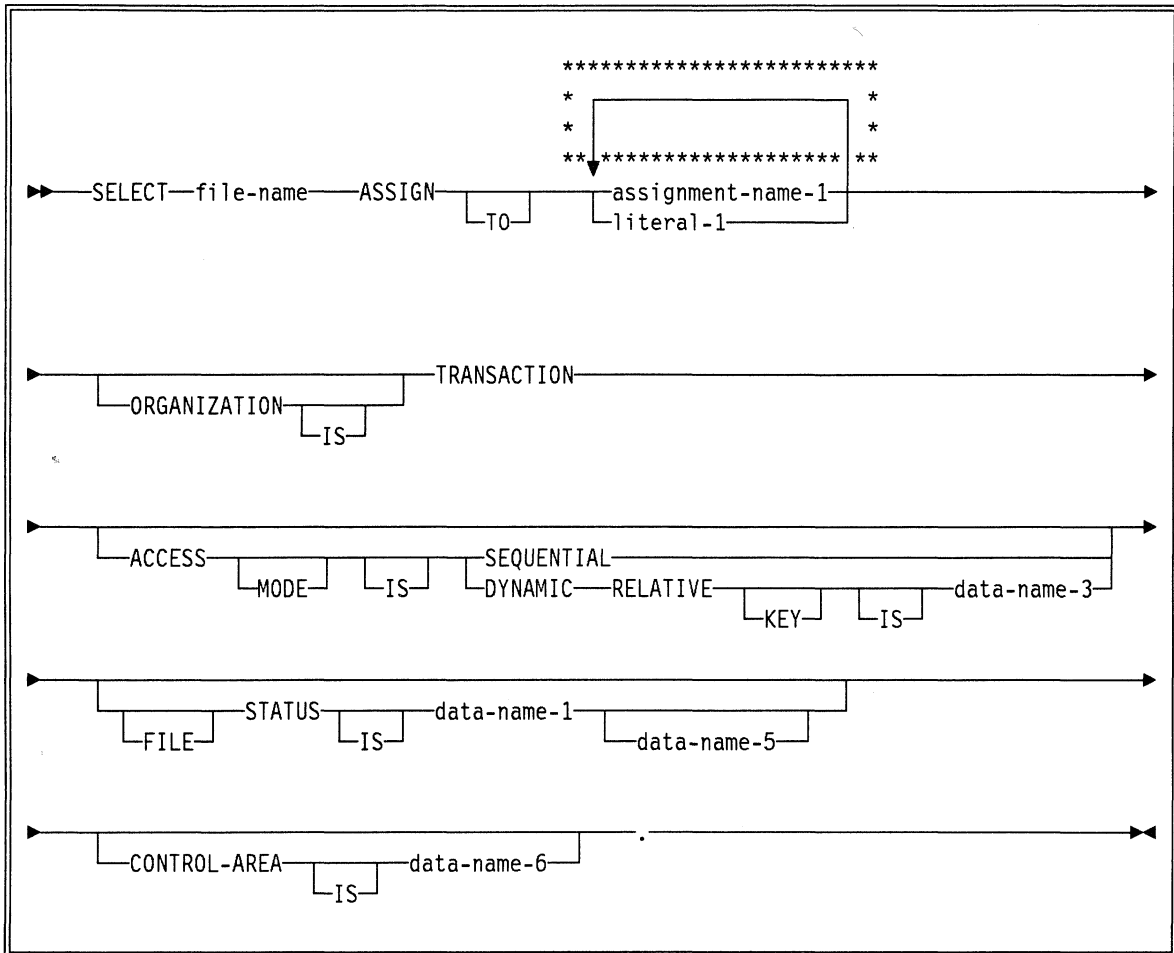
- 5** The data description for MULTIPLE-REC has been defined using the COPY DDS statement.
- Note:** Only the fields that are copied are named fields. Refer to the DDS of this example for comments regarding the DDS used.
- 6** Format SIGNON is the format with the INVITE keyword. This is the format that will be used to invite devices via the WRITE statement.
- 7** This is the declaration for the extended file-status MULTIPLE-FS2. It is a 4-byte field that is subdivided into a major return code (first 2 bytes) and a minor return code (last 2 bytes).
- 8** STATION-ATTR is where the ACCEPT statement contains the attribute data for the TRANSACTION file MULTIPLE-FILE. See item **9A**.
- 9** In this statement, the extended file status MULTIPLE-FS2 is being written.
- 9A** This is an example of accepting attribute-data for the TRANSACTION file MULTIPLE-FILE. Because there is no interest in a specific program device, but rather the last program device used, the FOR phrases are not used with the ACCEPT.
- 10** This statement opens the TRANSACTION file MULTIPLE-FILE. Because the ACQPGMDEV parameter of the CRTDSPF command has the value *NONE, no program devices are implicitly acquired when this file is opened.
- 11** This statement acquires the program device contained in the variable LIST-OF-TERMINALS (COUNTER), for the TRANSACTION file MULTIPLE-FILE.
- 12** This WRITE statement is inviting the program device specified in the TERMINAL phrase. The format SIGNON has the DDS keyword INVITE associated with it. Refer to item **13**.
- 13** This READ statement will read from any invited program device. See item **12**. If the wait time expires before anyone inputs to the invited devices, the extended file status will be set to "0310" and processing will continue. See item **14**.
- 14** In this statement, the extended file status for MULTIPLE-FILE is being checked to see if the wait time expired.
- 15** The program device name stored in the control area is used to determine which program device was used to sign on. See item **4**.
- 16** This DROP statement detaches the program device contained in the variable LIST-OF-TERMINALS from the TRANSACTION file MULTIPLE-FILE.

Environment Division

File-Control Entry

The TRANSACTION file must be named by a file-control entry in the FILE-CONTROL paragraph. This entry also specifies other information related to the file.

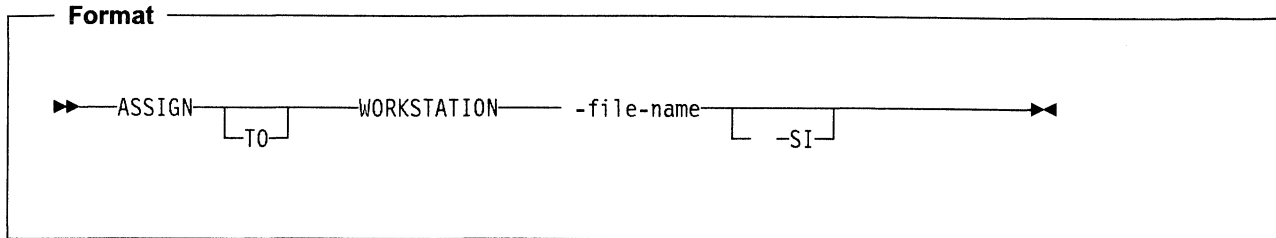
Format



ASSIGN Clause

The ASSIGN clause associates the TRANSACTION file with a display file or ICF file through the use of assignment-name-1.

Assignment-name-1 has the following structure:



Device specifies the type of device associated with the file. The value must be WORKSTATION.

The AS/400 file name is a one-to-ten character external name of the display file or ICF file specified on the create device file commands, CRTDSPF or CRTICFF.

The attribute -SI is used to specify the file level option for a separate indicator area. See “Using Indicators with Transaction Files” on page 138 for further details.

The second and subsequent assignment-names are syntax-checked, but are treated as documentation.

ORGANIZATION Clause

The ORGANIZATION clause specifies the logical structure of a file. TRANSACTION organization signifies interaction between the program and either a work station user or another system.

TRANSACTION Organization: TRANSACTION processing is defined as the random arrival of a record from one of multiple possible sources followed by appropriate processing, and finally, by the output of results or feedback information of some type to the source of the record.

In some cases, all records are homogeneous; that is, a logical transaction is completed with one exchange of records. In other situations, a series of records is passed back and forth in a logical progression with various record types either being selected by the initiator or as part of the processing based on input data values.

Each transaction can be processed by a different program, or multiple transactions can be processed by the same program, depending on the system environment.

The initiation of a transaction can cause a program to be scheduled to process the transaction.

A transaction can consist of a series of alternating requests and responses (a dialogue). Each request and response can consist of multiple logical records.

ACCESS MODE Clause

For files with TRANSACTION organization, the access mode can be SEQUENTIAL or DYNAMIC.

Note: **Dynamic processing** is a method of reading from or writing to a file in a nonsequential order and reading from a file in a sequential order with the same OPEN statement.

When ACCESS IS SEQUENTIAL is specified or implied, the format name contained in the format name field of the control area specifies which record was accessed. When ACCESS IS SEQUENTIAL is specified for a TRANSACTION file, do not specify the RELATIVE KEY data item.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request. Random accessing of a TRANSACTION file is only valid if subfile processing is being performed. For subfile processing, you *must* specify ACCESS IS DYNAMIC.

RELATIVE KEY Clause

The RELATIVE KEY clause specifies the relative record number for a specific record in a subfile. The RELATIVE KEY data item, data-name-3, must be defined as an unsigned integer and cannot be scaled. Also, the data item must not be defined in a record description entry associated with the TRANSACTION file.

FILE STATUS Clause

Data-name-5 identifies the extended-file-status data item, which contains major and minor return codes. These major and minor return codes can, in some cases, indicate I/O errors when the file status code does not. After an I/O operation is performed on an unopened file, the extended file status will have a value of zeros.

For more information about the FILE STATUS clause, refer to “File Status and Feedback Areas” on page 101. General considerations about the FILE STATUS clause and data-name-1 are described in Part 2 of the *COBOL/400* Reference* in the section, “FILE STATUS Clause.”

For information about the role of file status in error handling, refer to Chapter 6, “COBOL/400 Exception and Error Handling” on page 67.

Data-name-5 must be defined in the Data Division as a 4-byte alphanumeric data item, and must *not* be defined in the File Section. The first 2 bytes of the extended-file-status data item contain the major return code, and the second 2 bytes contain the minor return code. Return codes are moved into data-name-5 after any input or output operation (except the ACCEPT or CLOSE statement) on the TRANSACTION file. The values placed in data-name-5 can also be accessed by the ACCEPT statement using the I-O-FEEDBACK function-name. For more information about the major and minor return codes, see the *Data Management Guide* and the *ICF Programmer’s Guide*.

CONTROL-AREA Clause

The CONTROL-AREA clause specifies device-dependent and system-dependent information that is used to control input/output operations for TRANSACTION files.

Data-name-6 is a CONTROL-AREA data item that must be defined in the LINKAGE SECTION or WORKING-STORAGE SECTION. Data-name-6 is assumed to have the following format:

```
01 data-name-6.  
  02 function-key PIC X(2).  
      (Function key feedback field)  
  02 device-name PIC X(10).  
      (Program device name)  
  02 record-format PIC X(10).  
      (Record format)
```

Data-name-6 must be 2, 12, or 22 characters long. Based upon the length of data-name-6, the compiler assumes the availability of key feedback bytes, the program device name, and record format.

Programming Note: For an ICF file, the actual name of a device may be different from the program device name (data-name-11).

Information is moved into data-name-6 for each READ operation from a file that has been assigned to a WORKSTATION device type. The information is valid only if the READ operation is successfully completed (provided the wait time has not expired). The information is in the fixed format as shown in the following example:

```
FILE-CONTROL.  
SELECT SCREEN-FILE  
  ASSIGN TO WORKSTATION-MYFMTS  
  ORGANIZATION IS TRANSACTION  
  CONTROL-AREA IS  
  TRANSACTION-CONTROL-AREA.  
:  
WORKING-STORAGE SECTION.  
01 TRANSACTION-CONTROL-AREA.  
* FEEDBACK ITEM  
  02 FUNCTION-KEY PIC XX.  
  02 TERMINAL-ID PIC X(10).  
  02 FORMAT-NAME PIC X(10).
```

Each field in the TRANSACTION-CONTROL-AREA data item in the example is described as follows:

- FUNCTION-KEY: A two-digit number inserted in the field by the work station interface that identifies the function key the operator pressed to initiate the transaction. The codes are as follows:

00	Enter key
01-24	Function keys 1 through 24
90	Roll Up/Page Down key
91	Roll Down/Page Up key
92	Print key
93	Help key
94	Clear key
95	Home key
99	Undefined

Any function keys for which feedback information is desired must be defined for the display file using DDS.

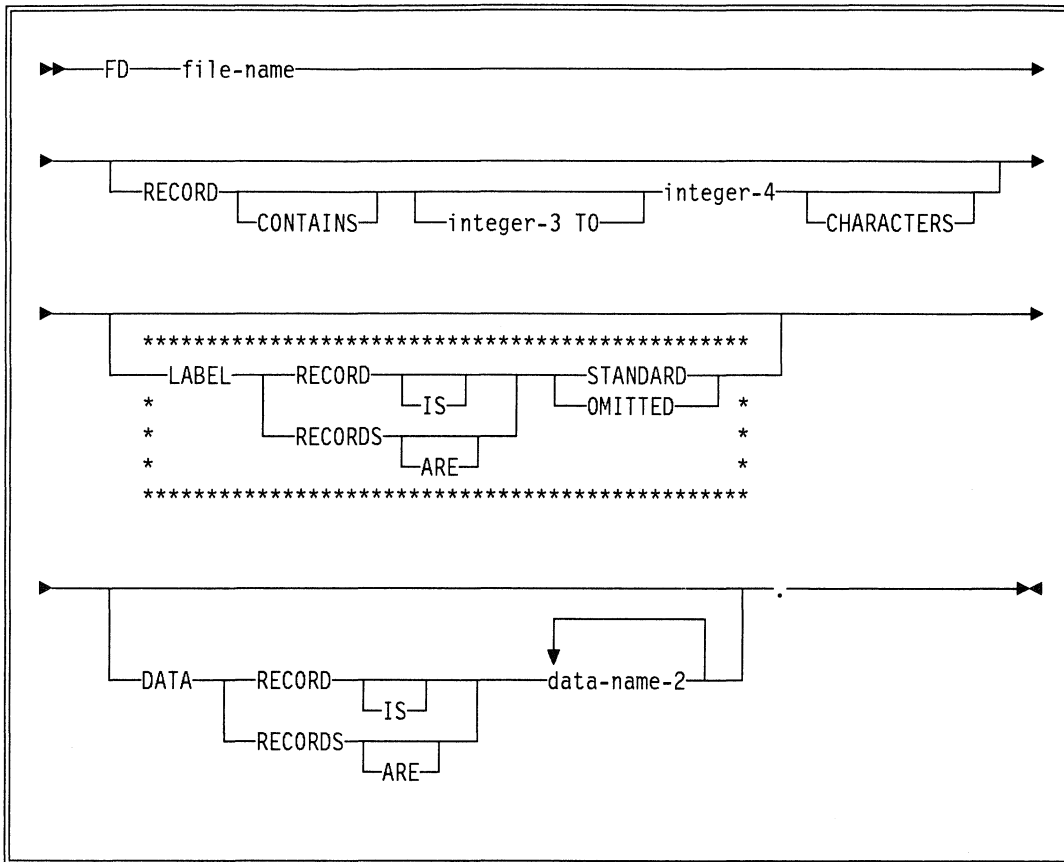
- TERMINAL-ID: The *program device name*.
- FORMAT-NAME: The DDS record format name that was referenced by the last I/O statement run.

Data Division

File Description Entry

A file description entry consists of a level indicator (FD), a file name, and a series of independent clauses. For a TRANSACTION file, the independent clauses allowed are the RECORD CONTAINS clause, the LABEL RECORDS clause, and the DATA RECORDS clause.

Format



The LABEL RECORDS clause specifies whether or not labels are present. This clause is required in every file description entry. This clause is syntax-checked, but is treated as documentation.

Boolean Data Items

The use of Boolean data and the use of indicators are described under “Data Description Entry – Boolean Data” on page 140.

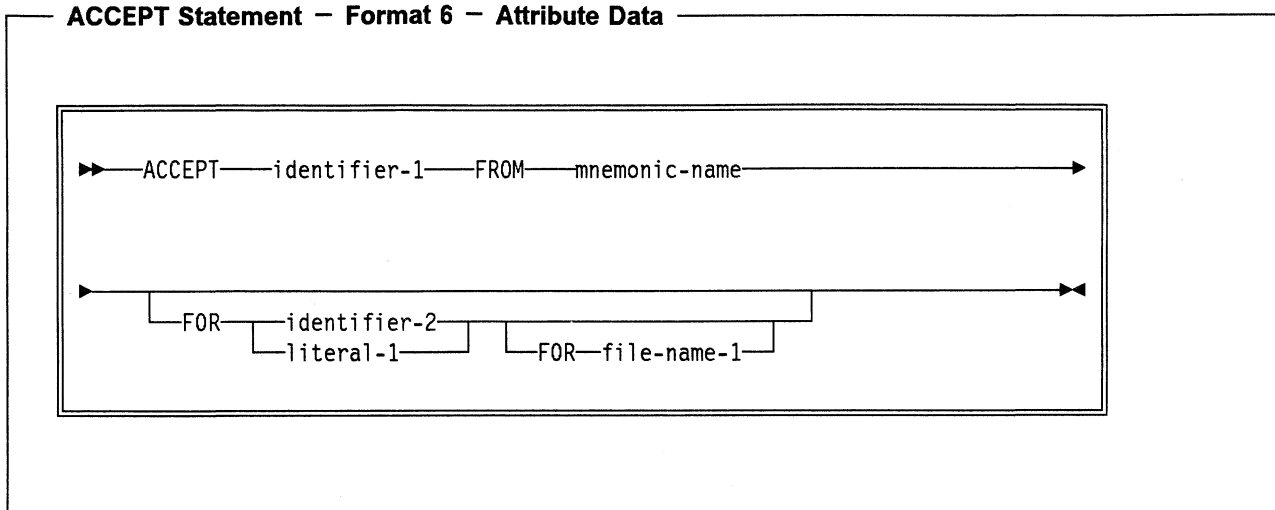
Procedure Division

Procedure Division Concepts

The COBOL/400 language provides a number of extensions to PROCEDURE DIVISION statements to support TRANSACTION processing. The sections that follow describe the statements involved and their usage.

ACCEPT Statement

The ACCEPT statement retrieves information (attribute data) about a particular program device associated with a TRANSACTION file.



This format of the ACCEPT statement can only be used for files with an organization of TRANSACTION. Mnemonic-name must be associated with the function-name ATTRIBUTE-DATA in the SPECIAL-NAMES paragraph.

If file-name is not specified, the default file for the ACCEPT statement is the first TRANSACTION file specified in a SELECT clause of the FILE-CONTROL paragraph.

Literal-1 or the contents of identifier-2, if specified, indicates the program device name for which attribute data is made available. This device must be defined by a CRTDSPF, ADDICFDEVE, or OVRICFDEVE CL command. The device does not actually have to be acquired. Literal-1, if specified, must be nonnumeric and 10 characters or fewer in length. The contents of identifier-2, if specified, must be an alphanumeric data item 10 characters or fewer in length. If an incorrect program device name is specified, or if the file is not open at the time the ACCEPT statement is processed, message LBE7205

ACCEPT ATTRIBUTE-DATA statement has failed (C D F).

is issued and processing terminates.

If both FOR phrases are omitted (indicating the default TRANSACTION file is being used), the ACCEPT statement uses the program device from which a READ, WRITE, REWRITE, or ACCEPT (Attribute Data) operation on the default file was most recently performed. If the only prior operation on the file was an OPEN, the ACCEPT statement uses the program device implicitly acquired by the file when the file was opened. When both FOR phrases are omitted, a program device must have been acquired to use this particular format of the ACCEPT statement.

Program device attributes are moved into identifier-1 from the appropriate attribute data format, according to the rules for a group MOVE without the CORRESPONDING phrase.

You can make use of multiple display files along with ordinary files in a program that includes an Extended ACCEPT or Extended DISPLAY statement. (See the *COBOL/400* Reference* for more information.)

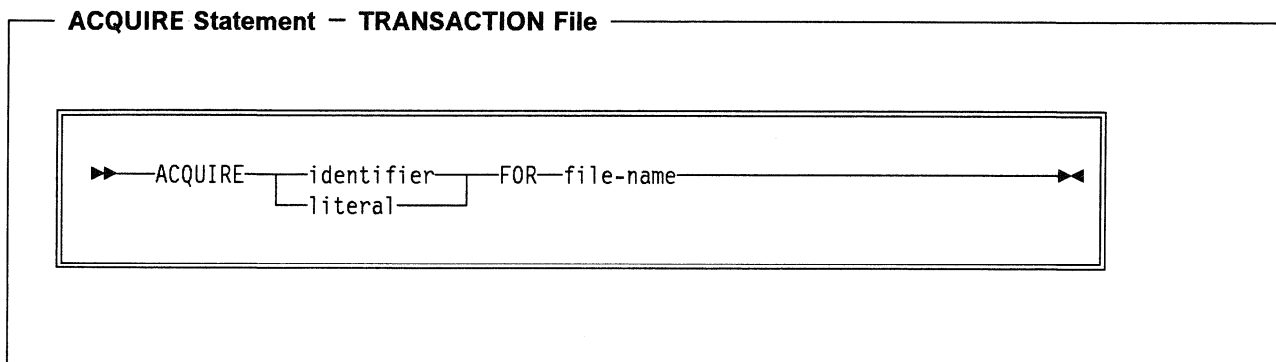
Attribute Data Formats

The attribute data retrieved by the ACCEPT statement has two different formats, depending if the data is for a work station or for a communications device.

The ATTRIBUTE-DATA mnemonic name can be used *only* to obtain information about a program device for a TRANSACTION file. Attribute data does *not* provide information about the status of a completed or attempted I/O operation. To obtain information about I/O operations, use the Format 3 ACCEPT statement with the I-O-FEEDBACK or OPEN-FEEDBACK mnemonic names. For more information about these mnemonic names, see the "SPECIAL NAMES Paragraph" section of the *COBOL/400* Reference*.

ACQUIRE Statement

The ACQUIRE statement acquires a program device for a TRANSACTION file.



Literal or the contents of identifier indicates the program device name to be acquired by the specified file. Literal, if specified, must be nonnumeric and 10 characters or fewer in length. Identifier, if specified, must refer to an alphanumeric data item 10 characters or fewer in length.

File-name must be the name of a file with an organization of TRANSACTION, and the file must be open when the ACQUIRE statement is run. A compilation error message is issued if the organization is not TRANSACTION.

For a description of conditions that must be met before a communications device can be acquired, see the *ICF Programmer's Guide*. For more information about the requirements for displays, see the *Data Management Guide*.

Successful completion of the ACQUIRE operation makes the program device available for input and output operations.

If the ACQUIRE operation is unsuccessful, the file status value is set to 9H and the USE AFTER EXCEPTION/ERROR procedure is called (if specified). For more information, refer to Chapter 6, "COBOL/400 Exception and Error Handling."

Only one program device can be implicitly acquired when a file is opened. If a file is an ICF file, the single implicitly acquired program device is determined by the ACQPGMDEV parameter of the CRTICFF command. If the file is a display

file, the single implicitly acquired program device is determined by the first entry in the DEV parameter of the CRTDSPF command. Additional program devices *must* be explicitly acquired.

A program device is explicitly acquired by using the ACQUIRE statement. For an ICF file, that device must have been defined to the file with the ADDICFDEVE or OVRICFDEVE CL command before the file was opened. For display files there is no such requirement. That is, the device named in the ACQUIRE statement does not have to be specified in the DEV parameter of the CRTDSPF command, the CHGDSPF command, or the OVRDSPF command. For a display file, the program device name must match the display device.

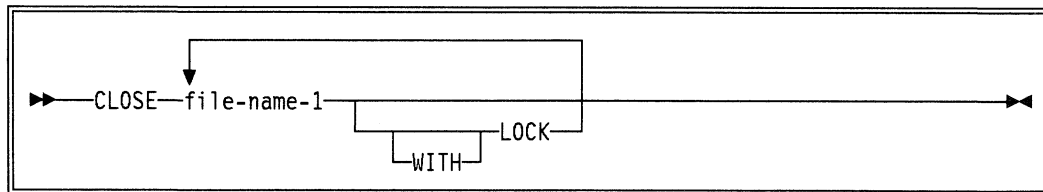
The ACQUIRE statement can also be used as an aid in recovering from I/O errors. For more information, see the "ACQUIRE Statement" section of the *COBOL/400* Reference*.

For more information about these commands, see the *CL Reference*.

CLOSE Statement

The CLOSE statement terminates the processing of volumes and files, with optional lock where applicable.

CLOSE Statement – Format 3 – TRANSACTION File

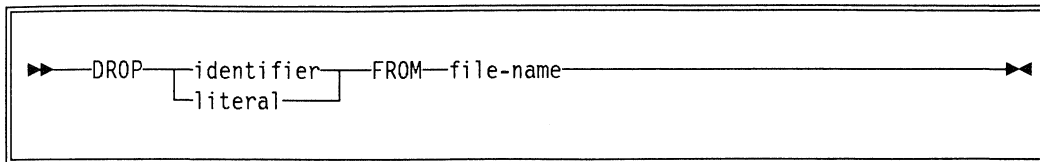


For a detailed discussion of the CLOSE statement, see the "CLOSE Statement" section of the *COBOL/400* Reference*.

DROP Statement

The DROP statement releases a program device that has been acquired by a TRANSACTION file.

DROP Statement



Literal or the contents of identifier indicates the program device name of the device to be dropped. Literal, if specified, must be nonnumeric and 10 characters or fewer in length. Identifier, if specified, must refer to an alphanumeric data item, 10 characters or fewer in length.

File-name must refer to a file with an organization of TRANSACTION, and the file must be open to be used in the DROP statement. If no DROP statement is issued, program devices attached to a TRANSACTION file are implicitly released when that file is finally closed.

Program devices specified in a DROP statement must have been acquired by the TRANSACTION file, either through an explicit ACQUIRE or through an implicit ACQUIRE at OPEN time.

After successful running of the DROP statement, the program device is no longer available for input or output operations through the TRANSACTION file. The device can be reacquired if necessary. The contents of the record area associated with a released program device are no longer available, even if the device is reacquired.

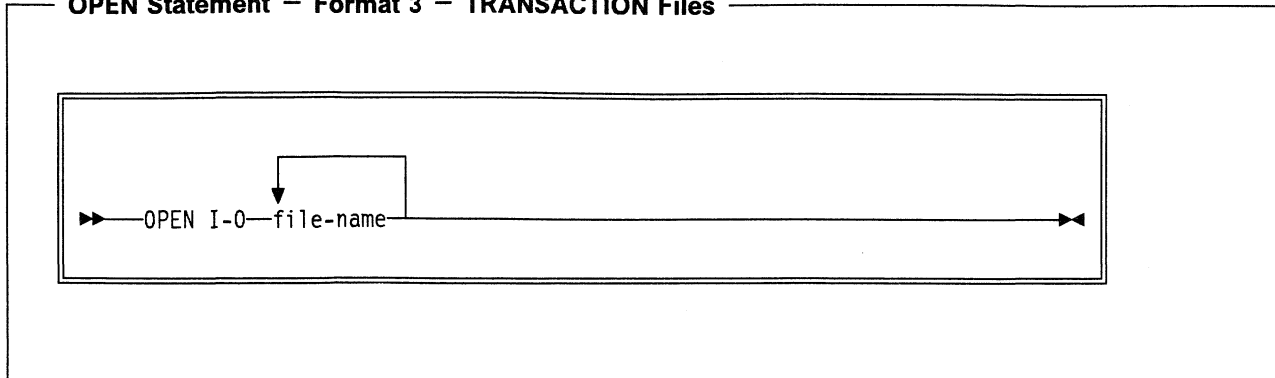
If the DROP operation is unsuccessful, the USE AFTER EXCEPTION/ERROR procedure is processed (if specified). For more information, refer to Chapter 6, "COBOL/400 Exception and Error Handling."

The DROP statement can also be used as an aid in recovering from I/O errors. For more information, see the "DROP Statement" section of the *COBOL/400* Reference*.

OPEN Statement

The OPEN statement initiates the processing of files.

OPEN Statement – Format 3 – TRANSACTION Files



A TRANSACTION file must be opened in the I/O mode. For a further discussion of the OPEN statement, see the *COBOL/400* Reference*.

The OPEN statement can cause a program device to be implicitly acquired for a TRANSACTION file. For a further discussion about the acquiring of program devices, see the "ACQUIRE Statement" on page 174.

Common Processing Facilities

The following discussion on FORMAT, INDICATORS, SUBFILE, and TERMINAL phrases relates to the READ, REWRITE, and WRITE statements.

FORMAT Phrase

The literal or identifier specified must be a character string of 10 characters or fewer in length.

Multiple data records, each with a different format, can be concurrently active for a TRANSACTION file. If the FORMAT phrase is specified, it must specify a valid format name that is defined to the system, and the I/O operation must be performed on a data record of the same format. If the format is an invalid name or if it does not exist, the FILE STATUS data item, if specified, is set to a value of 9K and the contents of the record area are undefined.

DB-FORMAT-NAME Special Register: After the running of an input/output statement for a TRANSACTION file, the DB-FORMAT-NAME special register is modified according to the following rules:

- If the input/output operation is successful, the record format name is implicitly moved to the special register after completion of the input/output operation.
- If the input/output operation is unsuccessful, DB-FORMAT-NAME contains the record format name used in the last successful input/output operation.

When the FORMAT phrase is not specified, DB-FORMAT-NAME can be used if the file contains a default record format name. The default value is always moved to the DB-FORMAT-NAME special register.

DB-FORMAT-NAME is implicitly defined as PICTURE X(10).

INDICATORS Phrase

The identifier specified in the INDICATORS phrase must be either an elementary Boolean data item specified without the OCCURS clause or a group item that has elementary Boolean data items subordinate to it.

When a data record is written or rewritten, indicators can be written or rewritten with it. The indicators can control how the record is displayed and the various data management functions.

When a data record is read, indicators can be read with it. The indicators can be used to pass information about the data record and how it was entered into your program.

By defining a format using DDS, you determine what functions are to be controlled by indicators, and which indicators control a particular function.

For detailed information on the INDICATORS phrase, refer to "Using Indicators with Transaction Files" on page 138.

SUBFILE Phrase

When the SUBFILE phrase is specified, it indicates that all formats referenced by the statement are subfiles. When SUBFILE is not specified in a TRANSACTION I/O statement, it indicates that none of the formats referenced by the statement are subfiles. This information is not verified at compilation time. If it is specified incorrectly, the subfile is processed as a series of input/output operations directly to the display device. When the specified format name exists as a display file format, the READ/WRITE operations complete successfully.

When SUBFILE is not specified, the RELATIVE KEY data item associated with the file, if specified, is not referenced or changed by the I/O operation.

When SUBFILE is specified, a RELATIVE KEY data item must be defined for the file. Its value is referenced, and sometimes changed, by the I/O operation. See each of the statements associated with SUBFILE operations for a detailed description of when and how the RELATIVE KEY data item is changed.

The SUBFILE phrase can be specified only for display files.

TERMINAL Phrase

When the TERMINAL phrase is specified, it indicates a specific program device is to be used for a READ, WRITE, or REWRITE operation on a TRANSACTION file.

The TERMINAL phrase can be omitted for I/O operations on single device files, because that device is always used.

If the TERMINAL phrase is omitted for an I/O operation on a TRANSACTION file that has acquired multiple program devices, the program device that last attempted a READ, WRITE, REWRITE, ACQUIRE, DROP, or ACCEPT (Attribute Data) operation on the file is used. If the only prior operation on the file was an OPEN, the default program device used is the program device implicitly acquired by the TRANSACTION file when the file was opened. A run-time error message occurs if no program device has been acquired when the file is opened.

For a READ statement with both the TERMINAL phrase and the NO DATA phrase specified, the imperative-statement in the NO DATA phrase is run only if data is

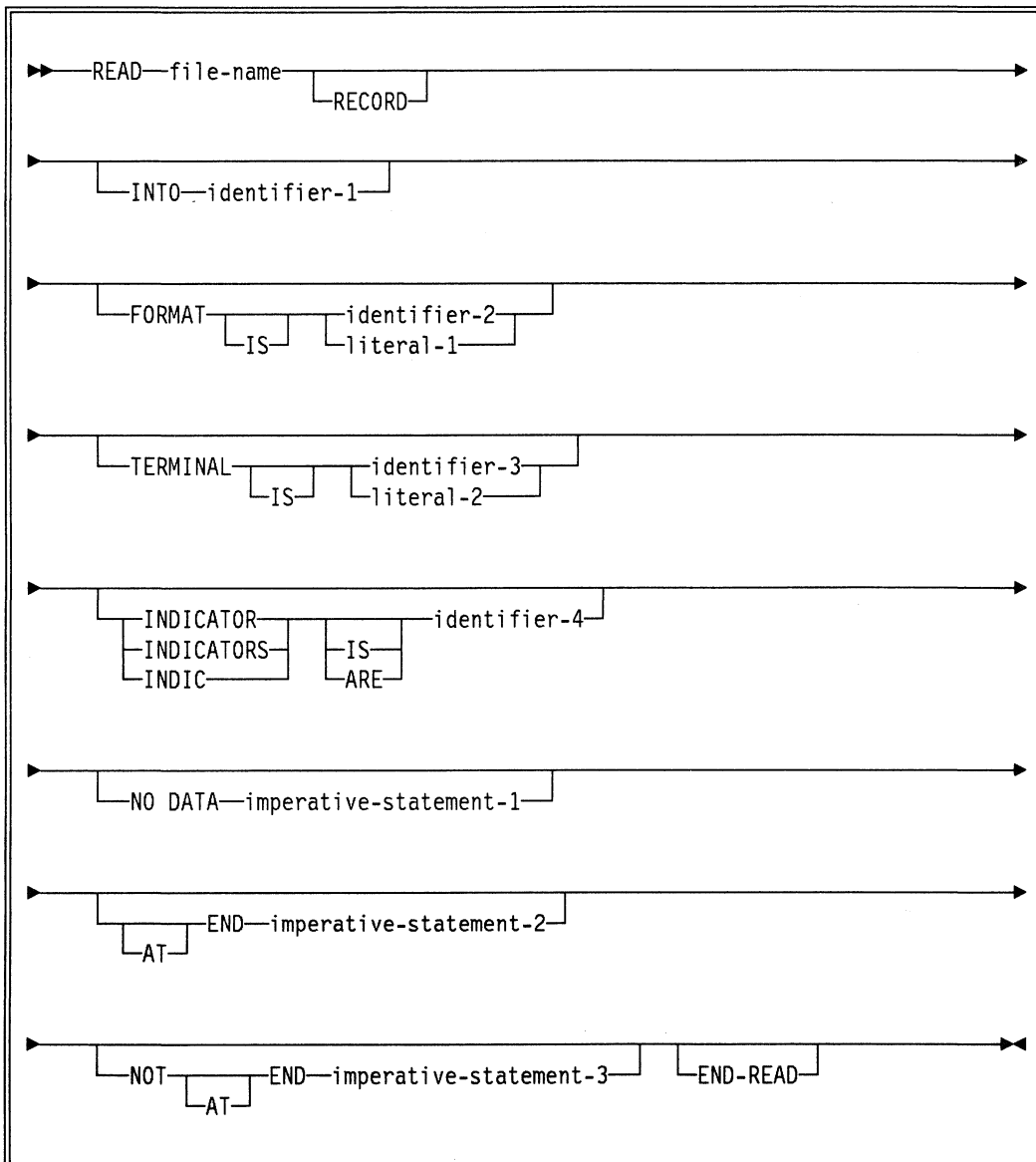
not immediately available from the program device specified by the TERMINAL phrase.

If the TERMINAL phrase is specified and the data-item or literal has a value of blanks, the phrase is treated at run time as if it were not specified.

READ Statement

The READ statement makes available a record from a device, using a named format. If the format is a subfile, the READ statement makes available a specified record from that subfile.

READ Statement – Format 4 – TRANSACTION File (Nonsubfile)



Format 4 is used only to read a format that is not a subfile. The RELATIVE KEY data item, if specified in the FILE-CONTROL entry, is not used. The Format 4 READ statement is not valid for a subfile record. However, a Format 4 READ statement for the subfile control record format must be used to place those subfile records that were updated on a display into the subfile.

If the requested data is available, it is returned in the record area. The names of the record format and the program device are returned in the I-O-FEEDBACK area in the CONTROL-AREA.

The READ statement is valid only when there are acquired devices for the file. If a READ is processed and there are no acquired devices, the file status is set to 92 (logic error).

The manner in which the Format 4 READ statement functions depends on:

- If the READ is for a single device file or a multiple device file
- If a specific program device has been requested through the TERMINAL phrase
- If a specific record format has been requested through the FORMAT phrase
- If the NO DATA phrase has been specified.

In the following sections, references to *data available* or *returned* include the situation where only the response indicators are set. This also applies even when a separate indicator area is used and the indicators are not returned in the record area for the file.

The following chart shows the possible combinations of phrases and the function performed for a single device file or a multiple device file. For example, if TERMINAL is N, FORMAT is N, and NO DATA is N, the single device is D and multiple device is A.

Function	Phrase	Y=Yes N=No
Checked at Compilation	TERMINAL ²	N N N Y Y Y Y
	FORMAT ²	N N Y Y N N Y Y
	NO DATA	N Y N Y N Y N Y
Determined at Run Time	Single Device	D C D B D C D B
	Multiple Device	A A D B D C D B

Codes A through D are explained below:

Code A – Read From Invited Program Device (Multiple Device Files only)

This type of READ receives data from the first invited program device that has data available. Invited program devices are work stations or other communication devices that are invited to send input. The inviting is done by writing to the program device with a format specifying the DDS keyword INVITE. Once an invited program device is actually read from, it is no longer invited. That program device will not be used for input by another READ statement unless reinvited, or unless a READ is directed to it specifying the TERMINAL phrase or FORMAT phrase.

² If the phrase is specified and the data item or literal is blank, the phrase is treated at run time as if it were not specified.

The record format returned from the program device is determined by the system. See the chapter on display device support in the *Data Management Guide* for information on how record format is determined for work stations. See the *ICF Programmer's Guide* for information on the FMTSLT parameter on the ADDICFDEVE and OVRICFDEVE commands.

This READ can be completed without returning any data in the following cases:

- If there are no invited devices.
- If a controlled cancelation of the job occurs. This results in a file status value of 9A and a major/minor return code value of 0309.
- If the NO DATA phrase is omitted, and the specified wait time expires. This results in a file status value of 00 and a major/minor return code value of 0310.
- If the specified wait time is the value entered on the WAITRCD parameter for the file.
- If the NO DATA phrase is specified, and no data is immediately available when the READ is processed.

If data is available, it is returned in the record area. The record format is returned in the I-O-FEEDBACK area and in the CONTROL-AREA. For more information about "Reading from Invited Program Devices," see the *ICF Programmer's Guide*.

Code B – Read From One Program Device (Combination not Allowed)

A compilation-time message is issued, and the NO DATA phrase is ignored. See the table entry for the same combination of phrases with the NO DATA phrase omitted.

Code C – Read From One Program Device (with NO DATA phrase)

This function of the READ statement never causes program processing to stop and wait until data is available. Either the data is immediately available or the NO DATA imperative-statement is processed.

This READ function can be used to periodically check if data is available from a particular program device (either the default program device or one specified by the TERMINAL phrase). This checking for data is done in the following manner:

1. The program device is determined as follows:
 - a. If the TERMINAL phrase was omitted or contains blanks, the default program device is used. The default program device is the one used by the last attempted READ, WRITE, REWRITE, ACQUIRE, or DROP statement. If none of the above I/O operations were previously issued, the default program device is the first program device acquired.
 - b. If the TERMINAL phrase was specified, the indicated program device is used.
2. A check is done to determine if data is available and if the program device is invited.
3. If data is available, that data is returned in the record area and the program device is no longer invited. If no data is immediately available, the NO DATA imperative-statement is run and the program device remains invited.

4. If the program device is not invited, the AT END condition exists and the file status is set to 10.

Code D—Read From One Program Device (without NO DATA Phrase)

This READ always waits for data to be made available. Even if the job receives a controlled cancellation, or a WAITRCD time is specified for the file, the program will never regain control from the READ statement. This READ operation is performed in the following manner:

1. The program device is determined as follows:
 - a. If the TERMINAL phrase is omitted or contains a blank value, the default program device is used. The default program device is the program device used by the last attempted READ, WRITE, REWRITE, ACQUIRE, DROP or ACCEPT (Attribute Data) statement. If none of these operations has been done, the program device implicitly acquired when the file was opened is used. If there are no acquired devices, the AT END condition exists.
 - b. If the TERMINAL phrase is specified, the indicated program device is used.
2. The record format is determined as follows:
 - a. If the FORMAT phrase is omitted or contains blanks, the record format returned is determined by the system. For information on how the record format is calculated for work station devices, refer to the *Data Management Guide*. For information about how the record format is determined for communications, see the section on the FMTSLT parameter on the ADDICFDEVE and OVRICFDEVE commands in the *ICF Programmer's Guide*.
 - b. If the FORMAT phrase is specified, the indicated record format is returned. If the data available does not match the requested record format, a file status of 9G is set.
3. Program processing stops until data becomes available. The data is returned in the record area after the READ statement is run. If the program device was previously invited, it will no longer be invited after this READ statement.

INTO Phrase

The INTO phrase can be specified if:

- Only one record description is subordinate to the file description entry,

OR

- All record names associated with file-name and the data item referenced by identifier-1 describe a group item or an elementary alphanumeric item.

FORMAT Phrase

Literal-1 or identifier-2 specifies the name of the record format to be read. Literal-1, if specified, must be nonnumeric and 10 characters or fewer in length. Identifier-2, if specified, must refer to an alphanumeric data item, 10 characters or fewer in length. If identifier-2 contains blanks, the READ statement is run as if the FORMAT phrase were omitted.

NO DATA Phrase

When the NO DATA phrase is specified, the READ statement determines if data is immediately available. If data is available, the data is returned in the record area. If no data is immediately available, imperative-statement-1 is processed. The NO DATA phrase prevents the READ statement from waiting for data to become available.

TERMINAL Phrase

Literal-2 or identifier-3 specifies the program device name. Literal-2, if specified, must be nonnumeric and 10 characters or fewer in length. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or fewer in length. The program device must have been acquired before the READ statement is processed. If identifier-3 contains blanks, the READ statement is processed as if the TERMINAL phrase were omitted. For a single device file, the TERMINAL phrase can be omitted. The program device is assumed to be that single device.

If the TERMINAL phrase is omitted for a READ of a TRANSACTION file that has acquired multiple program devices, the default program device is used. See the discussion of the TERMINAL phrase on page 178, to see how the default program device is determined.

AT END Phrase

Imperative-statement-2 is performed when the AT END condition is detected.

Note: An AT END condition occurs at the following times:

- During a READ statement for a sequentially accessed file when no next logical record exists in the file, or when the number of significant digits in the relative record number is larger than the size of the relative key data item, or when an optional input file is not present.
- During a RETURN statement when no logical record exists for the associated sort or merge file.
- During a SEARCH statement when the search operation ends without satisfying the condition specified in any of the associated WHEN phrases.

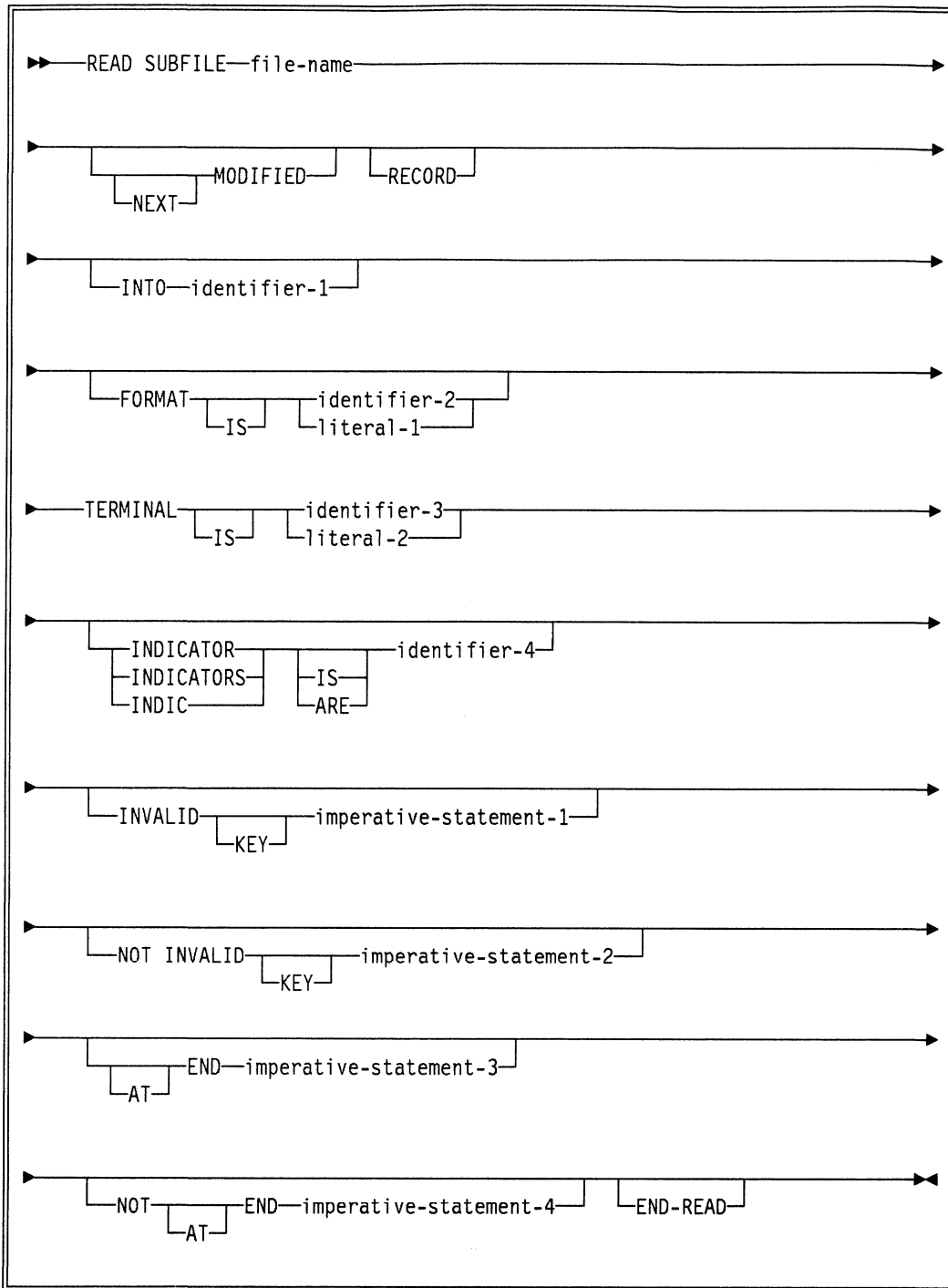
NOT AT END Phrase

This phrase allows you to specify procedures to be performed when the READ operation is successful.

END-READ Phrase

The END-READ phrase serves to explicitly delimit the scope of the statement.

READ Statement – Format 5 – TRANSACTION File (Subfile)



Format 5 is used only to read a format that is a subfile record. The AT END phrase can only be used when the NEXT MODIFIED phrase is specified. The INVALID KEY phrase must not be used when the NEXT MODIFIED phrase is specified.

Format 5 cannot be used for communications devices. If the subfile format of the READ statement is used for a communications device, the READ fails and a file status of 90 is set.

Random Access of Subfile Records: The NEXT MODIFIED phrase must not be used to randomly access records in a subfile. The INVALID KEY phrase can only be used for random access of subfile records.

Sequential Access of Subfile Records: The NEXT MODIFIED phrase must be specified to access subfile records sequentially. The AT END phrase can only be specified with the NEXT MODIFIED phrase.

NEXT MODIFIED Phrase

When NEXT MODIFIED is not specified, the data record made available is the record in the subfile with a relative record number that corresponds to the value of the RELATIVE KEY data item.

When the NEXT MODIFIED phrase is not specified, and if the RELATIVE KEY data item contains a value other than the relative record number of a record in the subfile, the INVALID KEY condition exists and the running of the READ statement is unsuccessful.

When the NEXT MODIFIED phrase is specified, the record made available is the next modified record following the current pointer position in the file. For information about turning on the Modified Data Tag, see the *Data Management Guide*.

The search for the next modified record begins:

- At the beginning of the subfile if:
 - An I/O operation has been performed for the subfile control record.
 - The I/O operation cleared, initialized, or displayed the subfile.
- For all other cases, with the record following the record that was read by a previous read operation.

The value of the RELATIVE KEY data item is updated to reflect the relative record number of the record made available to the program.

If NEXT MODIFIED is specified and there are no further user-modified records in the subfile, the AT END condition exists. Imperative-statement-2, or an applicable USE AFTER ERROR/EXCEPTION procedure, if any, is then run.

FORMAT Phrase

When a format-name is not specified, the format used is the last record format written to the display device that contains input fields, input/output fields, or hidden fields. If no such format exists for the display file, the format used is the record format of the last WRITE operation to the display device.

Note: An **input field** is a field specified in a display file or database file that is reserved for information supplied by a user.

If the FORMAT phrase is specified, literal-1 or the contents of identifier-2 must specify a format, which is active for the appropriate program device. The READ statement reads a data record of the specified format.

To ensure correct results, always specify the **FORMAT** phrase for multiple format files. For more information on the **FORMAT** phrase, see the Procedure Division, “Common Processing Facilities” on page 177.

TERMINAL Phrase

See Format 4 of the **READ** Statement for general considerations concerning the **TERMINAL** phrase.

For a Format 5 **READ**, if the **TERMINAL** phrase is omitted for a file that has multiple devices acquired for it, a record is read from the subfile associated with the default program device. See the discussion of the **TERMINAL** phrase on page 178, to see how the default program device is determined.

INVALID KEY Phrase

If the **RELATIVE KEY** data item at the time of running the statement contains a value that does not correspond to a relative record number for the subfile, the **INVALID KEY** condition exists and the running of the statement is unsuccessful. To see what happens next, refer to the diagrams on pages 74 through 76.

For a Format 5 **READ**, do not specify the **INVALID KEY** phrase if the **NEXT MODIFIED** phrase is not specified and there is no applicable **USE** procedure specified for the file name.

NOT INVALID KEY Phrase

This phrase allows you to specify procedures to be performed when the **READ** operation is successful.

AT END Phrase

If the **NEXT MODIFIED** phrase is specified and there is no user-modified record in the subfile, the **AT END** condition exists, and the **READ** operation is unsuccessful.

Specify the **AT END** phrase when the **NEXT MODIFIED** phrase is used, and no applicable **USE** procedure is specified for the file name. If the **AT END** phrase and a **USE** procedure are both specified for a file, and the **AT END** condition arises, control transfers to the **AT END** imperative statement and the **USE** procedure is not run.

NOT AT END Phrase

This phrase allows you to specify procedures to be performed when the **READ** operation is successful.

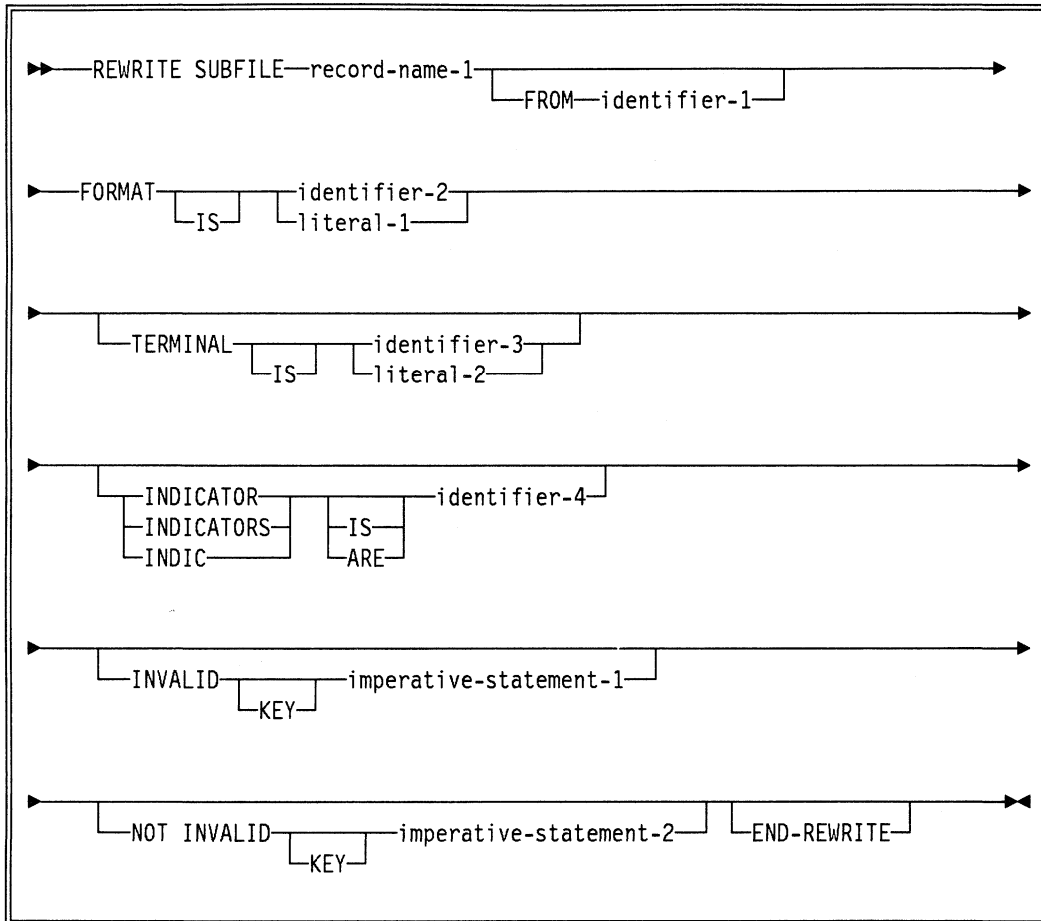
END-READ Phrase

The **END-READ** phrase serves to explicitly delimit the scope of the statement.

REWRITE Statement

The REWRITE statement is used to replace a subfile record that already exists in the subfile.

REWRITE Statement – Format 2 – TRANSACTION File (Subfile)



The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced. A successful READ operation on the record must be done prior to the REWRITE operation. The record replaced in the subfile is the record in the subfile accessed by the previous READ operation.

FORMAT Phrase

The record format specified in the FORMAT phrase must be the record format accessed on the previous READ operation. Literal-1 or the contents of identifier-2 must be the name of the subfile format accessed on the previous READ. For more information on the FORMAT phrase, see "Common Processing Facilities" on page 177.

TERMINAL Phrase

The TERMINAL phrase indicates which program device's subfile is to have a record rewritten. If the TERMINAL phrase is specified, literal-2 or identifier-3 must refer to a work station that has been acquired by the TRANSACTION file. If literal-2 or identifier-3 contains blanks, the TERMINAL phrase has no effect. The program device specified by the TERMINAL phrase must have been acquired, either explicitly or implicitly, and must have a subfile associated with the device.

Literal-2 or identifier-3 must be a valid program device name. Literal-2, if specified, must be nonnumeric and 10 characters or fewer. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or fewer.

If the TERMINAL phrase is omitted from a TRANSACTION file that has acquired multiple program devices, the subfile used is the subfile associated with the last program device from which a READ of the TRANSACTION file was attempted.

The REWRITE statement cannot be used for communications devices. If the REWRITE statement is used for a communications device, the operation fails and a file status of 90 is set.

INVALID KEY Phrase

If the RELATIVE KEY data item at the time of running the statement contains a value that does not correspond to a relative record number for the subfile, the INVALID KEY condition exists and the running of the statement is unsuccessful. To see what happens next, refer to the diagrams on pages 74 through 76.

NOT INVALID KEY Phrase

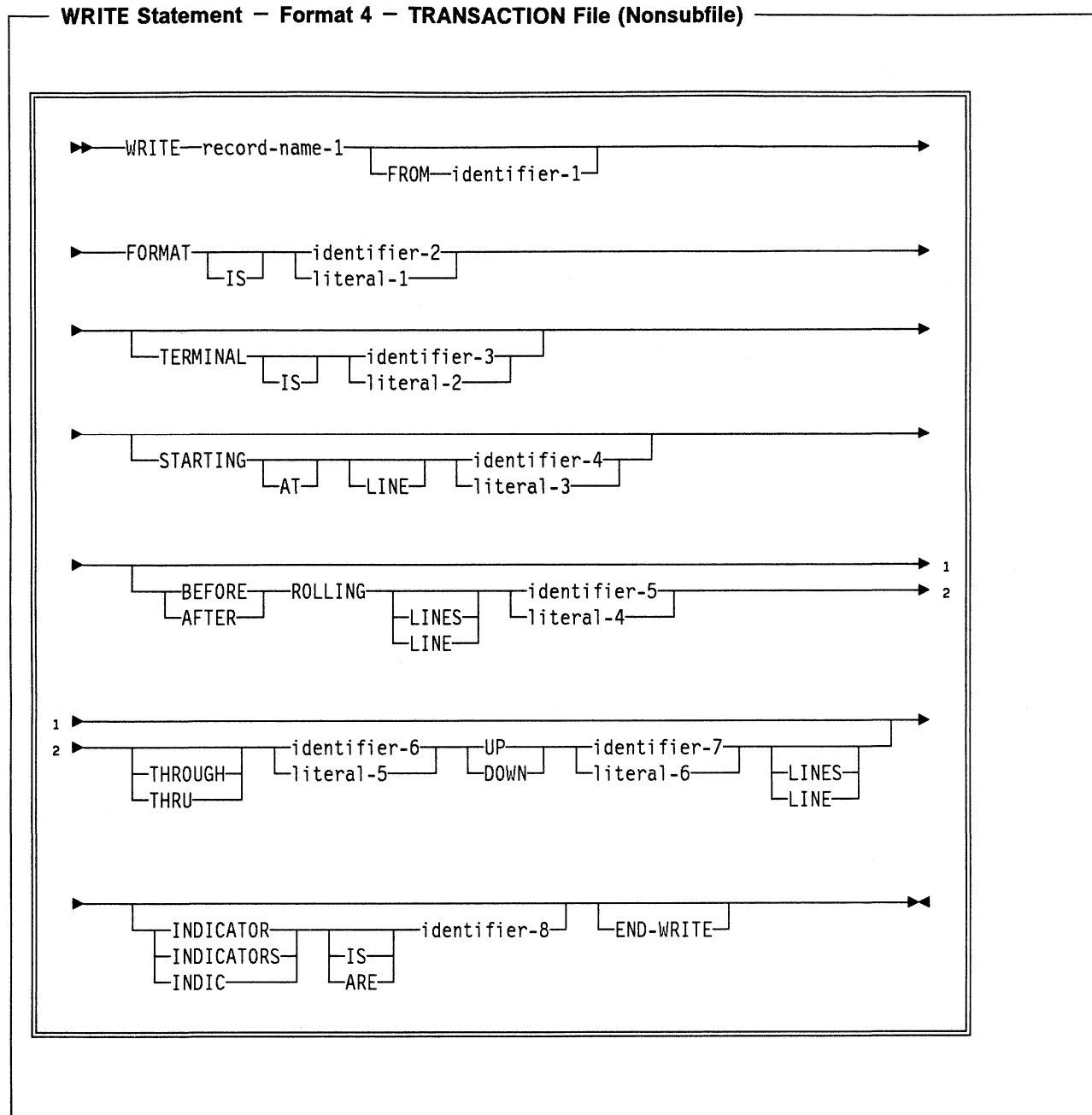
This phrase allows you to specify procedures to be performed when the REWRITE operation is successful.

END-REWRITE Phrase

The END-REWRITE phrase serves to explicitly delimit the scope of the statement.

WRITE Statement

The WRITE statement releases a logical record to the file.



TERMINAL Phrase

The TERMINAL phrase specifies the program devices to which the output record is to be sent.

The contents of literal-2 or identifier-3 must be the name of a program device previously acquired, either implicitly or explicitly, by the file. Literal-2, if specified, must be nonnumeric and 10 characters or fewer in length. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or fewer in length. A value of blanks is treated as if the TERMINAL phrase were omitted.

If only a single program device was acquired by the TRANSACTION file, the TERMINAL phrase can be omitted. That program device is always used for the WRITE.

If the TERMINAL phrase is omitted for a WRITE operation to a TRANSACTION file that has acquired multiple program devices, the default program device is used. See the discussion of the TERMINAL phrase on page 178 to see how the default program device is determined.

STARTING Phrase

The STARTING phrase specifies the starting line number for the record formats that use the variable start line keyword. This phrase is only valid for display devices.

The actual line number on which a field begins can be determined from the following equation:

$$\text{Actual-line} = \text{Start-line} + \text{DDS Start-line} - 1$$

Figure 66. Line Number Equation for the STARTING Phrase

Where:

Actual-line is the actual line number

Start-line is the starting line number specified in the program

DDS Start-line is the line number specified in positions 39 through 41 of the Data Description Specifications form.

The WRITE operation is successful if:

- The result of the above equation is positive and less than or equal to the number of lines on the display.
- The value specified for the STARTING phrase is 0. In this case, a value of 1 is assumed.

The WRITE operation is unsuccessful, and the program ends, if:

- The result of the above equation is greater than the number of lines on the display.
- The value specified for the STARTING phrase is negative.

If the value specified for the STARTING phrase is within the screen area, any fields outside of the screen area are ignored.

Literal-3 of the STARTING phrase must be a numeric literal. Identifier-4 must be an elementary numeric item.

To use the STARTING phrase, the DDS record level keyword SLNO(*VAR) must be specified for the format being written. If the record format does not specify this keyword, the STARTING phrase is ignored at run time.

The DDS keyword CLRL also affects the STARTING phrase. CLRL controls how much of the display is cleared when the WRITE statement is processed.

See the *DDS Reference* for further information on SLNO(*VAR) and CLRL keywords.

ROLLING Phrase

The ROLLING phrase allows you to move lines displayed on the work station screen. All or some of the lines on the screen can be rolled up or down. The lines vacated by the rolled lines are cleared, and can have another screen format written into them. This phrase is only valid for display devices.

ROLLING is specified in the WRITE statement that is writing a new format to the display. You must specify whether the write is before or after the roll, the range of lines you want to roll, how many lines you want to roll these lines, and whether the roll operation is up or down.

After lines are rolled, the fields on these lines retain their DDS display attributes, for example, underlining, but lose their DDS usage attributes, for example, input-capability. Fields on lines that are written and then rolled (BEFORE ROLLING phrase) also lose their usage attributes.

If any part of a format is rolled, the entire format loses its usage attributes. If more than one format exists, only the rolled formats lose their usage attributes.

When you specify the ROLLING phrase, the following general rules apply.

- The DDS record level keyword ALWROL must be specified for every record format written in a WRITE statement containing the ROLLING phrase.
- Other DDS keywords mutually exclusive with the ALWROL keyword must not be used.
- Either of the DDS keywords, CLRL or OVERLAY, must be specified for a record format that is to be written and rolled to prevent the display from being cleared when that record format is written. See the *DDS Reference* manual for more information on DDS keywords.
- All the identifiers and literals must represent positive integer values.
- The roll starting line number (identifier-5 or literal-4) must not exceed the ending line number (identifier-6 or literal-5).
- The contents of lines that are rolled outside of the window specified by the starting and ending line numbers disappear.

Figure 67 on page 193 shows an example of a rolling operation. An initial screen format, FMT1, is written on the display. The program processes this screen format and is now ready to write the next screen format, FMT2, to the work station screen. Part of FMT1 is rolled down two lines before FMT2 is written to the display.

Processing of the following WRITE statement causes part of FMT1 to be rolled down two lines, and FMT2 to be written to the display:

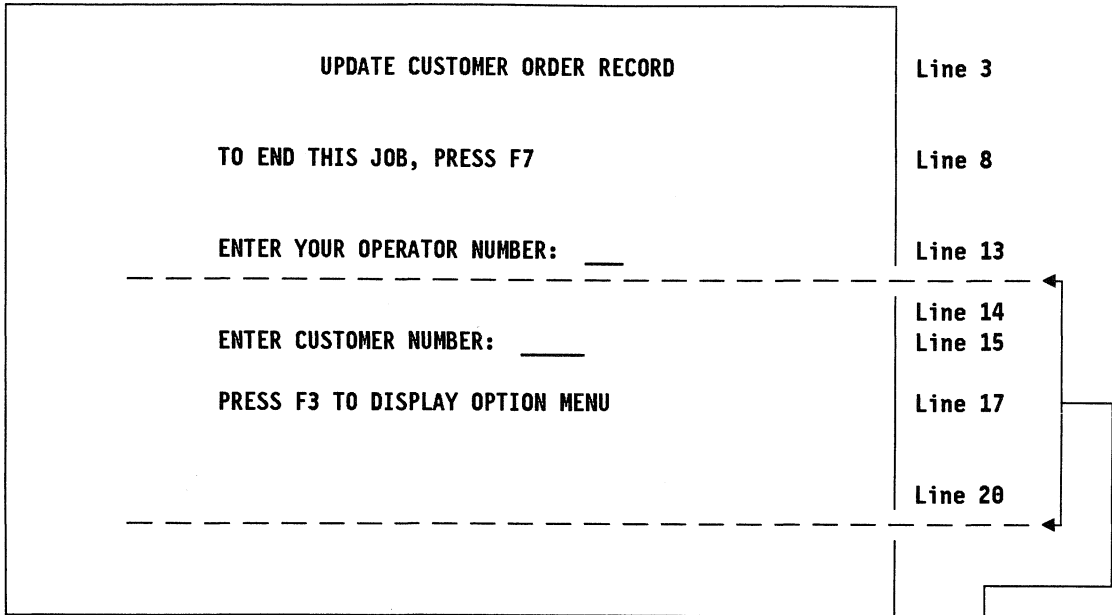
```
WRITE SCREENREC FORMAT "FMT2"  
  AFTER ROLLING LINES 14 THROUGH 20  
  DOWN 2 LINES
```

When this WRITE statement is run, the following steps occur:

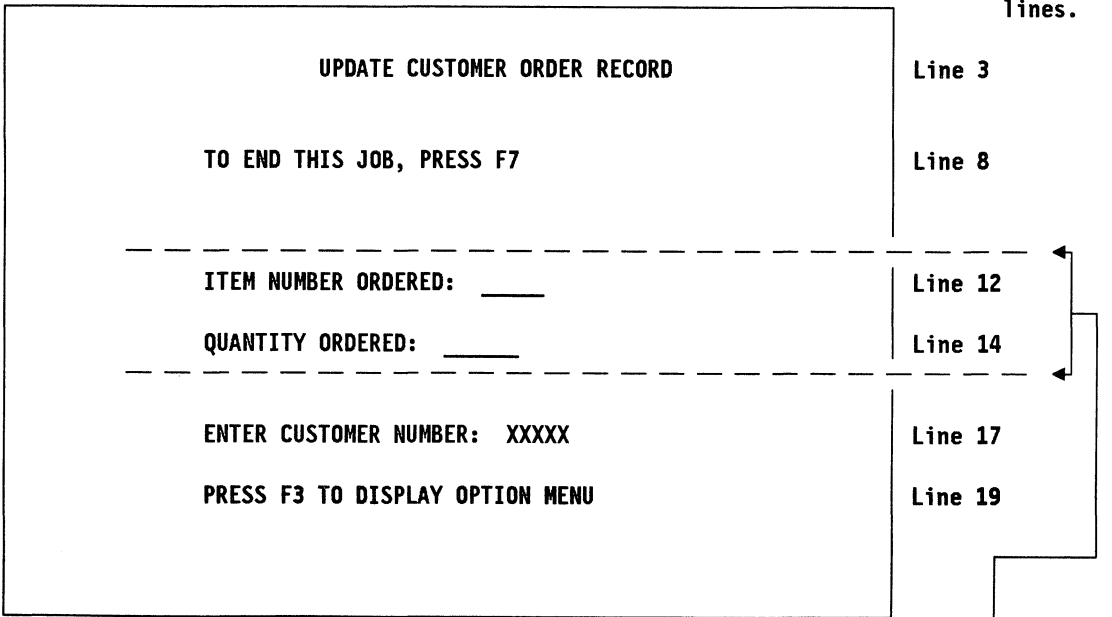
1. The contents of lines 14 through 20 are rolled down two lines.
 - a. The contents of lines 14 through 18 now appear on lines 16 through 20.

- b. The contents of lines 14 and 15 are vacated and cleared.
 - c. The contents of lines 19 and 20 are rolled outside the window and disappear.
2. After the rolling operation takes place, FMT2 is written to the display.
 - a. Part of FMT2 is written to the area vacated by the roll operation.
 - b. Part of FMT2 is written over the data left from FMT1.
3. When the contents of the display are returned to the program by a READ statement, only the input capable fields of FMT2 are returned.

DISPLAY BEFORE PROCESSING THE WRITE STATEMENT



DISPLAY AFTER PROCESSING THE WRITE STATEMENT

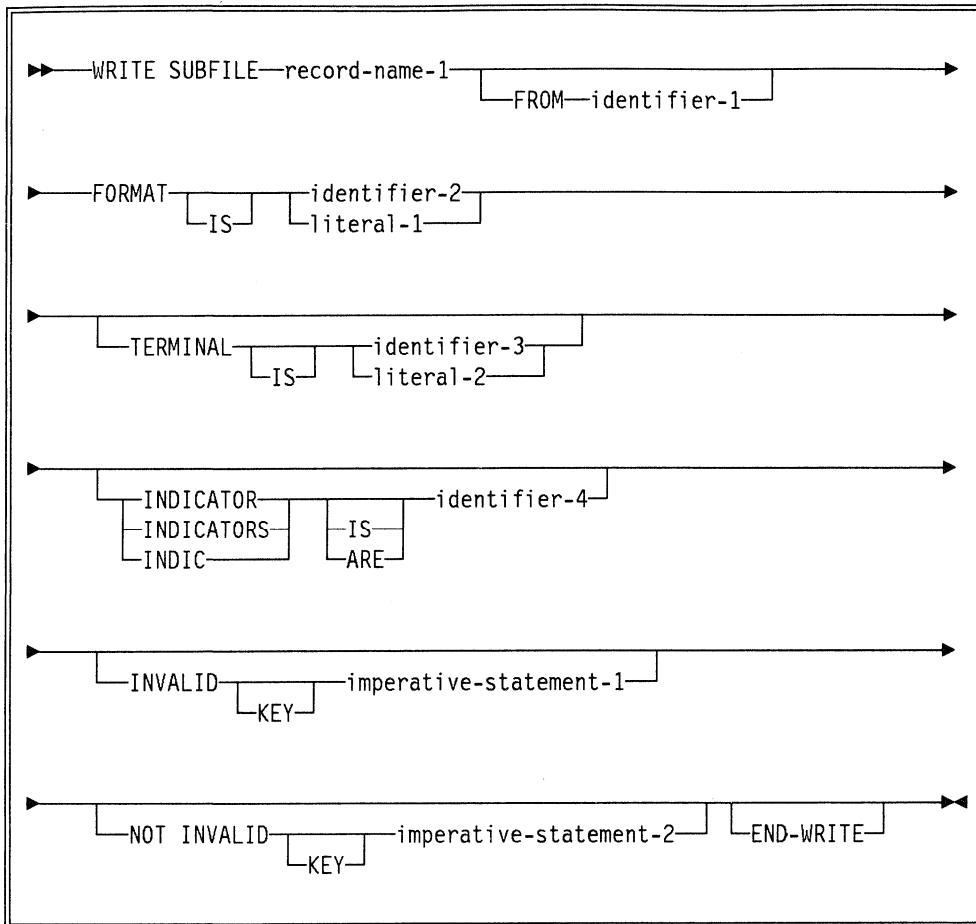


These seven lines of FMT1 will be rolled down 2 lines.

These three lines of FMT2 have been written over the previous lines.

Figure 67. Example of ROLLING Operation

WRITE Statement – Format 5 – TRANSACTION File (Subfile)



Format 5 can only be used for display devices. If the subfile form of the WRITE statement is used for any other type of device, the WRITE operation fails and a file status of 90 is set.

If the format is a subfile record, and SUBFILE is specified, the RELATIVE KEY clause must have been specified on the SELECT clause for the file being written. The record written to the subfile is the record in the subfile identified by the format name that has a relative record number equal to the value of the RELATIVE KEY data item. See the *Data Management Guide* for more information on subfiles.

TERMINAL Phrase

See the explanation following Format 4 for general considerations concerning the TERMINAL phrase.

The TERMINAL phrase specifies which program device's subfile is to have a record written to it. If the TERMINAL phrase is specified, literal-2 or identifier-3 must refer to a work station associated with the TRANSACTION file. If literal-2 or identifier-3 contains a value of blanks, the TERMINAL phrase is treated as if it

were not specified. The work station specified by the TERMINAL phrase must have been acquired, either explicitly or implicitly.

If the TERMINAL phrase is omitted, the subfile used is the subfile associated with the default program device. See the discussion of the TERMINAL phrase on page 178 to see how the default program device is determined.

INVALID KEY Phrase

The INVALID KEY condition exists if a record is already in the subfile with that record number, or if the relative record number specified is greater than the maximum allowable subfile record number. The INVALID KEY phrase should be specified in the WRITE SUBFILE statement for all files for which an appropriate USE procedure is not specified.

For information about what happens when the INVALID KEY condition arises, refer to the diagrams on pages 74 through 76.

NOT INVALID KEY Phrase

This phrase allows you to specify procedures to be performed when the WRITE operation is successful.

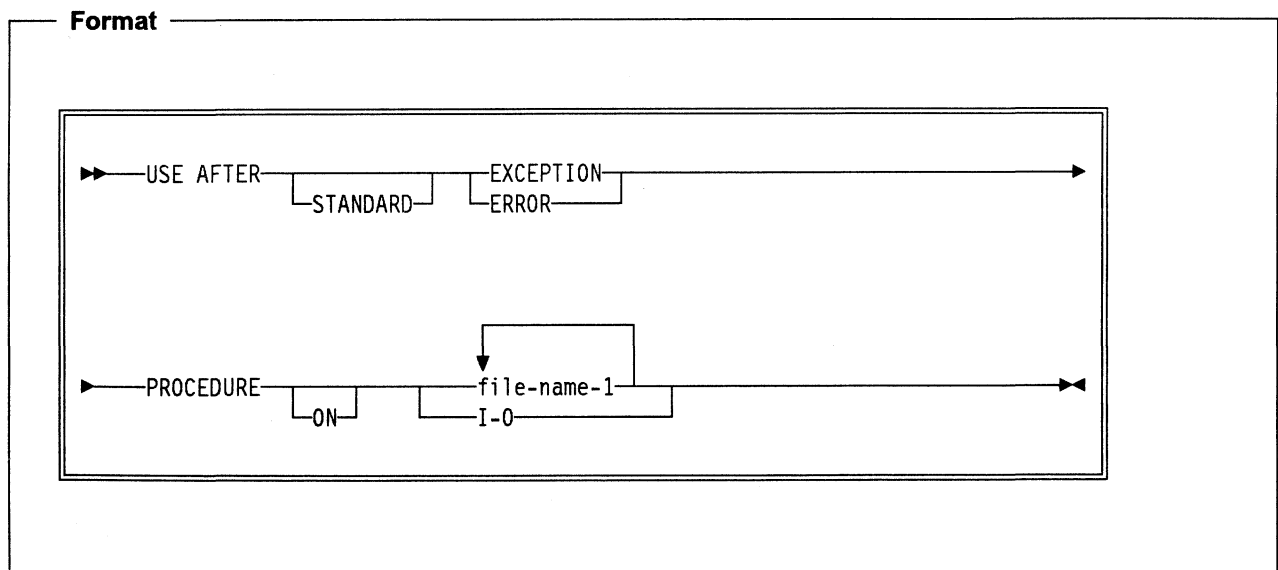
END-WRITE Phrase

The END-WRITE phrase serves to explicitly delimit the scope of the statement.

For a further discussion of the WRITE statement, the FROM phrase, and the INVALID KEY phrase, see the *COBOL/400* Reference*. For information on the FORMAT phrase, see the Procedure Division, "Common Processing Facilities" on page 177.

USE Statement

The USE statement specifies procedures for input/output error handling that are in addition to the standard procedures provided by the input/output control system.



See the "USE Statement" section of the *COBOL/400* Reference* for a further discussion of the USE statement.

Examples of Work Station Programs

This section contains examples of COBOL programs that illustrate work station applications on the AS/400 system.

Basic Inquiry Program

Figure 68 shows the associated DDS for a basic inquiry program that uses the COBOL TRANSACTION file.

File		Keying Instruction	Graphic Key	Description	Page of
Programmer	Date				

Sequence Number	Form Type	Indicator	Condition Name	Name	Length	References (R)	Data Type/Keyboard Shift	Function Position	Usage (B/O/D/E/H/M/N/P)	Location		Functions
										Line	Pos	
1	A		CUSTOMER MASTER INQUIRY FILE									
2	A											
3	A											
4	A											
5	A											
6	A											
7	A											
8	A											
9	A											
10	A											
11	A											
12	A											
13	A											
14	A											
15	A											
16	A											
17	A											
18	A											
19	A											
20	A											
21	A											
22	A											
23	A											
24	A											
25	A											
26	A											
27	A											
28	A											
29	A											
30	A											
31	A											
32	A											
33	A											
34	A											
35	A											
36	A											
37	A											
38	A											
39	A											
40	A											
41	A											
42	A											
43	A											
44	A											
45	A											
46	A											
47	A											
48	A											
49	A											
50	A											
51	A											
52	A											
53	A											
54	A											
55	A											
56	A											
57	A											
58	A											
59	A											
60	A											
61	A											
62	A											
63	A											
64	A											
65	A											
66	A											
67	A											
68	A											
69	A											
70	A											
71	A											
72	A											
73	A											
74	A											
75	A											
76	A											
77	A											
78	A											
79	A											
80	A											
81	A											
82	A											
83	A											
84	A											
85	A											
86	A											
87	A											
88	A											
89	A											
90	A											
91	A											
92	A											
93	A											
94	A											
95	A											
96	A											
97	A											
98	A											
99	A											
100	A											

Figure 68. Example of a TRANSACTION Inquiry Program Using a Single Display Device

The data description specifications (DDS) for the display device file (CUSMINQ) to be used by this program describe two record formats: CUSPMT and CUSFLDS.

The CUSPMT record format contains the constant 'Customer Master Inquiry', which identifies the display. It also contains the prompt 'Customer Number' and the input field (CUST) where you enter the customer number. Five underscores

appear under the input field CUST on the display where you are to enter the customer number. The error message:

Customer number not found

is also included in this record format. This message is displayed if indicator 99 is set to **ON** by the program. In addition, this record format defines a function key that you can press to end the program. When you press function key F3, indicator 15 is set to **ON** in the COBOL program. This indicator is then used to end the program.

The CUSFLDS record format contains the following constants:

- Name
- Address
- City
- State
- Zip Code
- A/R Balance.

These constants identify the fields to be written out from the program. This record format also describes the fields that correspond to these constants. All of these fields are described as output fields (blank in position 38) because they are filled in by the program; you do not enter any data into these fields. To enter another customer number, press Enter in response to this record. Notice that the CUSFLDS record overlays the CUSPMT record. Therefore, when the CUSFLDS record is written to the display, the CUSPMT record remains on the display.

In addition to describing the constants, fields, and attributes for the display, the record formats also define the line numbers and horizontal positions where the constants and fields are to be displayed.

Note: The field attributes are defined in a physical file (CUSMSTP) used for field reference purposes, instead of in the DDS for the display file. For example, EDTCDE(J) is defined in CUSMSTP for the field ARBAL.


```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                01/22/89
 2 000200 PROGRAM-ID. XMPLE766.                                  03/22/89
000300* SAMPLE TRANSACTION INQUIRY PROGRAM USING 1 DISPLAY DEVICE 01/22/89
 3 000400 AUTHOR. PROGRAMMER NAME.                               01/22/89
 4 000500 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.       01/22/89
 5 000600 DATE-WRITTEN. 12/21/88.                                01/22/89
 6 000670 DATE-COMPILED. 05/24/91 13:42:50 .
 7 000800 ENVIRONMENT DIVISION.                                  01/22/89
 8 000900 CONFIGURATION SECTION.                                 01/22/89
 9 001000 SOURCE-COMPUTER. IBM-AS400.                             01/22/89
10 001100 OBJECT-COMPUTER. IBM-AS400.                             01/22/89
11 001200 INPUT-OUTPUT SECTION.                                  01/22/89
12 001300 FILE-CONTROL.                                          01/22/89
13 001400 SELECT CUST-DISPLAY                                     01/22/89
14 001500 ASSIGN TO WORKSTATION-CUSMINQ                          01/22/89
15 001600 ORGANIZATION IS TRANSACTION                            01/22/89
16 001700 CONTROL-AREA IS WS-CONTROL.                            01/22/89
17 001800 SELECT CUST-MASTER                                     01/22/89
18 001900 ASSIGN TO DATABASE-CUSMSTP                             01/22/89
19 002000 ORGANIZATION IS INDEXED                                01/22/89
20 002100 ACCESS IS RANDOM                                       01/22/89
21 002200 RECORD KEY IS CUST OF CUSMST                           01/22/89
22 002300 FILE STATUS IS CM-STATUS.                               01/22/89
23 002400 DATA DIVISION.                                        01/22/89
24 002500 FILE SECTION.                                          01/22/89
25 002600 FD CUST-DISPLAY                                        01/22/89
26 002700 LABEL RECORDS ARE OMITTED.                            01/22/89
27 002800 01 DISP-REC.                                          01/22/89
28 002900 COPY DDS-ALL-FORMATS OF CUSMINQ.                       01/22/89
29 +000001 05 CUSMINQ-RECORD PIC X(80).                           <-ALL-FMTS
+000002* INPUT FORMAT:CUSPMT FROM FILE CUSMINQ OF LIBRARY XMPLIB <-ALL-FMTS
+000003* CUSTOMER PROMPT                                         <-ALL-FMTS
30 +000004 05 CUSPMT-I REDEFINES CUSMINQ-RECORD.                 <-ALL-FMTS
31 +000005 06 CUSPMT-I-INDIC.                                     <-ALL-FMTS
32 +000006 07 IN15 PIC 1 INDIC 15.                                <-ALL-FMTS
+000007* END OF PROGRAM                                          <-ALL-FMTS
33 +000008 07 IN99 PIC 1 INDIC 99.                                <-ALL-FMTS
+000009* CUSTOMER NUMBER NOT FOUND PRESS RESET, THE             <-ALL-FMTS
34 +000010 06 CUST PIC X(5).                                       <-ALL-FMTS
+000011* CUSTOMER NUMBER                                         <-ALL-FMTS
+000012* OUTPUT FORMAT:CUSPMT FROM FILE CUSMINQ OF LIBRARY XMPLIB <-ALL-FMTS
+000013* CUSTOMER PROMPT                                         <-ALL-FMTS
35 +000014 05 CUSPMT-0 REDEFINES CUSMINQ-RECORD.                 <-ALL-FMTS
36 +000015 06 CUSPMT-0-INDIC.                                     <-ALL-FMTS
37 +000016 07 IN99 PIC 1 INDIC 99.                                <-ALL-FMTS
+000017* CUSTOMER NUMBER NOT FOUND PRESS RESET, THE             <-ALL-FMTS
+000018* INPUT FORMAT:CUSFLDS FROM FILE CUSMINQ OF LIBRARY XMPLIB <-ALL-FMTS
+000019* CUSTOMER DISPLAY                                         <-ALL-FMTS
38 +000020 05 CUSFLDS-I REDEFINES CUSMINQ-RECORD.                 <-ALL-FMTS
39 +000021 06 CUSFLDS-I-INDIC.                                     <-ALL-FMTS
40 +000022 07 IN15 PIC 1 INDIC 15.                                <-ALL-FMTS
+000023* END OF PROGRAM                                          <-ALL-FMTS
+000024* OUTPUT FORMAT:CUSFLDS FROM FILE CUSMINQ OF LIBRARY XMPLIB <-ALL-FMTS
+000025* CUSTOMER DISPLAY                                         <-ALL-FMTS
41 +000026 05 CUSFLDS-0 REDEFINES CUSMINQ-RECORD.                 <-ALL-FMTS
42 +000027 06 NAME PIC X(25).                                       <-ALL-FMTS
+000028* CUSTOMER NAME                                           <-ALL-FMTS
43 +000029 06 ADDR PIC X(20).                                       <-ALL-FMTS
+000030* CUSTOMER ADDRESS                                         <-ALL-FMTS
44 +000031 06 CITY PIC X(20).                                       <-ALL-FMTS
+000032* CUSTOMER CITY                                           <-ALL-FMTS
45 +000033 06 STATE PIC X(2).                                       <-ALL-FMTS
+000034* STATE                                                   <-ALL-FMTS
46 +000035 06 ZIP PIC S9(5).                                       <-ALL-FMTS
+000036* ZIP CODE                                               <-ALL-FMTS
47 +000037 06 ARBAL PIC S9(6)V9(2).                                   <-ALL-FMTS
+000038* ACCOUNTS REC. BALANCE                                   <-ALL-FMTS
003000
48 003100 FD CUST-MASTER
49 003200 LABEL RECORDS ARE STANDARD.
50 003300 01 CUST-REC.
51 003400 COPY DDS-CUSMST OF CUSMSTP.
+000001* I-O FORMAT:CUSMST FROM FILE CUSMSTP OF LIBRARY XMPLIB CUSMST

```

Figure 70 (Part 1 of 2). Source Listing of a TRANSACTION Inquiry Program Using a Single Display Device.

```

5738CB1 V2R2M0 AS/400 COBOL Source
STMT SEQNBR -A 1 B.+...2.+...3.+...4.+...5.+...6.+...7..IDENTFCN S COPYNAME CHG DATE
+000002* CUSTOMER MASTER RECORD CUSMST
+000003*THE KEY DEFINITIONS FOR RECORD FORMAT CUSMST CUSMST
+000004* NUMBER NAME RETRIEVAL TYPE ALTSEQ CUSMST
+000005* 0001 CUST ASCENDING AN NO CUSMST
52 +000006 05 CUSMST. CUSMST
53 +000007 06 CUST PIC X(5). CUSMST
+000008* CUSTOMER NUMBER CUSMST
54 +000009 06 NAME PIC X(25). CUSMST
+000010* CUSTOMER NAME CUSMST
55 +000011 06 ADDR PIC X(20). CUSMST
+000012* CUSTOMER ADDRESS CUSMST
56 +000013 06 CITY PIC X(20). CUSMST
+000014* CUSTOMER CITY CUSMST
57 +000015 06 STATE PIC X(2). CUSMST
+000016* STATE CUSMST
58 +000017 06 ZIP PIC S9(5) COMP-3. CUSMST
+000018* ZIP CODE CUSMST
59 +000019 06 SRHCOD PIC X(6). CUSMST
+000020* CUSTOMER NUMBER SEARCH CODE CUSMST
60 +000021 06 CUSTYP PIC S9(1) COMP-3. CUSMST
+000022* CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT CUSMST
61 +000023 06 ARBAL PIC S9(6)V9(2) COMP-3. CUSMST
+000024* ACCOUNTS REC. BALANCE CUSMST
62 +000025 06 ORDBAL PIC S9(6)V9(2) COMP-3. CUSMST
+000026* A/R AMT. IN ORDER FILE CUSMST
63 +000027 06 LSTAMT PIC S9(6)V9(2) COMP-3. CUSMST
+000028* LAST AMT. PAID IN A/R CUSMST
64 +000029 06 LSTDAT PIC S9(6) COMP-3. CUSMST
+000030* LAST DATE PAID IN A/R CUSMST
65 +000031 06 CRDLMT PIC S9(6)V9(2) COMP-3. CUSMST
+000032* CUSTOMER CREDIT LIMIT CUSMST
66 +000033 06 SLSYR PIC S9(8)V9(2) COMP-3. CUSMST
+000034* CUSTOMER SALES THIS YEAR CUSMST
67 +000035 06 SLSLYR PIC S9(8)V9(2) COMP-3. CUSMST
+000036* CUSTOMER SALES LAST YEAR CUSMST
003500
68 003600 WORKING-STORAGE SECTION.
69 003700 01 ONE PIC 1 VALUE B"1".
70 003800 01 CM-STATUS PIC X(2).
71 003900 01 WS-CONTROL.
72 004000 02 WS-IND PIC X(2).
73 004100 02 WS-FORMAT PIC X(10).
74 004200 PROCEDURE DIVISION.
004300 BEGIN.
75 004400 OPEN I-O CUST-DISPLAY, INPUT CUST-MASTER.
76 004500 MOVE ZERO TO IN99 OF CUSPMT-0.
004600 LOOP.
77 004700 WRITE DISP-REC FORMAT IS "CUSPMT".
78 004800 READ CUST-DISPLAY RECORD.
79 004900 IF IN15 OF CUSPMT-I
005000 IS EQUAL TO ONE
80 005100 THEN GO TO FINIS.
81 005200 MOVE CUST OF CUSPMT-I TO CUST OF CUSMST.
82 005300 READ CUST-MASTER RECORD.
83 005400 IF CM-STATUS IS NOT EQUAL "00" THEN
84 005500 MOVE ONE TO IN99 OF CUSPMT-0, GO TO LOOP.
86 005600 MOVE CORRESPONDING CUSMST TO CUSFLDS-0.
87 005700 WRITE DISP-REC FORMAT IS "CUSFLDS".
88 005800 READ CUST-DISPLAY RECORD.
89 005900 IF IN15 OF CUSFLDS-I
006000 IS EQUAL TO ONE
90 006100 THEN GO TO FINIS.
91 006200 MOVE ZERO TO IN99 OF CUSPMT-0.
92 006300 GO TO LOOP.
006400 FINIS.
93 006500 CLOSE CUST-DISPLAY, CUST-MASTER.
006600 RETURN-TO-CALLER.
94 006700 EXIT PROGRAM.
***** END OF SOURCE *****

```

Figure 70 (Part 2 of 2). Source Listing of a TRANSACTION Inquiry Program Using a Single Display Device.

The complete source listing for this program example is shown here. In particular, note the FILE-CONTROL and FD entries and the data structures generated by the Format 2 COPY statements.

The WRITE operation in statement 77 writes the CUSPMT format to the display. This record prompts you to enter a customer number. If you enter a customer number and press Enter, the next READ operation then reads the record back into the program.

The READ operation in statement 82 uses the customer number (CUST) field to retrieve the corresponding CUSMST record from the CUSMSTP file. If no record is found in the CUSMSTP file, indicator 99 is set on. The GO TO operation in statement 84, which is run when indicator 99 is set on, causes the program to branch back to the beginning. The message:

```
Customer number not found
```

is displayed when the format is written, because it is conditioned by indicator 99 in the DDS for the file. When you receive this message, the keyboard locks. You must press the Reset key in response to this message to unlock the keyboard. You can then enter another customer number.

If the READ operation retrieves a record from the CUSMSTP file, the WRITE operation writes the CUSFLDS record to the display work station. This record contains the customer's name, address, and accounts receivable balance.

You then press Enter, and the program branches back to the beginning. You can enter another customer number or end the program. To end the program, press F3, which sets on indicator 15 in the program.

When indicator 15 is on, the program closes all files and processes the EXIT PROGRAM statement. The program then returns control to the individual who called the COBOL program.

This is the initial display written by the WRITE operation in statement 77:

```
Customer Master Inquiry
Customer Number _____
Use F3 to end program, use enter key to return to prompt screen
```

This display appears if a record is found in the CUSMSTP file for the customer number entered in response to the first display:

```
Customer Master Inquiry

Customer Number  1000

Use F3 to end program, use enter key to return to prompt screen

Name  EXAMPLE WHOLESALERS LTD.
Address ANYWHERE STREET
City  ACITY
State  IL           Zipcode 12345
A/R balance      137.02
```

This display appears if the CUSMSTP file does not contain a record for the customer number entered in response to the first display:

```
Customer Master Inquiry

Customer Number

Use F3 to end program, use enter key to return to prompt screen

Customer number not found, press reset, then enter valid number
```

Order Inquiry Programs Using Subfiles

Figure 72 on page 206 shows an example of an order inquiry program, XMPLE773, that uses subfiles. The associated DDS is also shown, except for the DDS for the customer master file, CUSMSTP. Refer to Figure 69 on page 198 for the DDS for CUSMSTP.

XMPLE773 displays all the detail order records for the requested order number. The program prompts you to enter the order number that is to be reviewed. The order number is checked against the order header file, ORDHDRP. If the order number exists, the customer number accessed from the order header file is checked against the customer master file, CUSMSTP. All order detail records in ORDDTLP for the requested order are read and written to the subfile. A write for the subfile control record format is processed, and the detail order records in the subfile are displayed for you to review. You end the program by pressing F12.

File	Keying Instruction	Graphic	Description
Programmer	Date	Key	Page of

Sequence Number	Form Type	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV	AW	AX	AY	AZ	BA	BB	BC	BD	BE	BF	BG	BH	BI	BJ	BK	BL	BM	BN	BO	BP	BQ	BR	BS	BT	BU	BV	BW	BX	BY	BZ	CA	CB	CC	CD	CE	CF	CG	CH	CI	CJ	CK	CL	CM	CN	CO	CP	CQ	CR	CS	CT	CU	CV	CW	CX	CY	CZ	DA	DB	DC	DD	DE	DF	DG	DH	DI	DJ	DK	DL	DM	DN	DO	DP	DQ	DR	DS	DT	DU	DV	DW	DX	DY	DZ	EA	EB	EC	ED	EE	EF	EG	EH	EI	EJ	EK	EL	EM	EN	EO	EP	EQ	ER	ES	ET	EU	EV	EW	EX	EY	EZ	FA	FB	FC	FD	FE	FF	FG	FH	FI	FJ	FK	FL	FM	FN	FO	FP	FQ	FR	FS	FT	FU	FV	FW	FX	FY	FZ	GA	GB	GC	GD	GE	GF	GG	GH	GI	GJ	GK	GL	GM	GN	GO	GP	GQ	GR	GS	GT	GU	GV	GW	GX	GY	GZ	HA	HB	HC	HD	HE	HF	HG	HH	HI	HJ	HK	HL	HM	HN	HO	HP	HQ	HR	HS	HT	HU	HV	HW	HX	HY	HZ	IA	IB	IC	ID	IE	IF	IG	IH	II	IJ	IK	IL	IM	IN	IO	IP	IQ	IR	IS	IT	IU	IV	IW	IX	IY	IZ	JA	JB	JC	JD	JE	JF	JG	JH	JI	JJ	JK	JL	JM	JN	JO	JP	JQ	JR	JS	JT	JU	JV	JW	JX	JY	JZ	KA	KB	KC	KD	KE	KF	KG	KH	KI	KJ	KL	KM	KN	KO	KP	KQ	KR	KS	KT	KU	KV	KW	KX	KY	KZ	LA	LB	LC	LD	LE	LF	LG	LH	LI	LJ	LK	LM	LN	LO	LP	LQ	LR	LS	LT	LU	LV	LW	LX	LY	LZ	MA	MB	MC	MD	ME	MF	MG	MH	MI	MJ	MK	ML	MM	MN	MO	MP	MQ	MR	MS	MT	MU	MV	MW	MX	MY	MZ	NA	NB	NC	ND	NE	NF	NG	NH	NI	NJ	NK	NL	NM	NN	NO	NP	NQ	NR	NS	NT	NU	NV	NW	NX	NY	NZ	OA	OB	OC	OD	OE	OF	OG	OH	OI	OJ	OK	OL	OM	ON	OO	OP	OQ	OR	OS	OT	OU	OV	OW	OX	OY	OZ	PA	PB	PC	PD	PE	PF	PG	PH	PI	PJ	PK	PL	PM	PN	PO	PP	PQ	PR	PS	PT	PU	PV	PW	PX	PY	PZ	QA	QB	QC	QD	QE	QF	QG	QH	QI	QJ	QK	QL	QM	QN	QO	QP	QQ	QR	QS	QT	QU	QV	QW	QX	QY	QZ	RA	RB	RC	RD	RE	RF	RG	RH	RI	RJ	RK	RL	RM	RN	RO	RP	RQ	RR	RS	RT	RU	RV	RW	RX	RY	RZ	SA	SB	SC	SD	SE	SF	SG	SH	SI	SJ	SK	SL	SM	SN	SO	SP	SQ	SR	SS	ST	SU	SV	SW	SX	SY	SZ	TA	TB	TC	TD	TE	TF	TG	TH	TI	TJ	TK	TL	TM	TN	TO	TP	TQ	TR	TS	TT	TU	TV	TW	TX	TY	TZ	UA	UB	UC	UD	UE	UF	UG	UH	UI	UJ	UK	UL	UM	UN	UO	UP	UQ	UR	US	UT	UU	UV	UW	UX	UY	UZ	VA	VB	VC	VD	VE	VF	VG	VH	VI	VJ	VK	VL	VM	VN	VO	VP	VQ	VR	VS	VT	VU	VV	VW	VX	VY	VZ	WA	WB	WC	WD	WE	WF	WG	WH	WI	WJ	WK	WL	WM	WN	WO	WP	WQ	WR	WS	WT	WU	WV	WW	WX	WY	WZ	XA	XB	XC	XD	XE	XF	XG	XH	XI	XJ	XK	XL	XM	XN	XO	XP	XQ	XR	XS	XT	XU	XV	XW	XX	XY	XZ	YA	YB	YC	YD	YE	YF	YG	YH	YI	YJ	YK	YL	YM	YN	YO	YP	YQ	YR	YS	YT	YU	YV	YW	YX	YY	YZ	ZA	ZB	ZC	ZD	ZE	ZF	ZG	ZH	ZI	ZJ	ZK	ZL	ZM	ZN	ZO	ZP	ZQ	ZR	ZS	ZT	ZU	ZV	ZW	ZX	ZY	ZZ	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV	AW	AX	AY	AZ	BA	BB	BC	BD	BE	BF	BG	BH	BI	BJ	BK	BL	BM	BN	BO	BP	BQ	BR	BS	BT	BU	BV	BW	BX	BY	BZ	CA	CB	CC	CD	CE	CF	CG	CH	CI	CJ	CK	CL	CM	CN	CO	CP	CQ	CR	CS	CT	CU	CV	CW	CX	CY	CZ	DA	DB	DC	DD	DE	DF	DG	DH	DI	DJ	DK	DL	DM	DN	DO	DP	DQ	DR	DS	DT	DU	DV	DW	DX	DY	DZ	EA	EB	EC	ED	EE	EF	EG	EH	EI	EJ	EK	EL	EM	EN	EO	EP	EQ	ER	ES	ET	EU	EV	EW	EX	EY	EZ	FA	FB	FC	FD	FE	FF	FG	FH	FI	FJ	FK	FL	FM	FN	FO	FP	FQ	FR	FS	FT	FU	FV	FW	FX	FY	FZ	GA	GB	GC	GD	GE	GF	GG	GH	GI	GJ	GK	GL	GM	GN	GO	GP	GQ	GR	GS	GT	GU	GV	GW	GX	GY	GZ	HA	HB	HC	HD	HE	HF	HG	HH	HI	HJ	HK	HL	HM	HN	HO	HP	HQ	HR	HS	HT	HU	HV	HW	HX	HY	HZ	IA	IB	IC	ID	IE	IF	IG	IH	II	IJ	IK	IL	IM	IN	IO	IP	IQ	IR	IS	IT	IU	IV	IW	IX	IY	IZ	JA	JB	JC	JD	JE	JF	JG	JH	JI	JJ	JK	JL	JM	JN	JO	JP	JQ	JR	JS	JT	JU	JV	JW	JX	JY	JZ	KA	KB	KC	KD	KE	KF	KG	KH	KI	KJ	KL	KM	KN	KO	KP	KQ	KR	KS	KT	KU	KV	KW	KX	KY	KZ	LA	LB	LC	LD	LE	LF	LG	LH	LI	LJ	LK	LM	LN	LO	LP	LQ	LR	LS	LT	LU	LV	LW	LX	LY	LZ	MA	MB	MC	MD	ME	MF	MG	MH	MI	MJ	MK	ML	MM	MN	MO	MP	MQ	MR	MS	MT	MU	MV	MW	MX	MY	MZ	NA	NB	NC	ND	NE	NF	NG	NH	NI	NJ	NK	NL	NM	NN	NO	NP	NQ	NR	NS	NT	NU	NV	NW	NX	NY	NZ	OA	OB	OC	OD	OE	OF	OG	OH	OI	OJ	OK	OL	OM	ON	OO	OP	OQ	OR	OS	OT	OU	OV	OW	OX	OY	OZ	PA	PB	PC	PD	PE	PF	PG	PH	PI	PJ	PK	PL	PM	PN	PO	PP	PQ	PR	PS	PT	PU	PV	PW	PX	PY	PZ	QA	QB	QC	QD	QE	QF	QG	QH	QI	QJ	QK	QL	QM	QN	QO	QP	QQ	QR	QS	QT	QU	QV	QW	QX	QY	QZ	RA	RB	RC	RD	RE	RF	RG	RH	RI	RJ	RK	RL	RM	RN	RO	RP	RQ	RR	RS	RT	RU	RV	RW	RX	RY	RZ	SA	SB	SC	SD	SE	SF	SG	SH	SI	SJ	SK	SL	SM	SN	SO
-----------------	-----------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

File Programmer	Date	Keying Instruction	Graphic Key	Description	Page of
--------------------	------	-----------------------	----------------	-------------	------------

Sequence Number	Form Type and/or Comment (A,D,r)	Conditioning				Name	Length	Reference ID	Data Type/Keyboard Shift	Initial Position	Range of Values	Location		Functions
		Indicator	Indicator	Indicator	Indicator							Line	Pos	
A														
A						ORDINDD								EXISTING ORDER REVIEW
A														SEL
A						ITEM	5	0	10					2TEXT('ITEM NUMBER')
A						QTYORD	3	0	10					9TEXT('QUANTITY ORDERED')
A						DESCRP	30	0	10					14TEXT('ITEM DESCRIPTION')
A						PRICE	6	2	10					46TEXT('SELLING PRICE')
A						EXTENS	8	2	10					56EDTCDE(J)
A														TEXT('EXTENSION AMOUNT OF +
A														QTYORD X PRICE')
A														SELC(L(SUB1))
A														SELC(R
A														SFLDSP
A														SFLDSPCT1
A														SELSI7(57)
A														SFLPAG(14)
A														SFLEND
A														OVERLAY
A														LOCK
A														ROLLUP(97 'CONTINUE DISPLAY')
A														CA12(98 'END OF PROGRAM')
A														SETOFF(57 'DISPLAY SUBFILE')
A														SETOFF(58 'OFF-DISPLAY SUBCT1 ON=+
A														CLEAR SUBFILE')
A														1 2'EXISTING ORDER INQUIRY'
A														2'ORDER'
A						ORDERN	5	0	3					8TEXT('ORDER NUMBER')
A														ERRMSG('ORDER NUMBER NOT FOUND' 61)
A														ERRMSG('NO LINES FOR THIS ORDER' 47)
A														ERRMSG('NO CUSTOMER RECORD FOUND FO+ R THIS ORDER' 62)
A														4 2'DATE'
A						ORDDAT	6	0	4					7TEXT('DATE ORDER WAS ENTERED')
A														5 2'CUST #'
A						CUST	5	0	5					8TEXT('CUSTOMER NUMBER')
A						NAME	25	0	3					16TEXT('CUSTOMER NAME')
A						ADDR	20	0	4					16TEXT('CUSTOMER ADDRESS')
A						CITY	20	0	5					16TEXT('CUSTOMER CITY')
A						STATE	2	0	6					16TEXT('CUSTOMER STATE')
A						ZIP	5	0	6					31TEXT('ZIP CODE')
A														1 44'TOTAL'
A						ORDAMT	8	2	1					51TEXT('TOTAL DOLLAR AMOUNT OF THE + ORDER')
A														2 44'STATUS'
A						STSORD	12	0	2					51
A														3 44'OPEN'
A						STSOPN	12	0	3					51
A														4 44'CUSTOMER ORDER'
A						CUSORD	15	0	4					59TEXT('CUSTOMER PURCHASE ORDER + NUMBER')
A														5 44'SHIP VIA'
A						SHPVIA	15	0	5					59TEXT('SHIPPING INSTRUCTIONS')
A														6 44'PRINTED DATE'
A						PRIDAT	6	0	6					57TEXT('DATE ORDER WAS PRINTED')
A														7 29'INVOICE'
A						INVNUM	5	0	7					38TEXT('INVOICE NUMBER')
A														7 64'MTH'
A						ACTMTH	2	0	7					68TEXT('ACCOUNTING MONTH OF SALE')
A														7 72'YEAR'
A						ACTYR	2	0	7					77TEXT('ACCOUNTING YEAR OF SALE')
A														8 2'ITEM'
A														8 8'QTY'
A														8 14'ITEM DESCRIPTION'
A														8 46'PRICE'
A														8 55'EXTENSION'

Figure 71 (Part 2 of 3). Data Description Specifications for an Order Inquiry Program


```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B. ....2. ....3. ....4. ....5. ....6. ....7..IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                01/25/89
 2 000200 PROGRAM-ID. XMPLE773.                                    03/22/89
 3 000300* SAMPLE ORDER INQUIRY PROGRAM                            03/22/89
 4 000400 AUTHOR. PROGRAMMER NAME.                                01/25/89
 5 000500 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.        01/25/89
 6 000600 DATE-WRITTEN. 12/22/88.                                  01/25/89
 7 000700 DATE-COMPILED. 05/24/91 13:29:54 .                      03/01/90
 8 000800 ENVIRONMENT DIVISION.                                    01/25/89
 9 000900 CONFIGURATION SECTION.                                   01/25/89
10 001000 SOURCE-COMPUTER. IBM-AS400.                              01/25/89
11 001100 OBJECT-COMPUTER. IBM-AS400.                              01/25/89
12 001200 INPUT-OUTPUT SECTION.                                    01/25/89
13 001300 FILE-CONTROL..                                          01/25/89
14 001400 SELECT ORDER-HEADER-FILE                                01/25/89
15 001500 ASSIGN TO DATABASE-ORDHDRP                               03/21/89
16 001600 ORGANIZATION IS INDEXED                                 01/25/89
17 001700 ACCESS MODE IS RANDOM                                   01/26/89
18 001800 RECORD KEY IS ORDER OF ORDER-HEADER-RECORD.            01/26/89
19 001900 SELECT ORDER-DETAIL-FILE                                 01/25/89
20 002000 ASSIGN TO DATABASE-ORDDTLP                               03/21/89
21 002100 ORGANIZATION IS INDEXED                                 01/25/89
22 002200 ACCESS IS DYNAMIC                                       01/25/89
23 002300 RECORD KEY IS ORDER-DETAIL-RECORD-KEY.                  01/27/89
24 002400 SELECT CUSTOMER-MASTER-FILE                             01/25/89
25 002500 ASSIGN TO DATABASE-CUSMSTP                               01/25/89
26 002600 ORGANIZATION IS INDEXED                                 01/25/89
27 002700 ACCESS IS RANDOM                                         01/25/89
28 002800 RECORD KEY IS CUST OF CUSTOMER-MASTER-RECORD.          01/26/89
29 002900 SELECT EXISTING-ORDER-DISPLAY-FILE                       01/25/89
30 003000 ASSIGN TO WORKSTATION-ORDINQD                           03/23/89
31 003100 ORGANIZATION IS TRANSACTION                             01/25/89
32 003200 ACCESS IS DYNAMIC                                       01/25/89
33 003300 RELATIVE KEY IS SUBFILE-RECORD-NUMBER                   01/25/89
34 003400 FILE STATUS IS STATUS-CODE-ONE.                          01/25/89
35 003500 DATA DIVISION.                                          01/25/89
36 003600 FILE SECTION.                                            01/25/89
37 003700 FD ORDER-HEADER-FILE                                     01/25/89
38 003800 LABEL RECORDS ARE STANDARD.                               01/25/89
39 003900 01 ORDER-HEADER-RECORD.                                  01/25/89
40 004000 COPY DDS-ORDHDR OF ORDHDRP.                              03/21/89
+000001* I-O FORMAT:ORDHDR FROM FILE ORDHDRP OF LIBRARY XMPLIB    ORDHDR
+000002* ORDER HEADER RECORD                                       ORDHDR
+000003*THE KEY DEFINITIONS FOR RECORD FORMAT ORDHDR              ORDHDR
+000004* NUMBER NAME RETRIEVAL TYPE ALTSEQ                       ORDHDR
+000005* 0001 ORDERN ASCENDING SIGNED NO                          ORDHDR
40 +000006 05 ORDHDR.                                             ORDHDR
41 +000007 06 CUST PIC X(5).                                       ORDHDR
+000008* CUSTOMER NUMBER                                           ORDHDR
42 +000009 06 ORDERN PIC S9(5) COMP-3.                               ORDHDR
+000010* ORDER NUMBER                                              ORDHDR
43 +000011 06 ORDDAT PIC S9(6) COMP-3.                               ORDHDR
+000012* DATE ORDER ENTERED                                        ORDHDR
44 +000013 06 CUSORD PIC X(15).                                       ORDHDR
+000014* CUSTOMER PURCHASE ORDER NUMBER                            ORDHDR
45 +000015 06 SHPVIA PIC X(15).                                       ORDHDR
+000016* SHIPPING INSTRUCTIONS                                     ORDHDR
46 +000017 06 ORDSTS PIC S9(1) COMP-3.                               ORDHDR
+000018* ORDER STATUS 1PCS 2CNT 3CHK 4RDY 5PRT 6PC                ORDHDR
47 +000019 06 OPRNAM PIC X(10).                                       ORDHDR
+000020* OPERATOR WHO ENTERED ORD                                   ORDHDR
48 +000021 06 ORDAMT PIC S9(6)V9(2) COMP-3.                           ORDHDR
+000022* DOLLAR AMOUNT OF ORDER                                    ORDHDR
49 +000023 06 CUSTYP PIC S9(1) COMP-3.                               ORDHDR
+000024* CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT                ORDHDR
50 +000025 06 INVNUM PIC S9(5) COMP-3.                               ORDHDR
+000026* INVOICE NUMBER                                           ORDHDR
51 +000027 06 PRDAT PIC S9(6) COMP-3.                               ORDHDR
+000028* DATE ORDER WAS PRINTED                                    ORDHDR
52 +000029 06 OPNSTS PIC S9(1) COMP-3.                               ORDHDR
+000030* ORDER OPEN STATUS 1=OPEN 2= CLOSE 3=CANCEL                ORDHDR
53 +000031 06 TOTLIN PIC S9(3) COMP-3.                               ORDHDR
+000032* TOTAL LINE ITEMS IN ORDER                                 ORDHDR
54 +000033 06 ACTMTH PIC S9(2) COMP-3.                               ORDHDR
+000034* ACCOUNTING MONTH OF SALE                                  ORDHDR
55 +000035 06 ACTYR PIC S9(2) COMP-3.                               ORDHDR
+000036* ACCOUNTING YEAR OF SALE                                   ORDHDR

```

Figure 72 (Part 1 of 7). Example of an Order Inquiry Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
56 +000037          06 STATE          PIC X(2).          ORDHDR
+000038*          STATE          ORDHDR
57 +000039          06 AMPAID          PIC S9(6)V9(2)  COMP-3.        ORDHDR
+000040*          AMOUNT PAID          ORDHDR
004100
58 004200 FD ORDER-DETAIL-FILE
59 004300 LABEL RECORDS ARE STANDARD.
60 004400 01 ORDER-DETAIL-RECORD.
61 004500 COPY DDS-ORDDTL OF ORDDTLP.
+000001* I-0 FORMAT:ORDDTL FROM FILE ORDDTLP OF LIBRARY XMLPLIB ORDDTL
+000002* ORDER DETAIL RECORD ORDDTL
+000003*THE KEY DEFINITIONS FOR RECORD FORMAT ORDDTL ORDDTL
+000004* NUMBER NAME RETRIEVAL TYPE ALTSEQ ORDDTL
+000005* 0001 ORDERN ASCENDING SIGNED NO ORDDTL
+000006* 0002 LINNUM ASCENDING SIGNED NO ORDDTL
62 +000007 05 ORDDTL. ORDDTL
63 +000008 06 CUST PIC X(5). ORDDTL
+000009* CUSTOMER NUMBER ORDDTL
64 +000010 06 ORDERN PIC S9(5) COMP-3. ORDDTL
+000011* ORDER NUMBER ORDDTL
65 +000012 06 LINNUM PIC S9(3) COMP-3. ORDDTL
+000013* LINE NUMBER OF LINE IN ORDER ORDDTL
66 +000014 06 ITEM PIC S9(5) COMP-3. ORDDTL
+000015* ITEM NUMBER ORDDTL
67 +000016 06 QTYORD PIC S9(3) COMP-3. ORDDTL
+000017* QUANTITY ORDERED ORDDTL
68 +000018 06 DESCRP PIC X(30). ORDDTL
+000019* ITEM DESCRIPTION ORDDTL
69 +000020 06 PRICE PIC S9(4)V9(2) COMP-3. ORDDTL
+000021* SELLING PRICE ORDDTL
70 +000022 06 EXTENS PIC S9(6)V9(2) COMP-3. ORDDTL
+000023* EXTENSION AMOUNT OF QTYORD X PRICE ORDDTL
71 +000024 06 WHSLOC PIC X(3). ORDDTL
+000025* BIN NO. ORDDTL
72 +000026 06 ORDDAT PIC S9(6) COMP-3. ORDDTL
+000027* DATE ORDER WAS ENTERED ORDDTL
73 +000028 06 CUSTYP PIC S9(1) COMP-3. ORDDTL
+000029* CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5= ORDDTL
74 +000030 06 STATE PIC X(2). ORDDTL
+000031* STATE ORDDTL
75 +000032 06 ACTMTH PIC S9(2) COMP-3. ORDDTL
+000033* ACCOUNTING MONTH OF SALE ORDDTL
76 +000034 06 ACTYR PIC S9(2) COMP-3. ORDDTL
+000035* ACCOUNTING YEAR OF SALE ORDDTL
77 004600 66 ORDER-DETAIL-RECORD-KEY RENAMES ORDERN THRU LINNUM.
004700
78 004800 FD CUSTOMER-MASTER-FILE
79 004900 LABEL RECORDS ARE STANDARD.
80 005000 01 CUSTOMER-MASTER-RECORD.
81 005100 COPY DDS-CUSMST OF CUSMSTP.
+000001* I-0 FORMAT:CUSMST FROM FILE CUSMSTP OF LIBRARY XMLPLIB CUSMST
+000002* CUSTOMER MASTER RECORD CUSMST
+000003*THE KEY DEFINITIONS FOR RECORD FORMAT CUSMST CUSMST
+000004* NUMBER NAME RETRIEVAL TYPE ALTSEQ CUSMST
+000005* 0001 CUST ASCENDING AN NO CUSMST
82 +000006 05 CUSMST. CUSMST
83 +000007 06 CUST PIC X(5). CUSMST
+000008* CUSTOMER NUMBER CUSMST
84 +000009 06 NAME PIC X(25). CUSMST
+000010* CUSTOMER NAME CUSMST
85 +000011 06 ADDR PIC X(20). CUSMST
+000012* CUSTOMER ADDRESS CUSMST
86 +000013 06 CITY PIC X(20). CUSMST
+000014* CUSTOMER CITY CUSMST
87 +000015 06 STATE PIC X(2). CUSMST
+000016* STATE CUSMST
88 +000017 06 ZIP PIC S9(5) COMP-3. CUSMST
+000018* ZIP CODE CUSMST
89 +000019 06 SRHCOD PIC X(6). CUSMST
+000020* CUSTOMER NUMBER SEARCH CODE CUSMST
90 +000021 06 CUSTYP PIC S9(1) COMP-3. CUSMST
+000022* CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT CUSMST
91 +000023 06 ARBAL PIC S9(6)V9(2) COMP-3. CUSMST
+000024* ACCOUNTS REC. BALANCE CUSMST
92 +000025 06 ORDBAL PIC S9(6)V9(2) COMP-3. CUSMST

```

Figure 72 (Part 2 of 7). Example of an Order Inquiry Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
+000026*                A/R AMT. IN ORDER FILE          CUSMST
93 +000027              06 LSTAMT          PIC S9(6)V9(2) COMP-3.    CUSMST
+000028*                LAST AMT. PAID IN A/R          CUSMST
94 +000029              06 LSTDAT          PIC S9(6)          COMP-3.    CUSMST
+000030*                LAST DATE PAID IN A/R          CUSMST
95 +000031              06 CRDLMT          PIC S9(6)V9(2) COMP-3.    CUSMST
+000032*                CUSTOMER CREDIT LIMIT          CUSMST
96 +000033              06 SLSYR          PIC S9(8)V9(2) COMP-3.    CUSMST
+000034*                CUSTOMER SALES THIS YEAR        CUSMST
97 +000035              06 SLSLYR         PIC S9(8)V9(2) COMP-3.    CUSMST
+000036*                CUSTOMER SALES LAST YEAR        CUSMST
005200

98 005300 FD EXISTING-ORDER-DISPLAY-FILE
99 005400 LABEL RECORDS ARE OMITTED.
100 005500 01 EXISTING-ORDER-DISPLAY-RECORD.
101 005600 COPY DDS-ALL-FORMATS OF ORDINQD.
102 +000001              05 ORDINQD-RECORD PIC X(171).          <-ALL-FMTS
+000002*                I-O FORMAT:SUB1 FROM FILE ORDINQD OF LIBRARY XMPLIB <-ALL-FMTS
+000003*                <-ALL-FMTS
103 +000004              05 SUB1 REDEFINES ORDINQD-RECORD.    <-ALL-FMTS
104 +000005              06 ITEM          PIC S9(5).          <-ALL-FMTS
+000006*                ITEM NUMBER                    <-ALL-FMTS
105 +000007              06 QTYORD          PIC S9(3).          <-ALL-FMTS
+000008*                QUANTITY ORDERED              <-ALL-FMTS
106 +000009              06 DESCRP          PIC X(30).          <-ALL-FMTS
+000010*                ITEM DESCRIPTION              <-ALL-FMTS
107 +000011              06 PRICE          PIC S9(4)V9(2).      <-ALL-FMTS
+000012*                SELLING PRICE                <-ALL-FMTS
108 +000013              06 EXTENS          PIC S9(6)V9(2).      <-ALL-FMTS
+000014*                EXTENSION AMOUNT OF QTYORD X PRICE <-ALL-FMTS
+000015*                INPUT FORMAT:SUBCTL1 FROM FILE ORDINQD OF LIBRARY XMPLIB <-ALL-FMTS
+000016*                <-ALL-FMTS
109 +000017              05 SUBCTL1-I REDEFINES ORDINQD-RECORD. <-ALL-FMTS
110 +000018              06 SUBCTL1-I-INDIC. <-ALL-FMTS
111 +000019              07 IN97          PIC 1 INDIC 97.      <-ALL-FMTS
+000020*                CONTINUE DISPLAY              <-ALL-FMTS
112 +000021              07 IN98          PIC 1 INDIC 98.      <-ALL-FMTS
+000022*                END OF PROGRAM                <-ALL-FMTS
113 +000023              07 IN57          PIC 1 INDIC 57.      <-ALL-FMTS
+000024*                DISPLAY SUBFILE                <-ALL-FMTS
114 +000025              07 IN58          PIC 1 INDIC 58.      <-ALL-FMTS
+000026*                OFF = DISPLAY SUBCTL1 ON = CLEAR SUBFILE <-ALL-FMTS
115 +000027              07 IN61          PIC 1 INDIC 61.      <-ALL-FMTS
+000028*                ORDER NUMBER NOT FOUND        <-ALL-FMTS
116 +000029              07 IN47          PIC 1 INDIC 47.      <-ALL-FMTS
+000030*                NO LINE FOR THIS ORDER        <-ALL-FMTS
117 +000031              07 IN62          PIC 1 INDIC 62.      <-ALL-FMTS
+000032*                NO CUSTOMER RECORD            <-ALL-FMTS
118 +000033              06 ORDERN          PIC S9(5).          <-ALL-FMTS
+000034*                ORDER NUMBER                  <-ALL-FMTS
+000035*                OUTPUT FORMAT:SUBCTL1 FROM FILE ORDINQD OF LIBRARY XMPLIB <-ALL-FMTS
+000036*                <-ALL-FMTS
119 +000037              05 SUBCTL1-0 REDEFINES ORDINQD-RECORD. <-ALL-FMTS
120 +000038              06 SUBCTL1-0-INDIC. <-ALL-FMTS
121 +000039              07 IN58          PIC 1 INDIC 58.      <-ALL-FMTS
+000040*                OFF = DISPLAY SUBCTL1 ON = CLEAR SUBFILE <-ALL-FMTS
122 +000041              07 IN57          PIC 1 INDIC 57.      <-ALL-FMTS
+000042*                DISPLAY SUBFILE                <-ALL-FMTS
123 +000043              07 IN45          PIC 1 INDIC 45.      <-ALL-FMTS
124 +000044              07 IN47          PIC 1 INDIC 47.      <-ALL-FMTS
+000045*                NO LINE FOR THIS ORDER        <-ALL-FMTS
125 +000046              07 IN61          PIC 1 INDIC 61.      <-ALL-FMTS
+000047*                ORDER NUMBER NOT FOUND        <-ALL-FMTS
126 +000048              07 IN62          PIC 1 INDIC 62.      <-ALL-FMTS
+000049*                NO CUSTOMER RECORD            <-ALL-FMTS
127 +000050              06 ORDERN          PIC S9(5).          <-ALL-FMTS
+000051*                ORDER NUMBER                  <-ALL-FMTS
128 +000052              06 ORDDAT          PIC S9(6).          <-ALL-FMTS
+000053*                DATE ORDER WAS ENTERED        <-ALL-FMTS
129 +000054              06 CUST          PIC X(5).          <-ALL-FMTS
+000055*                CUSTOMER NUMBER              <-ALL-FMTS
130 +000056              06 NAME          PIC X(25).          <-ALL-FMTS
+000057*                CUSTOMER NAME                <-ALL-FMTS
131 +000058              06 ADDR          PIC X(20).          <-ALL-FMTS
+000059*                CUSTOMER ADDRESS              <-ALL-FMTS
132 +000060              06 CITY          PIC X(20).          <-ALL-FMTS

```

Figure 72 (Part 3 of 7). Example of an Order Inquiry Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B.+. . . . 2. . . . +. . . . 3. . . . +. . . . 4. . . . +. . . . 5. . . . +. . . . 6. . . . +. . . . 7. IDENTFCN S COPYNAME  CHG DATE
+000061*                CUSTOMER CITY                <-ALL-FMTS
133 +000062             06 STATE                      PIC X(2).                <-ALL-FMTS
+000063*                CUSTOMER STATE                <-ALL-FMTS
134 +000064             06 ZIP                        PIC S9(5).                <-ALL-FMTS
+000065*                ZIP CODE                      <-ALL-FMTS
135 +000066             06 ORDAMT                    PIC S9(6)V9(2).          <-ALL-FMTS
+000067*                TOTAL AMOUNT OF ORDER        <-ALL-FMTS
136 +000068             06 STSORD                    PIC X(12).               <-ALL-FMTS
137 +000069             06 STSOPN                    PIC X(12).               <-ALL-FMTS
138 +000070             06 CUSORD                    PIC X(15).               <-ALL-FMTS
+000071*                CUSTOMER PURCHASE ORDER NUMBER <-ALL-FMTS
139 +000072             06 SHPVIA                    PIC X(15).               <-ALL-FMTS
+000073*                SHIPPING INSTRUCTIONS        <-ALL-FMTS
140 +000074             06 PRDAT                      PIC S9(6).               <-ALL-FMTS
+000075*                DATE ORDER WAS PRINTED      <-ALL-FMTS
141 +000076             06 INVNUM                    PIC S9(5).               <-ALL-FMTS
+000077*                INVOICE NUMBER              <-ALL-FMTS
142 +000078             06 ACTMTH                    PIC S9(2).               <-ALL-FMTS
+000079*                ACCOUNTING MONTH OF SALE    <-ALL-FMTS
143 +000080             06 ACTYR                      PIC S9(2).               <-ALL-FMTS
+000081*                ACCOUNTING YEAR OF SALE     <-ALL-FMTS
005700
144 005800 WORKING-STORAGE SECTION.
145 005900 01 EXISTING-ORDER-DISPLAY-KEY.
146 006000 05 SUBFILE-RECORD-NUMBER                PIC 9(2)
147 006100                                         VALUE ZERO.
006200
148 006300 01 ORDER-STATUS-COMMENT-VALUES.
149 006400 05 FILLER                              PIC X(12)
150 006500                                         VALUE "1-IN PROCESS".
151 006600 05 FILLER                              PIC X(12)
152 006700                                         VALUE "2-CONTINUED ".
153 006800 05 FILLER                              PIC X(12)
154 006900                                         VALUE "3-CREDIT CHK".
155 007000 05 FILLER                              PIC X(12)
156 007100                                         VALUE "4-READY PRT ".
157 007200 05 FILLER                              PIC X(12)
158 007300                                         VALUE "5-PRINTED ".
159 007400 05 FILLER                              PIC X(12)
160 007500                                         VALUE "6-PICKED ".
161 007600 05 FILLER                              PIC X(12)
162 007700                                         VALUE "7-INVOICED ".
163 007800 05 FILLER                              PIC X(12)
164 007900                                         VALUE "8-INVALID ".
165 008000 05 FILLER                              PIC X(12)
166 008100                                         VALUE "9-CANCELED ".
008200
167 008300 01 ORDER-STATUS-COMMENT-TABLE
168 008400 REDEFINES ORDER-STATUS-COMMENT-VALUES.
169 008500 05 ORDER-STATUS OCCURS 9 TIMES.
170 008600 10 ORDER-STATUS-COMMENT                PIC X(12).
008700
171 008800 01 OPEN-STATUS-COMMENT-VALUES.
172 008900 05 FILLER                              PIC X(12)
173 009000                                         VALUE "1-OPEN ".
174 009100 05 FILLER                              PIC X(12)
175 009200                                         VALUE "2-CLOSED ".
176 009300 05 FILLER                              PIC X(12)
177 009400                                         VALUE "3-CANCELED ".
009500
178 009600 01 OPEN-STATUS-COMMENT-TABLE
179 009700 REDEFINES OPEN-STATUS-COMMENT-VALUES.
180 009800 05 OPEN-STATUS OCCURS 3 TIMES.
181 009900 10 OPEN-STATUS-COMMENT                PIC X(12).
010000
182 010100 01 ERRHDL-PARAMETERS.
183 010200 05 STATUS-CODE-ONE                      PIC X(2).
184 010300 88 SUBFILE-IS-FULL                      VALUE "9M".
010400
185 010500 01 ERRPGM-PARAMETERS.
186 010600 05 DISPLAY-PARAMETER                    PIC X(8)
187 010700                                         VALUE "ORD220D ".
188 010800 05 DUMMY-ONE                            PIC X(6)
189 010900                                         VALUE SPACES.
190 011000 05 DUMMY-TWO                            PIC X(8)
191 011100                                         VALUE SPACES.

```

Figure 72 (Part 4 of 7). Example of an Order Inquiry Program

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
192 011200 05 STATUS-CODE-TWO.
193 011300 10 PRIMARY PIC X(1).
194 011400 10 SECONDARY PIC X(1).
195 011500 10 FILLER PIC X(5).
196 011600 VALUE SPACES.
    011700
197 011800 01 SWITCH-AREA.
198 011900 05 SW01 PIC 1.
199 012000 88 NO-MORE-DETAIL-LINE-ITEMS VALUE B"1".
200 012100 88 MORE-DETAIL-LINE-ITEMS-EXIST VALUE B"0".
201 012200 05 SW02 PIC 1.
202 012300 88 WRITE-DISPLAY VALUE B"1".
203 012400 88 READ-DISPLAY VALUE B"0".
204 012500 05 SW03 PIC 1.
205 012600 88 SUBCTL1-FORMAT VALUE B"1".
206 012700 88 NOT-SUBCTL1-FORMAT VALUE B"0".
207 012800 05 SW04 PIC 1.
208 012900 88 SUB1-FORMAT VALUE B"1".
209 013000 88 NOT-SUB1-FORMAT VALUE B"0".
    013100
210 013200 01 INDICATOR-AREA.
211 013300 05 IN98 PIC 1 INDIC 98.
212 013400 88 END-OF-EXISTING-ORDER-INQUIRY VALUE B"1".
213 013500 05 IN97 PIC 1 INDIC 97.
214 013600 88 CONTINUE-DETAIL-LINES-DISPLAY VALUE B"1".
215 013700 05 IN62 PIC 1 INDIC 62.
216 013800 88 CUSTOMER-NOT-FOUND VALUE B"1".
217 013900 88 CUSTOMER-EXIST VALUE B"0".
218 014000 05 IN61 PIC 1 INDIC 61.
219 014100 88 ORDER-NOT-FOUND VALUE B"1".
220 014200 88 ORDER-EXIST VALUE B"0".
221 014300 05 IN58 PIC 1 INDIC 58.
222 014400 88 CLEAR-SUBFILE VALUE B"1".
223 014500 88 DISPLAY-SUBFILE-CONTROL VALUE B"0".
224 014600 05 IN57 PIC 1 INDIC 57.
225 014700 88 DISPLAY-SUBFILE VALUE B"1".
226 014800 05 IN47 PIC 1 INDIC 47.
227 014900 88 NO-DETAIL-LINES-FOR-ORDER VALUE B"1".
228 015000 88 DETAIL-LINES-FOR-ORDER-EXIST VALUE B"0".
229 015100 05 IN45 PIC 1 INDIC 45.
230 015200 88 END-OF-ORDER VALUE B"1".
    015300
231 015400 PROCEDURE DIVISION.
    015500
    015600 DECLARATIVES.
    015700 TRANSACTION-ERROR SECTION.
    015800 USE AFTER STANDARD ERROR PROCEDURE
    015900 EXISTING-ORDER-DISPLAY-FILE.
    016000 WORK-STATION-ERROR-HANDLER.
232 016100 IF SUBFILE-IS-FULL THEN
    016200 NEXT SENTENCE
    016300 ELSE
233- 016400 DISPLAY "WORK-STATION ERROR" STATUS-CODE-ONE.
    016500 END DECLARATIVES.
    016600
    016700 INQUIRY-INTO-EXISTING-ORDER SECTION.
    016800 MAINLINE-ROUTINE.
234 016900 PERFORM SET-UP-ROUTINE.
235 017000 PERFORM EXISTING-ORDER-INQUIRY
    017100 UNTIL END-OF-EXISTING-ORDER-INQUIRY.
236 017200 PERFORM CLEAN-UP-ROUTINE.
    017300
    017400 SET-UP-ROUTINE.
237 017500 OPEN INPUT ORDER-HEADER-FILE
    017600 ORDER-DETAIL-FILE
    017700 CUSTOMER-MASTER-FILE
    017800 I-O EXISTING-ORDER-DISPLAY-FILE.
238 017900 MOVE SPACES TO CUST OF SUBCTL1-0
    018000 NAME OF SUBCTL1-0
    018100 ADDR OF SUBCTL1-0
    018200 CITY OF SUBCTL1-0
    018300 STATE OF SUBCTL1-0
    018400 STSORD OF SUBCTL1-0
    018500 STSOPN OF SUBCTL1-0
    018600 CUSORD OF SUBCTL1-0.

```

Figure 72 (Part 5 of 7). Example of an Order Inquiry Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B..+...2....+...3....+...4....+...5....+...6....+...7..IDENTFCN S COPYNAME  CHG DATE
239 018700 MOVE ZEROS TO   ORDERN OF SUBCTL1-0
    018800                ORDDAT OF SUBCTL1-0
    018900                ZIP   OF SUBCTL1-0
    019000                ORDAMT OF SUBCTL1-0
    019100                PRDAT  OF SUBCTL1-0
    019200                INVNUM OF SUBCTL1-0
    019300                ACTMTH OF SUBCTL1-0
    019400                ACTYR  OF SUBCTL1-0.
240 019500 MOVE ALL B'0' TO INDICATOR-AREA.
241 019600 SET READ-DISPLAY
    019700 NOT-SUBCTL1-FORMAT
    019800 NOT-SUB1-FORMAT TO TRUE.
242 019900 MOVE CORR INDICATOR-AREA TO SUBCTL1-0-INDIC.
243 020000 WRITE EXISTING-ORDER-DISPLAY-RECORD FORMAT IS "SUBCTL1".
244 020100 READ EXISTING-ORDER-DISPLAY-FILE RECORD.
245 020200 MOVE CORR SUBCTL1-I-INDIC TO INDICATOR-AREA.
    020300
    020400 EXISTING-ORDER-INQUIRY.
246 020500 IF CONTINUE-DETAIL-LINES-DISPLAY THEN
247 020600     PERFORM READ-NEXT-ORDER-DETAIL-RECORD
248 020700     IF MORE-DETAIL-LINE-ITEMS-EXIST THEN
249 020800         IF ORDERN OF ORDER-DETAIL-RECORD IS NOT EQUAL TO
    020900             ORDERN OF ORDER-HEADER-RECORD THEN
250 021000             SET DISPLAY-SUBFILE TO TRUE
251 021100             SET NO-DETAIL-LINES-FOR-ORDER TO TRUE
    021200     ELSE
252 021300         PERFORM SUBFILE-SET-UP
    021400     ELSE
253 021500         SET DISPLAY-SUBFILE TO TRUE
254 021600         SET NO-DETAIL-LINES-FOR-ORDER TO TRUE
    021700     ELSE
255 021800         PERFORM ORDER-NUMBER-VALIDATION.
256 021900     MOVE CORR INDICATOR-AREA TO SUBCTL1-0-INDIC.
257 022000     SET WRITE-DISPLAY TO TRUE.
258 022100     SET SUBCTL1-FORMAT TO TRUE.
259 022200     WRITE EXISTING-ORDER-DISPLAY-RECORD FORMAT IS "SUBCTL1".
260 022300     READ EXISTING-ORDER-DISPLAY-FILE RECORD.
261 022400     MOVE CORR SUBCTL1-I-INDIC TO INDICATOR-AREA.
    022500 ORDER-NUMBER-VALIDATION.
262 022600     PERFORM READ-ORDER-HEADER-FILE.
263 022700     IF ORDER-EXIST THEN
264 022800         PERFORM READ-CUSTOMER-MASTER-FILE
265 022900         IF CUSTOMER-EXIST THEN
266 023000             PERFORM READ-FIRST-ORDER-DETAIL-RECORD
267 023100             IF DETAIL-LINES-FOR-ORDER-EXIST THEN
268 023200                 PERFORM SUBFILE-SET-UP
    023300     ELSE
    023400         NEXT SENTENCE
    023500     ELSE
    023600         NEXT SENTENCE
    023700     ELSE
    023800         NEXT SENTENCE.
    023900 READ-ORDER-HEADER-FILE.
269 024000 MOVE ORDERN OF SUBCTL1-I OF EXISTING-ORDER-DISPLAY-RECORD
    024100     TO ORDERN OF ORDER-HEADER-RECORD.
270 024200 READ ORDER-HEADER-FILE
271 024300 INVALID KEY SET ORDER-NOT-FOUND TO TRUE.
    024400 READ-CUSTOMER-MASTER-FILE.
272 024500 MOVE CUST OF ORDER-HEADER-RECORD
    024600     TO CUST OF CUSTOMER-MASTER-RECORD.
273 024700 READ CUSTOMER-MASTER-FILE
274 024800 INVALID KEY SET CUSTOMER-NOT-FOUND TO TRUE.
    024900 READ-FIRST-ORDER-DETAIL-RECORD.
275 025000 MOVE ORDERN OF ORDER-HEADER-RECORD
    025100     TO ORDERN OF ORDER-DETAIL-RECORD.
276 025200 MOVE 1 TO LINNUM OF ORDER-DETAIL-RECORD.
277 025300 READ ORDER-DETAIL-FILE
278 025400 INVALID KEY SET NO-DETAIL-LINES-FOR-ORDER TO TRUE.
    025500 SUBFILE-SET-UP.
279 025600 SET CLEAR-SUBFILE TO TRUE.
280 025700 MOVE CORR INDICATOR-AREA TO SUBCTL1-0-INDIC.
281 025800 SET WRITE-DISPLAY TO TRUE.
282 025900 SET SUBCTL1-FORMAT TO TRUE.
283 026000 WRITE EXISTING-ORDER-DISPLAY-RECORD FORMAT IS "SUBCTL1".
284 026100 SET DISPLAY-SUBFILE-CONTROL TO TRUE.

```

Figure 72 (Part 6 of 7). Example of an Order Inquiry Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
285 026200  PERFORM BUILD-DISPLAY-SUBFILE
      026300  UNTIL NO-MORE-DETAIL-LINE-ITEMS
      026400  OR SUBFILE-IS-FULL.
286 026500  MOVE CORR ORDHDR OF ORDER-HEADER-RECORD
      026600  TO SUBCTL1-0 OF EXISTING-ORDER-DISPLAY-RECORD.
287 026700  MOVE CORR CUSMST OF CUSTOMER-MASTER-RECORD
      026800  TO SUBCTL1-0 OF EXISTING-ORDER-DISPLAY-RECORD.
288 026900  MOVE ORDER-STATUS(ORDSTS) TO STSORD.
289 027000  MOVE OPEN-STATUS(OPNSTS) TO STSOPN.
290 027100  SET MORE-DETAIL-LINE-ITEMS-EXIST TO TRUE.
291 027200  MOVE ZEROS TO SUBFILE-RECORD-NUMBER.
      027300  BUILD-DISPLAY-SUBFILE.
292 027400  MOVE CORR ORDDTL OF ORDER-DETAIL-RECORD
      027500  TO SUB1 OF EXISTING-ORDER-DISPLAY-RECORD.
293 027600  SET WRITE-DISPLAY TO TRUE.
294 027700  SET SUB1-FORMAT TO TRUE.
295 027800  ADD 1 TO SUBFILE-RECORD-NUMBER.
296 027900  WRITE SUBFILE EXISTING-ORDER-DISPLAY-RECORD FORMAT IS "SUB1".
297 028000  IF SUBFILE-IS-FULL THEN
298 028100  SET DISPLAY-SUBFILE TO TRUE
      028200  ELSE
299 028300  PERFORM READ-NEXT-ORDER-DETAIL-RECORD
300 028400  IF NO-MORE-DETAIL-LINE-ITEMS THEN
      028500  NEXT SENTENCE
      028600  ELSE
301 028700  IF ORDERN OF ORDER-DETAIL-RECORD IS NOT EQUAL TO
      028800  ORDERN OF ORDER-HEADER-RECORD THEN
302 028900  SET DISPLAY-SUBFILE TO TRUE
303 029000  SET NO-MORE-DETAIL-LINE-ITEMS TO TRUE
      029100  ELSE
      029200  NEXT SENTENCE.
      029300  READ-NEXT-ORDER-DETAIL-RECORD.
304 029400  READ ORDER-DETAIL-FILE NEXT RECORD
305 029500  AT END SET DISPLAY-SUBFILE TO TRUE
306 029600  SET NO-MORE-DETAIL-LINE-ITEMS TO TRUE.
      029700  CLEAN-UP-ROUTINE.
307 029800  CLOSE ORDER-HEADER-FILE
      029900  ORDER-DETAIL-FILE
      030000  CUSTOMER-MASTER-FILE
      030100  EXISTING-ORDER-DISPLAY-FILE.
308 030200  STOP RUN.
          ***** END OF SOURCE *****

```

Figure 72 (Part 7 of 7). Example of an Order Inquiry Program

This is the initial order-entry prompt display written to the work station:

```

Existing Order Entry          Total 000000000
                              Status
Order 12400                  Open
Date 000000                  Customer order
Cust #                        Ship via
                              Invoice 00000 Printed date 000000
                              Mth 00 Year 00
Item Qty Item Description    Price Extension

```


This display appears if there are detail order records for the customer whose order number was entered in the first display:

Existing Order Entry		Total	007426656	
		Status	7-INVOICED	
Order	17924	ABC HARDWARE LTD.	Open	2-CLOSED
Date	110587	123 ANYWHERE AVE.	Customer order	TESTCS17933001I
Cust #	11200	TORONTO	Ship via	TRUCKCO
		ONT	Printed date	082788
		M4K 0A0		
		Invoice 17924	Mth 12	Year 88
Item	Qty	Item Description	Price	Extension
33001	003	TORQUE WRENCH 75LB 14 INCH	009115	273.45
33100	001	TORQUE WRENCH W/GAUGE 200 LB	015777	650.95
44529	004	WOOD CHISEL - 3 1/4	006840	56.87
44958	002	POWER DRILL 1/2 REV	008200	797.50
46102	001	WROUGHT IRON RAILING 4FTX6FT	007930	237.75
46201	001	WROUGHT IRON HAND RAIL 6FT	007178	77.35
47902	002	ESCUTCHEON BRASS 15X4 INCHES	044488	213.00

This display appears if the ORDHDRP file does not contain a record for the order number entered on the first display:

Existing Order Entry		Total	000000000	
		Status		
Order	12400	Open		
Date	000000	Customer order		
Cust #		Ship via		
		Printed date	000000	
		00000		
		Invoice 00000	Mth 00	Year 00
Item	Qty	Item Description	Price	Extension
Order number not found				

A Payment Update Program

Figure 74 on page 217 shows an example of a payment update program, PAYUPDT. For the related DDS, see Figure 73 on page 214. For the related display-screen examples, see page 225. For the DDS for the customer master file, CUSMSTP, refer to Figure 69 on page 198.

In this example, payments from customers are registered. The clerk is prompted to enter one or more customer numbers and the amount of money to be credited

to each customer's account. The program checks the customer number and unconditionally accepts any payment for an existing customer who has invoices outstanding. If an overpayment will result from the amount of the payment from a customer, the clerk is given the option to accept or reject the payment. If no customer record exists for a customer number, an error message is issued. Payments can be entered until the clerk ends the program by pressing F12.

File		Keying Instruction	Graphic									Description	Page	of
Programmer	Date		Key											

Sequence Number	Form Type	And/Or Comment (A/D/+)	Conditioning				Name	Length	Reference (R)	Data Type/Repeat Shift	Repeat Positions	Location		Functions
			Indicator	Not (N)	Indicator	Not (N)						Line	Pos	
1	A	*												
2	A	*												
3	A	*												
4	A	*												
5	A	*												
6	A	*												
7	A	*												
8	A	*												
9	A	*												
10	A	*												
11	A	*												
12	A	*												
13	A	*												
14	A	*												
15	A	*												
16	A	*												
17	A	*												
18	A	*												
19	A	*												
20	A	*												
21	A	*												
22	A	*												
23	A	*												
24	A	*												
25	A	*												
26	A	*												
27	A	*												
28	A	*												
29	A	*												
30	A	*												
31	A	*												
32	A	*												
33	A	*												
34	A	*												
35	A	*												
36	A	*												
37	A	*												
38	A	*												
39	A	*												
40	A	*												
41	A	*												
42	A	*												
43	A	*												
44	A	*												
45	A	*												
46	A	*												
47	A	*												
48	A	*												
49	A	*												
50	A	*												
51	A	*												
52	A	*												
53	A	*												
54	A	*												
55	A	*												
56	A	*												
57	A	*												
58	A	*												
59	A	*												
60	A	*												
61	A	*												
62	A	*												
63	A	*												
64	A	*												
65	A	*												
66	A	*												
67	A	*												
68	A	*												
69	A	*												
70	A	*												
71	A	*												
72	A	*												
73	A	*												
74	A	*												
75	A	*												
76	A	*												
77	A	*												
78	A	*												
79	A	*												
80	A	*												
81	A	*												
82	A	*												
83	A	*												
84	A	*												
85	A	*												
86	A	*												
87	A	*												
88	A	*												
89	A	*												
90	A	*												
91	A	*												
92	A	*												
93	A	*												
94	A	*												
95	A	*												
96	A	*												
97	A	*												
98	A	*												
99	A	*												
100	A	*												

Figure 73 (Part 1 of 3). Example of a Data Description Specification for a Payment Update Program

File Programmer	Date	Keying Instruction	Graphic Key	Description	Page of
--------------------	------	-----------------------	----------------	-------------	------------

Sequence Number	Form Type 7 And/or Comment (A/D/E)	Indicator 10	Indicator 11	Indicator 12	Indicator 13	Indicator 14	Indicator 15	Indicator 16	Indicator 17	Type of Name of Spec. (R/W/A/K/S/D)	Name	Length	Reference (R)	Data Type/Keyboard Shift Decimal Positions	Location		Functions
															Line	Pos	
A*											DDS FOR THE DISPLAY DEVICE FILE PAYUPDTD						
A*											ACCOUNTS RECEIVABLE INTERACTIVE PAYMENT UPDATE						
A*											R SUBFILE1						SFL
A*																	TEXT('SUBFILE FOR CUSTOMER PAYMENT')
A*											ACPPMT	4A	I	5	4		TEXT('ACCEPT PAYMENT')
A																	VALUES('YES' 'NO')
A	51																DSPATR(RI MDT)
A	N51																DSPATR(ND PR)
A*											CUST	5	B	5	15		TEXT('CUSTOMER NUMBER')
A	52																DSPATR(RI)
A	53																DSPATR(ND)
A	54																DSPATR(PR)
A*											AMPAID	8	02B	5	24		TEXT('AMOUNT PAID')
A																	CHECK(FE)
A																	AUTO(RAB)
A																	CMP(GT 0)
A	52																DSPATR(RI)
A	53																DSPATR(ND)
A	54																DSPATR(PR)
A*											ECPMSG	31A	0	5	37		TEXT('EXCEPTION MESSAGE')
A	52																DSPATR(RI)
A	53																DSPATR(ND)
A	54																DSPATR(PR)
A*											OVRPMT	8Y	20	5	70		TEXT('OVERPAYMENT')
A																	EDTCDE(1)
A	55																DSPATR(BL)
A	N56																DSPATR(ND)
A*											STSCDE	1A	H				TEXT('STATUS CODE')

Figure 73 (Part 2 of 3). Example of a Data Description Specification for a Payment Update Program


```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7..IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                02/01/89
 2 000200 PROGRAM-ID. PAYUPDT.                                    03/22/89
 3 000300 ENVIRONMENT DIVISION.                                  02/01/89
 4 000400 CONFIGURATION SECTION.                                02/01/89
 5 000500 SOURCE-COMPUTER. IBM-AS400.                            02/02/89
 6 000600 OBJECT-COMPUTER. IBM-AS400.                            02/02/89
 7 000700 INPUT-OUTPUT SECTION.                                  02/01/89
 8 000800 FILE-CONTROL.                                          02/01/89
 9 000900     SELECT CUSTOMER-INVOICE-FILE                       02/01/89
10 001000     ASSIGN TO DATABASE-ORDHDRL                         02/01/89
11 001100     ORGANIZATION IS INDEXED                            02/01/89
12 001200     ACCESS MODE IS SEQUENTIAL                         02/01/89
13 001300     RECORD KEY IS COMP-KEY                             02/01/89
14 001400     FILE STATUS IS STATUS-CODE-ONE.                    02/01/89
15 001500     SELECT CUSTOMER-MASTER-FILE                       02/01/89
16 001600     ASSIGN TO DATABASE-CUSMSTP                         02/01/89
17 001700     ORGANIZATION IS INDEXED                            02/01/89
18 001800     ACCESS IS RANDOM                                   02/01/89
19 001900     RECORD KEY IS CUST OF CUSTOMER-MASTER-RECORD.    02/01/89
20 002000     SELECT PAYMENT-UPDATE-DISPLAY-FILE                02/01/89
21 002100     ASSIGN TO WORKSTATION-PAYUPDT                      03/22/89
22 002200     ORGANIZATION IS TRANSACTION                       02/01/89
23 002300     ACCESS IS DYNAMIC                                  02/01/89
24 002400     RELATIVE KEY IS REL-NUMBER                         02/01/89
25 002500     FILE STATUS IS STATUS-CODE-ONE                    02/01/89
26 002600     CONTROL-AREA IS WS-CONTROL.                        02/01/89
   002700
27 002800 DATA DIVISION.                                        02/01/89
28 002900 FILE SECTION.                                          02/01/89
29 003000 FD CUSTOMER-INVOICE-FILE                              02/01/89
30 003100 LABEL RECORDS ARE STANDARD.                            02/01/89
31 003200 01 CUSTOMER-INVOICE-RECORD.                           02/01/89
32 003300 COPY DDS-ORDHDR OF ORDHDRL.                            02/01/89
+000001* I-O FORMAT:ORDHDR FROM FILE ORDHDRL OF LIBRARY XPLIB  03/14/89
+000002*
+000003*THE KEY DEFINITIONS FOR RECORD FORMAT ORDHDR           ORDHDR
+000004* NUMBER NAME RETRIEVAL TYPE ALTSEQ                   ORDHDR
+000005* 0001 CUST ASCENDING AN NO ORDHDR
+000006* 0002 INVNUM ASCENDING SIGNED NO ORDHDR
33 +000007 05 ORDHDR. ORDHDR
34 +000008 06 CUST PIC X(5). ORDHDR
+000009* CUSTOMER NUMBER ORDHDR
35 +000010 06 INVNUM PIC S9(5) COMP-3. ORDHDR
+000011* INVOICE NUMBER ORDHDR
36 +000012 06 ORDERN PIC S9(5) COMP-3. ORDHDR
+000013* ORDER NUMBER ORDHDR
37 +000014 06 ORDDAT PIC S9(6) COMP-3. ORDHDR
+000015* DATE ORDER ENTERED ORDHDR
38 +000016 06 CUSORD PIC X(15). ORDHDR
+000017* CUSTOMER PURCHASE ORDER NUMBER ORDHDR
39 +000018 06 SHPVIA PIC X(15). ORDHDR
+000019* SHIPPING INSTRUCTIONS ORDHDR
40 +000020 06 ORDSTS PIC S9(1) COMP-3. ORDHDR
+000021* ORDER STATUS 1PCS 2CNT 3CHK 4RDY 5PRT 6PC ORDHDR
41 +000022 06 OPRNAM PIC X(10). ORDHDR
+000023* OPERATOR WHO ENTERED ORD ORDHDR
42 +000024 06 ORDAMT PIC S9(6)V9(2) COMP-3. ORDHDR
+000025* DOLLAR AMOUNT OF ORDER ORDHDR
43 +000026 06 CUSTYP PIC S9(1) COMP-3. ORDHDR
+000027* CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT ORDHDR
44 +000028 06 PRTDAT PIC S9(6) COMP-3. ORDHDR
+000029* DATE ORDER WAS PRINTED ORDHDR
45 +000030 06 OPNSTS PIC S9(1) COMP-3. ORDHDR
+000031* ORDER OPEN STATUS 1=OPEN 2= CLOSE 3=CANCEL ORDHDR
46 +000032 06 TOTLIN PIC S9(3) COMP-3. ORDHDR
+000033* TOTAL LINE ITEMS IN ORDER ORDHDR
47 +000034 06 ACTMTH PIC S9(2) COMP-3. ORDHDR
+000035* ACCOUNTING MONTH OF SALE ORDHDR
48 +000036 06 ACTYR PIC S9(2) COMP-3. ORDHDR
+000037* ACCOUNTING YEAR OF SALE ORDHDR
49 +000038 06 STATE PIC X(2). ORDHDR
+000039* STATE ORDHDR
50 +000040 06 AMPAID PIC S9(6)V9(2) COMP-3. ORDHDR
+000041* AMOUNT PAID ORDHDR
51 003400 66 COMP-KEY RENAMES CUST THRU INVNUM.
   003500

```

Figure 74 (Part 1 of 8). Source Listing of a Payment Update Program Example

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S COPYNAME  CHG DATE
52 003600 FD CUSTOMER-MASTER-FILE
53 003700 LABEL RECORDS ARE STANDARD.
54 003800 01 CUSTOMER-MASTER-RECORD.
55 003900 COPY DDS-CUSMST OF CUSMSTP.
+000001* I-O FORMAT:CUSMST FROM FILE CUSMSTP OF LIBRARY XMPLIB CUSMST
+000002* CUSTOMER MASTER RECORD CUSMST
+000003*THE KEY DEFINITIONS FOR RECORD FORMAT CUSMST CUSMST
+000004* NUMBER NAME RETRIEVAL TYPE ALTSEQ CUSMST
+000005* 0001 CUST ASCENDING AN NO CUSMST
56 +000006 05 CUSMST. CUSMST
57 +000007 06 CUST PIC X(5). CUSMST
+000008* CUSTOMER NUMBER CUSMST
58 +000009 06 NAME PIC X(25). CUSMST
+000010* CUSTOMER NAME CUSMST
59 +000011 06 ADDR PIC X(20). CUSMST
+000012* CUSTOMER ADDRESS CUSMST
60 +000013 06 CITY PIC X(20). CUSMST
+000014* CUSTOMER CITY CUSMST
61 +000015 06 STATE PIC X(2). CUSMST
+000016* STATE CUSMST
62 +000017 06 ZIP PIC S9(5) COMP-3. CUSMST
+000018* ZIP CODE CUSMST
63 +000019 06 SRHCOD PIC X(6). CUSMST
+000020* CUSTOMER NUMBER SEARCH CODE CUSMST
64 +000021 06 CUSTYP PIC S9(1) COMP-3. CUSMST
+000022* CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT CUSMST
65 +000023 06 ARBAL PIC S9(6)V9(2) COMP-3. CUSMST
+000024* ACCOUNTS REC. BALANCE CUSMST
66 +000025 06 ORDBAL PIC S9(6)V9(2) COMP-3. CUSMST
+000026* A/R AMT. IN ORDER FILE CUSMST
67 +000027 06 LSTAMT PIC S9(6)V9(2) COMP-3. CUSMST
+000028* LAST AMT. PAID IN A/R CUSMST
68 +000029 06 LSTDAT PIC S9(6) COMP-3. CUSMST
+000030* LAST DATE PAID IN A/R CUSMST
69 +000031 06 CRDLMT PIC S9(6)V9(2) COMP-3. CUSMST
+000032* CUSTOMER CREDIT LIMIT CUSMST
70 +000033 06 SLSYR PIC S9(8)V9(2) COMP-3. CUSMST
+000034* CUSTOMER SALES THIS YEAR CUSMST
71 +000035 06 SLSLYR PIC S9(8)V9(2) COMP-3. CUSMST
+000036* CUSTOMER SALES LAST YEAR CUSMST
004000
72 004100 FD PAYMENT-UPDATE-DISPLAY-FILE
73 004200 LABEL RECORDS ARE OMITTED.
74 004300 01 PAYMENT-UPDATE-DISPLAY-RECORD.
75 004400 COPY DDS-ALL-FORMATS OF PAYUPDTP.
76 +000001 05 PAYUPDTP-RECORD PIC X(59). <-ALL-FMTS
+000002* INPUT FORMAT:SUBFILE1 FROM FILE PAYUPDTP OF LIBRARY XMPLIB <-ALL-FMTS
+000003* SUBFILE FOR CUSTOMER PAYMENT <-ALL-FMTS
77 +000004 05 SUBFILE1-I REDEFINES PAYUPDTP-RECORD. <-ALL-FMTS
78 +000005 06 ACPPTM PIC X(4). <-ALL-FMTS
+000006* ACCEPT PAYMENT <-ALL-FMTS
79 +000007 06 CUST PIC X(5). <-ALL-FMTS
+000008* CUSTOMER NUMBER <-ALL-FMTS
80 +000009 06 AMPAID PIC S9(6)V9(2). <-ALL-FMTS
+000010* AMOUNT PAID <-ALL-FMTS
81 +000011 06 ECPMSG PIC X(31). <-ALL-FMTS
+000012* EXCEPTION MESSAGE <-ALL-FMTS
82 +000013 06 OVRPMT PIC S9(6)V9(2). <-ALL-FMTS
+000014* OVERPAYMENT <-ALL-FMTS
83 +000015 06 STSCDE PIC X(1). <-ALL-FMTS
+000016* STATUS CODE <-ALL-FMTS
+000017* OUTPUT FORMAT:SUBFILE1 FROM FILE PAYUPDTP OF LIBRARY XMPLIB <-ALL-FMTS
+000018* SUBFILE FOR CUSTOMER PAYMENT <-ALL-FMTS
84 +000019 05 SUBFILE1-0 REDEFINES PAYUPDTP-RECORD. <-ALL-FMTS
85 +000020 06 SUBFILE1-0-INDIC. <-ALL-FMTS
86 +000021 07 IN51 PIC 1 INDIC 51. <-ALL-FMTS
87 +000022 07 IN52 PIC 1 INDIC 52. <-ALL-FMTS
88 +000023 07 IN53 PIC 1 INDIC 53. <-ALL-FMTS
89 +000024 07 IN54 PIC 1 INDIC 54. <-ALL-FMTS
90 +000025 07 IN55 PIC 1 INDIC 55. <-ALL-FMTS
91 +000026 07 IN56 PIC 1 INDIC 56. <-ALL-FMTS
92 +000027 06 CUST PIC X(5). <-ALL-FMTS
+000028* CUSTOMER NUMBER <-ALL-FMTS
93 +000029 06 AMPAID PIC S9(6)V9(2). <-ALL-FMTS
+000030* AMOUNT PAID <-ALL-FMTS
94 +000031 06 ECPMSG PIC X(31). <-ALL-FMTS
+000032* EXCEPTION MESSAGE <-ALL-FMTS

```

Figure 74 (Part 2 of 8). Source Listing of a Payment Update Program Example

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
95 +000033          06 OVRPMT          PIC S9(6)V9(2).          <-ALL-FMTS
+000034*          OVERPAYMENT          <-ALL-FMTS
96 +000035          06 STSCDE          PIC X(1).          <-ALL-FMTS
+000036*          STATUS CODE          <-ALL-FMTS
+000037* INPUT FORMAT:CONTROL1 FROM FILE PAYUPDTD OF LIBRARY XMLPLIB <-ALL-FMTS
+000038*          SUBFILE CONTROL          <-ALL-FMTS
97 +000039          05 CONTROL1-I REDEFINES PAYUPDTD-RECORD. <-ALL-FMTS
98 +000040          06 CONTROL1-I-INDIC. <-ALL-FMTS
99 +000041          07 IN99          PIC 1 INDIC 99. <-ALL-FMTS
+000042*          HELP KEY          <-ALL-FMTS
100 +000043          07 IN98          PIC 1 INDIC 98. <-ALL-FMTS
+000044*          END PAYMENT UPDATE <-ALL-FMTS
101 +000045          07 IN97          PIC 1 INDIC 97. <-ALL-FMTS
+000046*          IGNORE INPUT          <-ALL-FMTS
+000047* OUTPUT FORMAT:CONTROL1 FROM FILE PAYUPDTD OF LIBRARY XMLPLIB <-ALL-FMTS
+000048*          SUBFILE CONTROL          <-ALL-FMTS
102 +000049          05 CONTROL1-0 REDEFINES PAYUPDTD-RECORD. <-ALL-FMTS
103 +000050          06 CONTROL1-0-INDIC. <-ALL-FMTS
104 +000051          07 IN61          PIC 1 INDIC 61. <-ALL-FMTS
105 +000052          07 IN62          PIC 1 INDIC 62. <-ALL-FMTS
106 +000053          07 IN99          PIC 1 INDIC 99. <-ALL-FMTS
+000054*          HELP KEY          <-ALL-FMTS
107 +000055          07 IN63          PIC 1 INDIC 63. <-ALL-FMTS
108 +000056          07 IN64          PIC 1 INDIC 64. <-ALL-FMTS
+000057* INPUT FORMAT:MESSAGE1 FROM FILE PAYUPDTD OF LIBRARY XMLPLIB <-ALL-FMTS
+000058*          MESSAGE RECORD          <-ALL-FMTS
+000059*          05 MESSAGE1-I REDEFINES PAYUPDTD-RECORD. <-ALL-FMTS
+000060* OUTPUT FORMAT:MESSAGE1 FROM FILE PAYUPDTD OF LIBRARY XMLPLIB <-ALL-FMTS
+000061*          MESSAGE RECORD          <-ALL-FMTS
109 +000062          05 MESSAGE1-0 REDEFINES PAYUPDTD-RECORD. <-ALL-FMTS
110 +000063          06 MESSAGE1-0-INDIC. <-ALL-FMTS
111 +000064          07 IN71          PIC 1 INDIC 71. <-ALL-FMTS
004500
112 004600 WORKING-STORAGE SECTION.
004700
113 004800 01 REL-NUMBER          PIC 9(05)
114 004900          VALUE ZEROS.
005000
115 005100 01 WS-CONTROL.
116 005200 05 WS-IND          PIC X(02).
117 005300 05 WS-FORMAT          PIC X(10).
118 005400 01 SYSTEM-DATE.
119 005500 05 SYSTEM-YEAR          PIC 99.
120 005600 05 SYSTEM-MONTH          PIC 99.
121 005700 05 SYSTEM-DAY          PIC 99.
122 005800 01 PROGRAM-DATE.
123 005900 05 PROGRAM-MONTH          PIC 99.
124 006000 05 PROGRAM-DAY          PIC 99.
125 006100 05 PROGRAM-YEAR          PIC 99.
126 006200 01 FILE-DATE REDEFINES PROGRAM-DATE
127 006300          PIC S9(6).
128 006400 01 EXCEPTION-STATUS.
129 006500 05 STATUS-CODE-ONE          PIC XX.
130 006600 88 SUBFILE-IS-FULL          VALUE '9M'.
131 006700 01 EXCEPTION-MESSAGES.
132 006800 05 MESSAGE-ONE          PIC X(31)
133 006900          VALUE 'CUSTOMER DOES NOT EXIST '.
134 007000 05 MESSAGE-TWO          PIC X(31)
135 007100          VALUE 'NO INVOICES EXIST FOR CUSTOMER '.
136 007200 05 MESSAGE-THREE          PIC X(31)
137 007300          VALUE 'CUSTOMER HAS AN OVER PAYMENT OF '.
138 007400 01 PROGRAM-VARIABLES.
139 007500 05 AMOUNT-OWED          PIC S9(6)V99.
140 007600 05 AMOUNT-PAID          PIC S9(6)V99.
141 007700 05 INVOICE-BALANCE          PIC S9(6)V99.
142 007800 01 ERRPGM-PARAMETERS.
143 007900 05 DISPLAY-PARAMETER          PIC X(8)
144 008000          VALUE 'PAYUPDTD'.
145 008100 05 DUMMY-ONE          PIC X(6)
146 008200          VALUE SPACES.
147 008300 05 DUMMY-TWO          PIC X(6)
148 008400          VALUE SPACES.
149 008500 05 STATUS-CODE-TWO.
150 008600 10 PRIMARY          PIC X(1).
151 008700 10 SECONDARY          PIC X(1).

```

Figure 74 (Part 3 of 8). Source Listing of a Payment Update Program Example

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B.+.2.+.3.+.4.+.5.+.6.+.7..IDENTFCN S COPYNAME  CHG DATE
152 008800      10 FILLER                PIC X(5)
153 008900                VALUE SPACES.
154 009000      05 DUMMY-THREE          PIC X(10)
155 009100                VALUE SPACES.
    009200
156 009300      01 SWITCH-AREA.
157 009400      05 SW01                  PIC 1.
158 009500      88 WRITE-DISPLAY        VALUE B'1'.
159 009600      88 READ-DISPLAY        VALUE B'0'.
160 009700      05 SW02                  PIC 1.
161 009800      88 SUBFILE1-FORMAT      VALUE B'1'.
162 009900      88 NOT-SUBFILE1-FORMAT  VALUE B'0'.
163 010000      05 SW03                  PIC 1.
164 010100      88 CONTROL1-FORMAT      VALUE B'1'.
165 010200      88 NOT-CONTROL1-FORMAT  VALUE B'1'.
166 010300      05 SW04                  PIC 1.
167 010400      88 NO-MORE-TRANSACTIONS-EXIST VALUE B'1'.
168 010500      88 TRANSACTIONS-EXIST   VALUE B'0'.
169 010600      05 SW05                  PIC 1.
170 010700      88 CUSTOMER-NOT-FOUND   VALUE B'1'.
171 010800      88 CUSTOMER-EXIST      VALUE B'0'.
172 010900      05 SW06                  PIC 1.
173 011000      88 NO-MORE-INVOICES-EXIST VALUE B'1'.
174 011100      88 CUSTOMER-INVOICE-EXIST VALUE B'0'.
175 011200      05 SW07                  PIC 1.
176 011300      88 NO-MORE-PAYMENT-EXIST VALUE B'0'.
177 011400      88 PAYMENT-EXIST        VALUE B'0'.
178 011500      05 SW08                  PIC 1.
179 011600      88 INPUT-ERRORS-EXIST   VALUE B'1'.
180 011700      88 NO-INPUT-ERRORS-EXIST VALUE B'0'.
181 011800      05 SW09                  PIC 1.
182 011900      88 OVER-PAYMENT-DISPLAYED-ONCE VALUE B'1'.
183 012000      88 OVER-PAYMENT-NOT-DISPLAYED VALUE B'0'.
    012100
184 012200      01 INDICATOR-AREA.
185 012300      05 IN99                  PIC 1 INDIC 99.
186 012400      88 HELP-IS-NEEDED      VALUE B'1'.
187 012500      88 HELP-IS-NOT-NEEDED   VALUE B'0'.
188 012600      05 IN98                  PIC 1 INDIC 98.
189 012700      88 END-OF-PAYMENT-UPDATE VALUE B'1'.
190 012800      05 IN97                  PIC 1 INDIC 97.
191 012900      88 IGNORE-INPUT        VALUE B'1'.
192 013000      05 IN51                  PIC 1 INDIC 51.
193 013100      88 DISPLAY-ACCEPT-PAYMENT VALUE B'1'.
194 013200      88 DO-NOT-DISPLAY-ACCEPT-PAYMENT VALUE B'0'.
195 013300      05 IN52                  PIC 1 INDIC 52.
196 013400      88 REVERSE-FIELD-IMAGE   VALUE B'1'.
197 013500      88 DO-NOT-REVERSE-FIELD-IMAGE VALUE B'0'.
198 013600      05 IN53                  PIC 1 INDIC 53.
199 013700      88 DO-NOT-DISPLAY-FIELD  VALUE B'1'.
200 013800      88 DISPLAY-FIELD        VALUE B'0'.
201 013900      05 IN54                  PIC 1 INDIC 54.
202 014000      88 PROTECT-INPUT-FIELD   VALUE B'1'.
203 014100      88 DO-NOT-PROTECT-INPUT-FIELD VALUE B'0'.
204 014200      05 IN55                  PIC 1 INDIC 55.
205 014300      88 MAKE-FIELD-BLINK     VALUE B'1'.
206 014400      88 DO-NOT-MAKE-FIELD-BLINK VALUE B'0'.
207 014500      05 IN56                  PIC 1 INDIC 56.
208 014600      88 DISPLAY-OVER-PAYMENT  VALUE B'1'.
209 014700      88 DO-NOT-DISPLAY-OVER-PAYMENT VALUE B'0'.
210 014800      05 IN61                  PIC 1 INDIC 61.
211 014900      88 CLEAR-SUBFILE        VALUE B'1'.
212 015000      88 DO-NOT-CLEAR-SUBFILE   VALUE B'0'.
213 015100      05 IN62                  PIC 1 INDIC 62.
214 015200      88 DISPLAY-SCREEN       VALUE B'1'.
215 015300      88 DO-NOT-DISPLAY-SCREEN  VALUE B'0'.
216 015400      05 IN63                  PIC 1 INDIC 63.
217 015500      88 DISPLAY-ACCEPT-HEADING VALUE B'1'.
218 015600      88 DO-NOT-DISPLAY-ACCEPT-HEADING VALUE B'0'.
219 015700      05 IN64                  PIC 1 INDIC 64.
220 015800      88 DISPLAY-EXCEPTION    VALUE B'1'.
221 015900      88 DO-NOT-DISPLAY-EXCEPTION VALUE B'0'.
222 016000      05 IN71                  PIC 1 INDIC 71.
223 016100      88 DISPLAY-ACCEPT-MESSAGE VALUE B'1'.
224 016200      88 DO-NOT-DISPLAY-ACCEPT-MESSAGE VALUE B'0'.
    016300

```

Figure 74 (Part 4 of 8). Source Listing of a Payment Update Program Example


```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B.+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S COPYNAME CHG DATE
225 016400 PROCEDURE DIVISION.
      016500
      016600 DECLARATIVES.
      016700
      016800 TRANSACTION-ERROR SECTION.
      016900     USE AFTER STANDARD ERROR PROCEDURE
      017000     PAYMENT-UPDATE-DISPLAY-FILE.
      017100 WORK-STATION-ERROR-HANDLER.
226 017200     IF SUBFILE-IS-FULL THEN
      017300     NEXT SENTENCE
      017400     ELSE
227 017500     DISPLAY 'ERROR IN PAYMENT-UPDATE' STATUS-CODE-ONE.
      017600 END DECLARATIVES.
      017700
      017800 CUSTOMER-PAYMENT-UPDATE SECTION.
      017900 MAINLINE-ROUTINE.
228 018000     PERFORM SET-UP-ROUTINE.
229 018100     PERFORM PROCESS-TRANSACTION-FILE
      018200     UNTIL END-OF-PAYMENT-UPDATE.
230 018300     PERFORM CLEAN-UP-ROUTINE.
      018400
      018500 SET-UP-ROUTINE.
231 018600     OPEN I-0     CUSTOMER-INVOICE-FILE
      018700     CUSTOMER-MASTER-FILE
      018800     PAYMENT-UPDATE-DISPLAY-FILE.
232 018900     MOVE ALL B'0' TO INDICATOR-AREA
      019000     SWITCH-AREA.
233 019100     ACCEPT SYSTEM-DATE FROM DATE.
234 019200     MOVE SYSTEM-YEAR TO PROGRAM-YEAR.
235 019300     MOVE SYSTEM-MONTH TO PROGRAM-MONTH.
236 019400     MOVE SYSTEM-DATE TO PROGRAM-DAY.
237 019500     SET WRITE-DISPLAY
      019600     CONTROL1-FORMAT
      019700     DO-NOT-DISPLAY-OVER-PAYMENT
      019800     DO-NOT-PROTECT-INPUT-FIELD
      019900     DO-NOT-REVERSE-FIELD-IMAGE
      020000     DO-NOT-MAKE-FIELD-BLINK
      020100     CLEAR-SUBFILE TO TRUE.
238 020200     MOVE CORR INDICATOR-AREA TO CONTROL1-0-INDIC.
239 020300     WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      020400     FORMAT IS 'CONTROL1'.
240 020500     SET DO-NOT-CLEAR-SUBFILE TO TRUE.
241 020600     PERFORM INITIALIZE-SUBFILE-RECORD 17 TIMES.
242 020700     SET DISPLAY-SCREEN TO TRUE.
243 020800     MOVE CORR INDICATOR-AREA TO CONTROL1-0-INDIC.
244 020900     WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      021000     FORMAT IS 'CONTROL1'.
245 021100     READ PAYMENT-UPDATE-DISPLAY-FILE RECORD
      021200     FORMAT IS 'CONTROL1'.
246 021300     MOVE CORR CONTROL1-I-INDIC TO INDICATOR-AREA.
      021400 PROCESS-TRANSACTION-FILE.
247 021500     IF HELP-IS-NOT-NEEDED THEN
248 021600     IF IGNORE-INPUT THEN
249 021700     SET WRITE-DISPLAY
      021800     CONTROL1-FORMAT
      021900     CLEAR-SUBFILE
      022000     DISPLAY-FIELD
      022100     DO-NOT-DISPLAY-OVER-PAYMENT
      022200     DO-NOT-PROTECT-INPUT-FIELD
      022300     DO-NOT-REVERSE-FIELD-IMAGE
      022400     DO-NOT-DISPLAY-ACCEPT-PAYMENT
      022500     DO-NOT-DISPLAY-ACCEPT-HEADING
      022600     DO-NOT-DISPLAY-ACCEPT-MESSAGE
      022700     DO-NOT-MAKE-FIELD-BLINK TO TRUE
250 022800     MOVE CORR INDICATOR-AREA TO CONTROL1-0-INDIC
251 022900     WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      023000     FORMAT IS 'CONTROL1'
252 023100     SET DO-NOT-CLEAR-SUBFILE TO TRUE
253 023200     MOVE 0 TO REL-NUMBER
254 023300     PERFORM INITIALIZE-SUBFILE-RECORD 17 TIMES
      023400     ELSE
255 023500     SET TRANSACTIONS-EXIST
      023600     DO-NOT-DISPLAY-ACCEPT-HEADING
      023700     DO-NOT-DISPLAY-ACCEPT-MESSAGE
      023800     DO-NOT-DISPLAY-EXCEPTION TO TRUE
256 023900     PERFORM READ-MODIFIED-SUBFILE-RECORD

```

Figure 74 (Part 5 of 8). Source Listing of a Payment Update Program Example

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7...IDENTFCN S COPYNAME  CHG DATE
257 024000          PERFORM TRANSACTION-VALIDATION
      024100          UNTIL NO-MORE-TRANSACTIONS-EXIST
258 024200          SET NO-INPUT-ERRORS-EXIST TO TRUE
259 024300          PERFORM TEST-FOR-RECORD-INPUT-ERRORS
      024400          VARYING REL-NUMBER
      024500          FROM 1
      024600          BY 1
      024700          UNTIL REL-NUMBER IS GREATER THAN 17
      024800          OR INPUT-ERRORS-EXIST
260 024900          IF NO-INPUT-ERRORS-EXIST THEN
261 025000          IF OVER-PAYMENT-DISPLAYED-ONCE THEN
262 025100          SET WRITE-DISPLAY
      025200          CONTROL1-FORMAT
      025300          DO-NOT-DISPLAY-OVER-PAYMENT
      025400          DO-NOT-PROTECT-INPUT-FIELD
      025500          DO-NOT-REVERSE-FIELD-IMAGE
      025600          DO-NOT-MAKE-FIELD-BLINK
      025700          DO-NOT-DISPLAY-ACCEPT-PAYMENT
      025800          DO-NOT-DISPLAY-ACCEPT-HEADING
      025900          DO-NOT-DISPLAY-ACCEPT-MESSAGE
      026000          DO-NOT-DISPLAY-EXCEPTION
      026100          CLEAR-SUBFILE
      026200          DISPLAY-FIELD
      026300          TO TRUE
263 026400          MOVE CORR INDICATOR-AREA TO CONTROL1-0-INDIC
264 026500          WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      026600          FORMAT IS 'CONTROL1'
265 026700          SET DO-NOT-CLEAR-SUBFILE TO TRUE
266 026800          MOVE 0 TO REL-NUMBER
267 026900          PERFORM INITIALIZE-SUBFILE-RECORD 17 TIMES
      027000          ELSE
268 027100          SET OVER-PAYMENT-DISPLAYED-ONCE TO TRUE
      027200          ELSE
      027300          NEXT SENTENCE
      027400          ELSE
      027500          NEXT SENTENCE.
269 027600          SET WRITE-DISPLAY, DISPLAY-SCREEN TO TRUE.
270 027700          MOVE CORR INDICATOR-AREA TO MESSAGE1-0-INDIC.
271 027800          WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      027900          FORMAT IS 'MESSAGE1'.
272 028000          SET WRITE-DISPLAY, CONTROL1-FORMAT TO TRUE.
273 028100          MOVE CORR INDICATOR-AREA TO CONTROL1-0-INDIC.
274 028200          WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      028300          FORMAT IS 'CONTROL1'.
275 028400          READ PAYMENT-UPDATE-DISPLAY-FILE RECORD
      028500          FORMAT IS 'CONTROL1'.
276 028600          MOVE CORR CONTROL1-I-INDIC TO INDICATOR-AREA.
      028700          READ-MODIFIED-SUBFILE-RECORD.
277 028800          READ SUBFILE PAYMENT-UPDATE-DISPLAY-FILE
      028900          NEXT MODIFIED RECORD FORMAT IS 'SUBFILE1'
278 029000          AT END SET NO-MORE-TRANSACTIONS-EXIST TO TRUE.
      029100          TEST-FOR-RECORD-INPUT-ERRORS.
279 029200          READ SUBFILE PAYMENT-UPDATE-DISPLAY-FILE RECORD
      029300          FORMAT IS 'SUBFILE1'.
280 029400          IF STSCDE OF SUBFILE1-I IS EQUAL TO '1' THEN
281 029500          SET INPUT-ERRORS-EXIST TO TRUE
      029600          ELSE
      029700          NEXT SENTENCE.
      029800          TRANSACTION-VALIDATION.
282 029900          MOVE CUST OF SUBFILE1-I OF PAYMENT-UPDATE-DISPLAY-RECORD
      030000          TO CUST OF CUSTOMER-MASTER-RECORD.
283 030100          SET CUSTOMER-EXIST TO TRUE.
284 030200          READ CUSTOMER-MASTER-FILE
285 030300          INVALID KEY SET CUSTOMER-NOT-FOUND TO TRUE.
286 030400          IF CUSTOMER-EXIST THEN
287 030500          MOVE CUST OF CUSMST TO CUST OF ORDHDR
288 030600          MOVE ZEROES TO INVNUM
289 030700          SET CUSTOMER-INVOICE-EXIST TO TRUE
290 030800          PERFORM START-ON-CUSTOMER-INVOICE-FILE
291 030900          IF CUSTOMER-INVOICE-EXIST THEN
292 031000          PERFORM READ-CUSTOMER-INVOICE-RECORD
293 031100          IF CUSTOMER-INVOICE-EXIST THEN
294 031200          PERFORM CUSTOMER-MASTER-FILE-UPDATE
295 031300          MOVE AMPAID OF SUBFILE1-I TO AMOUNT-PAID
296 031400          SET PAYMENT-EXIST TO TRUE

```

Figure 74 (Part 6 of 8). Source Listing of a Payment Update Program Example

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B.+. . . . . 2. . . . . 3. . . . . 4. . . . . 5. . . . . 6. . . . . 7. IDENTFCN S COPYNAME  CHG DATE
297 031500          PERFORM PAYMENT-UPDATE
      031600          UNTIL NO-MORE-INVOICES-EXIST
      031700          OR NO-MORE-PAYMENT-EXIST
298 031800          IF ARBAL OF CUSTOMER-MASTER-RECORD IS NEGATIVE
299 031900          SET MAKE-FIELD-BLINK
      032000          DISPLAY-FIELD
      032100          DO-NOT-REVERSE-FIELD-IMAGE
      032200          OVER-PAYMENT-NOT-DISPLAYED
      032300          DISPLAY-OVER-PAYMENT
      032400          DISPLAY-EXCEPTION
      032500          DO-NOT-DISPLAY-ACCEPT-PAYMENT
      032600          PROTECT-INPUT-FIELD TO TRUE
300 032700          MOVE ARBAL TO OVRPMT OF SUBFILE1-0
301 032800          MOVE MESSAGE-THREE TO ECPMSG OF SUBFILE1-0
302 032900          MOVE '0' TO STSCDE OF SUBFILE1-0
303 033000          PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
      033100          ELSE
304 033200          SET DO-NOT-DISPLAY-FIELD
      033300          DO-NOT-DISPLAY-OVER-PAYMENT
      033400          DO-NOT-REVERSE-FIELD-IMAGE
      033500          DO-NOT-MAKE-FIELD-BLINK
      033600          DO-NOT-DISPLAY-ACCEPT-PAYMENT
      033700          PROTECT-INPUT-FIELD TO TRUE
305 033800          MOVE SPACES TO ECPMSG OF SUBFILE1-0
306 033900          MOVE ZEROES TO OVRPMT OF SUBFILE1-0
307 034000          MOVE '0' TO STSCDE OF SUBFILE1-0
308 034100          PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
      034200          ELSE
309 034300          PERFORM NO-CUSTOMER-INVOICE-ROUTINE
      034400          ELSE
310 034500          PERFORM NO-CUSTOMER-INVOICE-ROUTINE
      034600          ELSE
311 034700          SET REVERSE-FIELD-IMAGE
      034800          DO-NOT-PROTECT-INPUT-FIELD
      034900          DISPLAY-FIELD
      035000          DO-NOT-DISPLAY-OVER-PAYMENT
      035100          DO-NOT-MAKE-FIELD-BLINK
      035200          DISPLAY-EXCEPTION
      035300          DO-NOT-DISPLAY-ACCEPT-PAYMENT
      035400          DO-NOT-PROTECT-INPUT-FIELD TO TRUE
312 035500          MOVE ZEROES TO OVRPMT OF SUBFILE1-0
313 035600          MOVE MESSAGE-ONE TO ECPMSG OF SUBFILE1-0
314 035700          MOVE '1' TO STSCDE OF SUBFILE1-0
315 035800          PERFORM REWRITE-DISPLAY-SUBFILE-RECORD.
316 035900          PERFORM READ-MODIFIED-SUBFILE-RECORD.
      036000          START-ON-CUSTOMER-INVOICE-FILE.
317 036100          START CUSTOMER-INVOICE-FILE
      036200          KEY IS GREATER THAN COMP-KEY
318 036300          INVALID KEY SET NO-MORE-INVOICES-EXIST TO TRUE.
      036400          READ-CUSTOMER-INVOICE-RECORD.
319 036500          READ CUSTOMER-INVOICE-FILE NEXT RECORD
320 036600          AT END SET NO-MORE-INVOICES-EXIST TO TRUE.
321 036700          IF CUST OF CUSTOMER-MASTER-RECORD
      036800          IS NOT EQUAL TO CUST OF CUSTOMER-INVOICE-RECORD THEN
322 036900          SET NO-MORE-INVOICES-EXIST TO TRUE
      037000          ELSE
323 037100          NEXT SENTENCE.
      037200          CUSTOMER-MASTER-FILE-UPDATE.
324 037300          MOVE FILE-DATE TO LSTDAT OF CUSTOMER-MASTER-RECORD.
      037400          MOVE AMPAID OF SUBFILE1-I
      037500          TO LSTAMT OF CUSTOMER-MASTER-RECORD.
325 037600          SUBTRACT AMPAID OF SUBFILE1-I
      037700          FROM ARBAL OF CUSTOMER-MASTER-RECORD.
326 037800          REWRITE CUSTOMER-MASTER-RECORD.
      037900          REWRITE-DISPLAY-SUBFILE-RECORD.
327 038000          MOVE AMPAID OF SUBFILE1-I TO AMPAID OF SUBFILE1-0.
328 038100          MOVE CUST OF SUBFILE1-I TO CUST OF SUBFILE1-0.
329 038200          SET WRITE-DISPLAY TO TRUE.
330 038300          SET SUBFILE1-FORMAT TO TRUE.
331 038400          MOVE CORR INDICATOR-AREA TO SUBFILE1-0-INDIC.
332 038500          REWRITE SUBFILE PAYMENT-UPDATE-DISPLAY-RECORD
      038600          FORMAT IS 'SUBFILE1'.
      038700          NO-CUSTOMER-INVOICE-ROUTINE.
333 038800          IF STSCDE OF SUBFILE1-I IS EQUAL TO '1' THEN
334 038900          IF ACPMPT OF SUBFILE1-I IS EQUAL TO '*NO' THEN
335 039000          SET DO-NOT-DISPLAY-FIELD

```

Figure 74 (Part 7 of 8). Source Listing of a Payment Update Program Example

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7..IDENTFCN S COPYNAME  CHG DATE
      039100          DO-NOT-DISPLAY-OVER-PAYMENT
      039200          DO-NOT-REVERSE-FIELD-IMAGE
      039300          DO-NOT-MAKE-FIELD-BLINK
      039400          DO-NOT-DISPLAY-ACCEPT-PAYMENT
      039500          PROTECT-INPUT-FIELD
      039600          TO TRUE
336  039700          MOVE SPACES TO ECPMSG OF SUBFILE1-0
337  039800          MOVE ZEROES TO OVRPMT OF SUBFILE1-0
338  039900          MOVE '0' TO STSCDE OF SUBFILE1-0
339  040000          PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
      040100          ELSE
340  040200          PERFORM CUSTOMER-MASTER-FILE-UPDATE
341  040300          SET MAKE-FIELD-BLINK
      040400          DISPLAY-FIELD
      040500          DO-NOT-REVERSE-FIELD-IMAGE
      040600          OVER-PAYMENT-NOT-DISPLAYED
      040700          DISPLAY-OVER-PAYMENT
      040800          DISPLAY-EXCEPTION
      040900          DO-NOT-DISPLAY-ACCEPT-PAYMENT
      041000          PROTECT-INPUT-FIELD
      041100          TO TRUE
342  041200          MOVE ARBAL TO OVRPMT OF SUBFILE1-0
343  041300          MOVE MESSAGE-THREE TO ECPMSG OF SUBFILE1-0
344  041400          MOVE '0' TO STSCDE OF SUBFILE1-0
345  041500          PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
      041600          ELSE
346  041700          SET REVERSE-FIELD-IMAGE
      041800          DISPLAY-FIELD
      041900          DO-NOT-PROTECT-INPUT-FIELD
      042000          DO-NOT-DISPLAY-OVER-PAYMENT
      042100          DISPLAY-EXCEPTION
      042200          DISPLAY-ACCEPT-PAYMENT
      042300          DISPLAY-ACCEPT-HEADING
      042400          DISPLAY-ACCEPT-MESSAGE
      042500          DO-NOT-MAKE-FIELD-BLINK
      042600          TO TRUE
347  042700          MOVE ZEROS TO OVRPMT OF SUBFILE1-0
348  042800          MOVE MESSAGE-TWO TO ECPMSG OF SUBFILE1-0
349  042900          MOVE '1' TO STSCDE OF SUBFILE1-0
350  043000          PERFORM REWRITE-DISPLAY-SUBFILE-RECORD.
      043100          PAYMENT-UPDATE.
351  043200          SUBTRACT AMPAID OF CUSTOMER-INVOICE-RECORD
      043300          FROM ORDAMT OF CUSTOMER-INVOICE-RECORD
      043400          GIVING AMOUNT-OWED.
352  043500          SUBTRACT AMOUNT-PAID
      043600          FROM AMOUNT-OWED
      043700          GIVING INVOICE-BALANCE.
353  043800          IF INVOICE-BALANCE IS LESS THAN .01 THEN
354  043900          MOVE 2 TO OPNSTS OF CUSTOMER-INVOICE-RECORD
355  044000          MOVE ORDAMT OF CUSTOMER-INVOICE-RECORD
      044100          TO AMPAID OF CUSTOMER-INVOICE-RECORD
356  044200          SUBTRACT AMOUNT-OWED
      044300          FROM AMOUNT-PAID
      044400          ELSE
357  044500          ADD AMOUNT-PAID TO AMPAID OF CUSTOMER-INVOICE-RECORD
358  044600          SET NO-MORE-PAYMENT-EXIST TO TRUE.
359  044700          REWRITE CUSTOMER-INVOICE-RECORD.
360  044800          IF NO-MORE-PAYMENT-EXIST THEN
      044900          NEXT SENTENCE
      045000          ELSE
361  045100          PERFORM READ-CUSTOMER-INVOICE-RECORD.
      045200          INITIALIZE-SUBFILE-RECORD.
362  045300          MOVE SPACES TO CUST OF SUBFILE1-0.
363  045400          MOVE SPACES TO ECPMSG OF SUBFILE1-0.
364  045500          MOVE '0' TO STSCDE OF SUBFILE1-0.
365  045600          MOVE ZEROS TO AMPAID OF SUBFILE1-0.
366  045700          MOVE ZEROS TO OVRPMT OF SUBFILE1-0.
367  045800          ADD 1 TO REL-NUMBER.
368  045900          MOVE CORR INDICATOR-AREA TO SUBFILE1-0-INDIC.
369  046000          WRITE SUBFILE PAYMENT-UPDATE-DISPLAY-RECORD
      046100          FORMAT IS 'SUBFILE1'.
      046200          CLEAN-UP-ROUTINE.
370  046300          CLOSE CUSTOMER-INVOICE-FILE
      046400          CUSTOMER-MASTER-FILE
      046500          PAYMENT-UPDATE-DISPLAY-FILE.
371  046600          STOP RUN.
          * * * * *   E N D   O F   S O U R C E   * * * * *

```

Figure 74 (Part 8 of 8). Source Listing of a Payment Update Program Example

This is the initial display that is written to the work station to prompt you to enter the customer number and payment:

Customer Payment Update Prompt		Date 05/24/91
Customer	Payment	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	
_____	_____	

Enter the customer numbers and payments:

Customer Payment Update Prompt		Date 05/24/91
Customer	Payment	
34500	2000	
40500	30000	
36000	2500	
12500	200	
22799	4500	
41900	7500	
10001	5000	
49500	2500	
13300	3500	
56900	4000	

Payments that would result in overpayments or that have incorrect customer numbers are left on the display and appropriate messages are added:

Customer Payment Update Prompt			Date 05/24/91
Accept Payment	Customer	Payment	Exception Message
_____	40500	30000	NO INVOICES EXIST FOR CUSTOMER
_____	12500	200	NO INVOICES EXIST FOR CUSTOMER
_____	41900 10001	7500 5000	NO INVOICES EXIST FOR CUSTOMER CUSTOMER DOES NOT EXIST
_____	13300	3500	NO INVOICES EXIST FOR CUSTOMER

Accept payment values: (*NO *YES)

Indicate which payments to accept:

Customer Payment Update Prompt			Date 05/24/91
Accept Payment	Customer	Payment	Exception Message
*NO	40500	30000	NO INVOICES EXIST FOR CUSTOMER
*YES	12500	200	NO INVOICES EXIST FOR CUSTOMER
*NO	41900 10001	7500 5000	NO INVOICES EXIST FOR CUSTOMER CUSTOMER DOES NOT EXIST
*NO	13300	3500	NO INVOICES EXIST FOR CUSTOMER

Accept payment values: (*NO *YES)

The accepted payments are processed, and overpayment information is displayed:

Customer Payment Update Prompt				Date
Accept Payment	Customer	Payment	Exception Message	
	12500	200	CUSTOMER HAS AN OVERPAYMENT OF	58.50
	10001	5000	CUSTOMER DOES NOT EXIST	

End of IBM Extension

Chapter 9. Printer Files

This chapter describes how COBOL/400 interacts with the different kinds of printer files.

You can obtain printed output from a COBOL program by issuing WRITE statements to one or more printer files. Each printer file must have a unique name and be assigned to a device of PRINTER or FORMATFILE in the ASSIGN clause of that file's FILE-CONTROL entry.

A device of PRINTER must be used for program-described files, and a device of FORMATFILE must be used for externally described printer files. The Create Print File (CRTPRTF) command can be used to create a printer file (see the *CL Reference* for further information on the CRTPRTF command), or one of the IBM-supplied printer-device files, such as QPRINT can be used.

The file operations that are valid for a printer file are WRITE, OPEN, and CLOSE. For a complete description of these operations, see the *COBOL/400* Reference*.

See the *DDS Reference* for information on the DDS for externally described printer files. For more information on FORMATFILE files, see "FORMATFILE Files" on page 230.

SPECIAL-NAMES Paragraph and the ADVANCING Phrase

When the mnemonic-name associated with the function-name CSP is specified in the ADVANCING phrase of a WRITE statement for a printer file, it has the same effect as specifying ADVANCING 0 LINES.

When the mnemonic-name associated with the function-name C01 is specified in the ADVANCING phrase of a WRITE statement for a printer file, it has the same effect as specifying ADVANCING PAGE.

The ADVANCING phrase cannot be specified in WRITE statements for files assigned to FORMATFILE.

LINAGE Clause

When the LINAGE clause is specified for a file assigned to PRINTER, all spacing and paging controls are handled internally by compiler generated code.

Paper positioning is done only when the first WRITE statement is run. The paper in the printer is positioned to a new physical page, and the LINAGE-COUNTER is set to 1. When the printer file is shared and other programs have written records to the file, the COBOL WRITE statement is still considered to be the first WRITE statement. Paper positioning is handled by the COBOL/400 compiler even though it is not the first WRITE statement for that file.

All spacing and paging for WRITE statements is controlled internally. The physical size of the page is ignored when paper positioning is not properly defined for the COBOL/400 compiler. For a file that has a LINAGE clause and is assigned to PRINTER, paging consists of spacing to the end of the logical page (page body) and then spacing past the bottom and top margins.

Use of the LINAGE clause degrades performance. The LINAGE clause should be used only as necessary. If the physical paging is acceptable, the LINAGE clause is not necessary.

The LINAGE clause should not be used for files assigned to FORMATFILE.

FORMATFILE Files

Externally described printer files must be assigned to a device of FORMATFILE. The term FORMATFILE is used because the FORMAT phrase is valid in WRITE statements for the file, and the data formatting is specified in the DDS for the file.

When you have specified a device of FORMATFILE, you can obtain formatting of printed output in two ways:

1. Choose the formats to print and their order by using appropriate values in the FORMAT phrases specified for WRITE statements. For example, use one format once per page to produce a heading, and use another format to produce the detail lines on the page.
2. Choose the appropriate options to be taken when each format is printed by setting indicator values and passing these indicators through the INDICATOR phrase for the WRITE statement. For example, fields may be underlined, blank lines may be produced before or after the format is printed, or the printing of certain fields may be skipped.

The use of external descriptions for printer files has the following advantages over program descriptions:

- Multiple lines can be printed by one WRITE statement. When multiple lines are written by one WRITE statement and the END-OF-PAGE condition is reached, the END-OF-PAGE imperative statement is processed after all of the lines are printed. It is possible to print lines in the overflow area, and onto the next page before the END-OF-PAGE imperative statement is processed.

Figure 75 on page 231 shows an example of an occurrence of the END-OF-PAGE condition through FORMATFILE.

- Optional printing of fields based on indicator values is possible.
- Editing of field values is easily defined.
- Maintenance of print formats, especially those used by multiple programs, is easier.

Use of the ADVANCING phrase for FORMATFILE files causes a compilation error to be issued. Advancing of lines is controlled in a FORMATFILE file through DDS keywords, such as SKIPA and SKIPB, and through the use of line numbers.

For FORMATFILE files, the LINAGE clause is invalid.

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7..IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                02/01/89
 2 000200 PROGRAM-ID.          FRMTFILE.                          03/22/89
 3 000300 AUTHOR.              PROGRAMMER NAME.                  01/27/89
 4 000400 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.        01/27/89
 5 000500 DATE-WRITTEN. 02/02/89.                                02/04/89
 8 000800 DATE-COMPIL'D. 05/24/91 14:29:31 .                     03/01/90
 7 000700 ENVIRONMENT DIVISION.                                  01/27/89
 8 000800 CONFIGURATION SECTION.                                01/27/89
 9 000900 SOURCE-COMPUTER. IBM-AS400.                            01/27/89
10 001000 OBJECT-COMPUTER. IBM-AS400.                            01/27/89
11 001100 INPUT-OUTPUT SECTION.                                  01/27/89
12 001200 FILE-CONTROL.                                          01/27/89
13 001300     SELECT PERSREPT ASSIGN TO FORMATFILE-PERSREPT-SI 1
14 001400     ORGANIZATION IS SEQUENTIAL.                        02/04/89
15 001500     SELECT PERSFILE ASSIGN TO DATABASE-PERSFILE
16 001600     ORGANIZATION IS INDEXED                            02/04/89
17 001700     ACCESS MODE IS SEQUENTIAL                         02/04/89
18 001800     RECORD IS EXTERNALLY-DESCRIBED-KEY.              02/04/89
19 001900 DATA DIVISION.                                        01/27/89
20 002000 FILE SECTION.                                          01/27/89
21 002100 FD PERSREPT                                           02/04/89
22 002200 LABEL RECORDS ARE STANDARD.                            02/04/89
23 002300 01 PERSREPT-REC.                                       02/04/89
24 002400     COPY DDS-ALL-FORMATS-0 OF PERSREPT. 2
25 +000001     05 PERSREPT-RECORD PIC X(130).                    <-ALL-FMTS
+000002* OUTPUT FORMAT:HEADING FROM FILE PERSREPT OF LIBRARY XMPLIB <-ALL-FMTS
+000003* <-ALL-FMTS
26 +000004     05 HEADING-0 REDEFINES PERSREPT-RECORD.           <-ALL-FMTS
27 +000005     06 ORDERTYPE PIC X(15).                            <-ALL-FMTS
+000006* OUTPUT FORMAT:DETAIL FROM FILE PERSREPT OF LIBRARY XMPLIB <-ALL-FMTS
+000007* <-ALL-FMTS
28 +000008     05 DETAIL-0 REDEFINES PERSREPT-RECORD. 3
29 +000009     06 NAME PIC X(30).                                  <-ALL-FMTS
30 +000010     06 EMPLNO PIC S9(6).                               <-ALL-FMTS
31 +000011     06 BIRTHDATE PIC X(6).                             <-ALL-FMTS
32 +000012     06 ADDRESS1 PIC X(35).                             <-ALL-FMTS
33 +000013     06 MARSTAT PIC X(1).                               <-ALL-FMTS
34 +000014     06 SPOUSENAME PIC X(30).                           <-ALL-FMTS
35 +000015     06 ADDRESS2 PIC X(20).                             <-ALL-FMTS
36 +000016     06 NUMCHILD PIC S9(2).                             <-ALL-FMTS
37 002500 FD PERSFILE
38 002600 LABEL RECORDS ARE STANDARD.
39 002700 01 PERSFILE-REC.
40 002800     COPY DDS-ALL-FORMATS-0 OF PERSFILE.
41 +000001     05 PERSFILE-RECORD PIC X(130).                    <-ALL-FMTS
+000002* I-O FORMAT:PERSREC FROM FILE PERSFILE OF LIBRARY XMPLIB <-ALL-FMTS
+000003* <-ALL-FMTS
+000004*THE KEY DEFINITIONS FOR RECORD FORMAT PERSREC <-ALL-FMTS
+000005* NUMBER NAME RETRIEVAL TYPE ALTSEQ <-ALL-FMTS
+000006* 0001 EMPLNO ASCENDING SIGNED NO <-ALL-FMTS
42 +000007     05 PERSREC REDEFINES PERSFILE-RECORD.             <-ALL-FMTS
43 +000008     06 EMPLNO PIC S9(6).                               <-ALL-FMTS
44 +000009     06 NAME PIC X(30).                                  <-ALL-FMTS
45 +000010     06 ADDRESS1 PIC X(35).                             <-ALL-FMTS
46 +000011     06 ADDRESS2 PIC X(20).                             <-ALL-FMTS
47 +000012     06 BIRTHDATE PIC X(6).                             <-ALL-FMTS
48 +000013     06 MARSTAT PIC X(1).                               <-ALL-FMTS
49 +000014     06 SPOUSENAME PIC X(30).                           <-ALL-FMTS
50 +000015     06 NUMCHILD PIC S9(2).                             <-ALL-FMTS
51 002900 WORKING-STORAGE SECTION.
52 003000 77 HEAD-ORDER PIC X(15)
53 003100 VALUE "EMPLOYEE NUMBER".
54 003200 01 PERSREPT-INDICS.
55 003300     COPY DDS-ALL-FORMATS-0-INDIC OF PERSREPT. 4
56 +000001     05 PERSREPT-RECORD.                                <-ALL-FMTS
+000002* OUTPUT FORMAT:HEADING FROM FILE PERSREPT OF LIBRARY XMPLIB <-ALL-FMTS
+000003* <-ALL-FMTS
+000004*     06 HEADING-0-INDIC. <-ALL-FMTS
+000005* OUTPUT FORMAT:DETAIL FROM FILE PERSREPT OF LIBRARY XMPLIB <-ALL-FMTS
+000006* <-ALL-FMTS
57 +000007     06 DETAIL-0-INDIC. <-ALL-FMTS
58 +000008     07 IN01 PIC 1 INDIC 01. <-ALL-FMTS
003400
59 003500 77 EOF-FLAG PIC X(1)
60 003600 VALUE "0".
61 003700 88 NOT-END-OF-FILE VALUE "0".

```

Figure 75 (Part 1 of 2). Example of the END-OF-PAGE Condition

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B.+...2....3.....4.....5.....6.....7..IDENTFCN S COPYNAME  CHG DATE
62 003800 88 END-OF-FILE          VALUE "1".
63 003900 77 MARRIED              PIC X(1)
64 004000          VALUE "M".
   004100
65 004200 PROCEDURE DIVISION.
   004300 FIRST-SECT SECTION.
   004400 FIRST-PARA.
66 004500     OPEN INPUT PERSFILE
   004600     OUTPUT PERSREPT.
67 004700     PERFORM HEADING-LINE.
68 004800     PERFORM PROCESS-RECORD UNTIL END-OF-FILE.
69 004900     CLOSE PERSFILE
   005000     PERSREPT.
70 005100     STOP RUN.
   005200
   005300 PROCESS-RECORD.
71 005400     READ PERSFILE AT END SET END-OF-FILE TO TRUE.
73 005500     IF NOT-END-OF-FILE THEN
74 005600         PERFORM PRINT-RECORD. 5
   005700
   005800 PRINT-RECORD.
75 005900     MOVE CORR PERSREC TO DETAIL-0. 6
76 006000     IF MARSTAT IN PERSFILE-REC IS EQUAL MARRIED THEN 7
77 006100         MOVE B"1" TO IN01 IN DETAIL-0-INDIC
   006200     ELSE
78 006300         MOVE B"0" TO IN01 IN DETAIL-0-INDIC.
79 006400     WRITE PERSREPT-REC FORMAT IS "DETAIL" 8
   006500     INDICATORS ARE DETAIL-0-INDIC
80 006600         AT EOP PERFORM HEADING-LINE. 9
   006700 HEADING-LINE.
81 006800     MOVE HEAD-ORDER TO ORDERTYPE
82 006900     WRITE PERSREPT-REC FORMAT IS "HEADING".
   007000
          * * * * *   E N D   O F   S O U R C E   * * * * *

```

Figure 75 (Part 2 of 2). Example of the END-OF-PAGE Condition

- 1 The externally described printer file is assigned to device FORMATFILE.
- 2 The Format 2 COPY statement is used to copy the fields for the printer file into the program.
- 3 Note that, although the fields in format DETAIL will be printed on three separate lines, they are defined in one record.
- 4 COPY-DDS is used to copy the indicators used in the printer file into the program.
- 5 Paragraph PROCESS-RECORD processes PRINT-RECORD for each employee record.
- 6 All fields in the employee record are moved to the record for format DETAIL.
- 7 If the employee is married, indicator 01 is turned on; if not, the indicator is turned off, preventing the spouse's name field in DETAIL from being printed.
- 8 Format DETAIL is printed with indicator 01 passed to control printing.
- 9 If the number of lines per page has been exceeded, END-OF-PAGE occurs. The format HEADING is printed on a new page.

File	Keying Instruction	Graphic	Description	Page	of
Programmer	Date	Key			

Sequence Number	Form Type	ANSI/ISO Comment (A/D) (1)	Conditioning				Name	Length	Reference (R)	Data Type/Keyboard Shift	Optional Positions	Usage (B/C/D/E/P/M/N/Z/P)	Location		Functions
			Indicator	Indicator	Indicator	Indicator							Line	Pos	
1	A*					PRINTER FILE DDS FOR									
2	A*					FORMATFILE EXAMPLE									
3	A*												1	INDARA REF(PERSFILE)	
4	A*													SKIPB(1) SPACEA(3)	
5	A*					R HEADING 2								15'PERSONNEL LISTING' 3	
6	A*													UNDERLINE	
7	A*													33'- ORDERED BY'	
8	A*					ORDERTYPE	15							48	
9	A*													80DATE EDTCDE(Y)	
10	A*													83TIME	
11	A*													115'PAGE:' 4	
12	A*													+1PAGNBR EDTCDE(3)	
13	A*					R DETAIL 5								SPACEA(3) 6	
14	A*					LINE 1								1'NAME:'	
15	A*					NAME	R							11UNDERLINE	
16	A*					EMPLNO	R							55'EMPLOYEE NUMBER:'	
17	A*													73	
18	A*					BIRTHDATE	R							87'DATE OF BIRTH:'	
19	A*													103SPACEA(1) 7	
20	A*					LINE 2								1'ADDRESS:'	
21	A*					ADDRESS1	R							11	
22	A*													55'MARITAL STATUS:'	
23	A*					MARSTAT	R							73	
24	A*	01												87'SPOUSE'S NAME:'	
25	A*	01	3			SPOUSENAMER								103	
26	A*					LINE 3								11SPACEB(1)	
27	A*					ADDRESS2	R							55'CHILDREN:'	
28	A*													73EDTCDE(3) 9	
29	A*					NUMCHILD	R								

Figure 76 (Part 2 of 2). DDS Example of the Use of Externally Described Printer Files Assigned to a Device of FORMATFILE

- 1 INDARA specifies that a separate indicator area is to be used for the file.
- 2 HEADING is the format name that provides headings for each page.
- 3 SKIPB(1) and SPACEA(3) are used to:
 - 1. Skip to line 1 of the next page before format HEADING is printed.
 - 2. Leave 3 blank lines after format HEADING is printed.
- 4 DATE, TIME, and PAGNBR are used to have the current date, time and page number printed automatically when format HEADING is printed.
- 5 DETAIL is the format name used to print the detail line for each employee in the personnel file.
- 6 SPACEA(3) causes three lines to be left blank after each employee detail line.
- 7 SPACEA(1) causes a blank line to be printed after the field BIRTHDATE is printed. As a result, subsequent fields in the same format are printed on a new line.

- 8** 01 means that these fields are printed only if the COBOL program turns indicator 01 on and passes it when format DETAIL is printed.
- 9** EDTCDE(3) is used to remove leading zeros when printing this numeric field.

Chapter 10. DISK and DATABASE Files

Database files, which are associated with the COBOL devices of DATABASE and DISK, can be:

- Externally described files, whose fields are described to OS/400 through DDS
- Program-described files, whose fields are described in the program that uses the file.

All database files are created by OS/400 Create File commands. See the *Database Guide* for a description of the Create File commands for database files.

DATABASE versus DISK Files

Assigning a file to DISK in COBOL restricts the user to traditional DISK processing. The use of DATABASE as the device permits the user to make use of the special COBOL/400 database features such as formats and duplicate record keys.

Processing Methods for DISK and DATABASE Files

COBOL Indexed Files

An indexed file is a file whose access path is built on key values. The user must create a keyed access path for an indexed file by using DDS.

To write standard ANSI X3.23-1985 COBOL programs that access an indexed file, you must create the file with certain characteristics. The following table lists these characteristics and what controls them:

Characteristic	Control
The file must be a physical file.	The CL command CRTPF
The file cannot have records with duplicate key values.	The DDS keyword UNIQUE
The file cannot be a shared file.	The CL command CRTPF
A key must be defined for the file.	DDS
Keys must be in ascending sequence.	DDS
Keys must be contiguous within the record.	DDS
Key fields must be alphanumeric. They cannot be numeric only.	DDS
The value of the key used for sequencing must include all 8 bits of every byte.	DDS
A starting position for retrieving records cannot be specified.	The CL command OVRDBF
Select/omit level keywords cannot be used for the file.	DDS

An indexed file is identified by the ORGANIZATION IS INDEXED clause of the SELECT statement.

The key fields identify the records in an indexed file. The user specifies the key field in the RECORD KEY clause of the SELECT statement. The RECORD KEY data item must be defined within a record description for the indexed file. If there are multiple record descriptions for the file, only one need contain the RECORD KEY data name. However, the same positions within the record description that contain the RECORD KEY data item are accessed in the other record descriptions as the KEY value for any references to the other record descriptions for that file.

An indexed file can be accessed sequentially, randomly by key, or dynamically.

Valid RECORD KEYS

The DDS for the file specifies the fields to be used as the key field. If the file has multiple key fields, the key fields must be contiguous in each record unless RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified.

When the DDS specifies only one key field for the file, the RECORD KEY must be a single field of the same length as the key field defined in the DDS.

If a Format 2 COPY statement is specified for the file, the RECORD KEY clause must specify one of the following:

- The name used in the DDS for the key field if the name is not a COBOL reserved word.
- The name used in the DDS for the key field with -DDS added to the end if the name is a COBOL reserved word.
- The data name defined with the proper length and at the proper location in a program-described record description for the file.
- EXTERNALLY-DESCRIBED-KEY. This keyword specifies that the keys defined in DDS for each record format are to be used for accessing the file. These keys can be noncontiguous. They can be defined at different positions within the record format.

When the DDS specifies multiple contiguous key fields, the RECORD KEY data name must be a single field with its length equal to the sum of the lengths of the multiple key fields in the DDS. If a Format 2 COPY statement is specified for the file, there must also be a program-described record description for the file that defines the RECORD KEY data name with the proper length and at the proper position in the record.

Contiguous items are consecutive elementary or group items in the Data Division that are contained in a single data hierarchy.

Referring to a Partial Key

A generic START statement allows the use of a partial key. The KEY IS phrase is required.

Refer to the "START Statement" in the *COBOL/400* Reference* for information about the rules for specifying a search argument that refers to a partial key.

Figure 77 on page 239 shows an example of generic START statements using a program-described file.

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S COPYNAME  CHG DATE
 7 000700 FILE-CONTROL.
 8 000800 SELECT FILE-1 ASSIGN TO DISK-FILE1
 9 000900 ACCESS IS DYNAMIC RECORD KEY IS FULL-NAME IN FILE-1
10 001000 ORGANIZATION IS INDEXED.
11 001100 DATA DIVISION.
12 001200 FILE SECTION.
13 001300 FD FILE-1 LABEL RECORDS ARE STANDARD.
14 001400 01 RECORD-DESCRIPTION.
15 001500 03 FULL-NAME.
16 001600 05 LAST-AND-FIRST-NAMES.
17 001700 07 LAST-NAME PIC X(20).
18 001800 07 FIRST-NAME PIC X(20).
19 001900 05 MIDDLE-NAME PIC X(20).
20 002000 03 LAST-FIRST-MIDDLE-INITIAL-NAME REDEFINES FULL-NAME
21 002100 PIC X(41).
22 002200 03 REST-OF-RECORD
    002300/
23 002400 PROCEDURE DIVISION.
    002500 START-PROGRAM.
24 002600 OPEN INPUT FILE-1.
    002700*
    002800* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
    002900* "SMITH"
25 003000 MOVE "SMITH" TO LAST-NAME.
26 003100 START FILE-1 KEY IS EQUAL TO LAST-NAME
27 003200 INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR " LAST-NAME
28 003300 GO-TO ERROR ROUTINE.
    003400*
    003500*
    003600*
    003700*
    003800* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
    003900* "SMITH" AND A FIRST NAME OF "ROBERT"
29 004000 MOVE "SMITH" TO LAST-NAME.
30 004100 MOVE "ROBERT" TO FIRST-NAME.
31 004200 START FILE-1 KEY IS EQUAL TO LAST-AND-FIRST-NAMES
32 004300 INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
    004400 LAST-AND-FIRST-NAMES
33 004500 GO-TO ERROR ROUTINE.
    004600*
    004700*
    004800*
    004900*
    005000* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
    005100* "SMITH", AND A FIRST NAME OF "ROBERT", AND A MIDDLE INITIAL OF "M"
34 005200 MOVE "SMITH" TO LAST-NAME.
35 005300 MOVE "ROBERT" TO FIRST-NAME.
36 005400 MOVE "M" TO MIDDLE-NAME.
37 005500 START FILE-1 KEY IS EQUAL TO LAST-AND-FIRST-MIDDLE-INITIAL-NAME
38 005600 INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
    005700 LAST-FIRST-MIDDLE-INITIAL-NAME
39 005800 GO-TO ERROR ROUTINE.
    005900
    006000
    006100 ERROR-ROUTINE.
40 006200 STOP-RUN.

```

Figure 77. Generic START Statements Using a Program-Described File

Figure 78 and Figure 79 show an example of generic START statements using an externally described file.

SEQNBR	*	1	2	3	4	5	6	7	8	DATE
100	A									UNIQUE
200	A	R	RDE							TEXT('RECORD DESCRIPTION')
300	A		FNAME		20					TEXT('FIRST NAME')
400	A		MINAME		1					TEXT('MIDDLE INITIAL NAME')
500	A		MNAME		19					TEXT('REST OF MIDDLE NAME')
600	A		LNAME		20					TEXT('LAST NAME')
700	A		PHONE		10	0				TEXT('PHONE NUMBER')
800	A		DATA		40					TEXT('REST OF DATA')
900	A		K LNAME							
1000	A		K FNAME							
1100	A		K MINAME							
1200	A		K MNAME							

Figure 78. Generic START Statements Using an Externally Described File -- DDS

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S COPYNAME  CHG DATE
 7 000700 FILE-CONTROL.
 8 000800 SELECT FILE-1 ASSIGN TO DATABASE-NAMES
 9 000900 ACCESS IS DYNAMIC RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
10 001000 ORGANIZATION IS INDEXED.
11 001100 DATA DIVISION.
12 001200 FILE SECTION.
13 001300 FD FILE-1 LABEL RECORDS ARE STANDARD.
14 001400 01 RECORD-DESCRIPTION
15 001500 COPY DDS-RDE IN NAMES-PUBS.
17 +000001
+000002* FROM FILE NAMES OF LIBRARY XAMPLIB RDE
+000003* RECORD DESCRIPTION RDE
18 +000004 05 RDE. RDE
+000005* RECORD KEY FOR INDEXED FILE, KEY'0002 KEY FIELD NAME FNAME RDE
19 +000006 06 FNAME PIC X(20). RDE
+000007* FIRST NAME RDE
+000008* RECORD KEY FOR INDEXED FILE, KEY'0003 KEY FIELD NAME MINAME RDE
20 +000009 06 MINAME PIC X(1). RDE
+000010* MIDDLE INITIAL NAME RDE
+000011* RECORD KEY FOR INDEXED FILE, KEY'0004 KEY FIELD NAME MNAME RDE
21 +000012 06 MNAME PIC X(19). RDE
+000013* REST OF MIDDLE NAME RDE
+000014* RECORD KEY FOR INDEXED FILE, KEY'0001 KEY FIELD NAME LNAME RDE
22 +000015 06 LNAME PIC X(20). RDE
+000016* LAST NAME RDE
23 +000017 06 PHONE PIC S9(10). COMP-3 RDE
+000018* PHONE NUMBER RDE
24 +000019 06 DATA-DDS PIC X(40). RDE
+000020* REST OF DATA RDE
25 001600 66 MIDDLE-NAME RENAMES MINAME THRU MNAME.
001700/
26 001800 PROCEDURE DIVISION.
001900 START PROGRAM.
27 002000 OPEN INPUT FILE-1.
002100*
002200* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME
002300* OF "SMITH"
28 002400 MOVE "SMITH" TO LNAME.
29 002500 START FILE-1 KEY IS EQUAL TO LNAME
30 002600 INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR " LNAME
31 002700 GO TO ERROR-ROUTINE.
002800* .
002900* .
003000* .
003100*
003200* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME
003300* OF "SMITH" AND A FIRST NAME OF "ROBERT"
32 003400 MOVE "SMITH" TO LNAME.
33 003500 MOVE "ROBERT" TO FNAME.
34 003600 START FILE-1 KEY IS EQUAL TO LNAME, FNAME
35 003700 INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
003800 LNAME " " FNAME
36 003900 GO TO ERROR-ROUTINE.
004000* .
004100* .
004200* .
004300*
004400* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
004500* "SMITH", A FIRST NAME OF "ROBERT", AND A MIDDLE INITIAL OF "M"
32 004600 MOVE "SMITH" TO LNAME.
33 004700 MOVE "ROBERT" TO FNAME.
33 004800 MOVE "M" TO MINAME.
34 004900 START FILE-1 KEY IS EQUAL TO LNAME, FNAME, MINAME
35 005000 INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
005100 LNAME SPACE FNAME SPACE MINAME
42 005200 GO TO ERROR-ROUTINE.
005300
005400
005500 ERROR-ROUTINE.
005600 STOP-RUN.

```

Figure 79. Generic START Statements Using an Externally Described File

Logical File Considerations

When a logical file with multiple record formats, each having associated key fields, is processed as an indexed file in COBOL, the following restrictions and considerations apply:

- The FORMAT phrase must be specified on all WRITE statements to the file unless a Record Format Selector Program exists and has been specified in the FMTSLR parameter of the Create Logical File (CRTLF) command, the Change Logical File (CHGLF) command, or the Override Database File (OVRDBF) command. For information on the use of format selector programs, refer to the *Database Guide*.
- If the access mode is RANDOM or DYNAMIC, and the DUPLICATES phrase is not specified for the file, the FORMAT phrase must be specified on all DELETE and REWRITE statements.
- When the FORMAT phrase is not specified, only the portion of the RECORD KEY data item that is common to all record formats for the file is used by the system as the key for the I/O statement. When the FORMAT phrase is specified, only the portion of the RECORD KEY data item that is defined for the specified record format is used by the system as the key. See the *Database Guide* for more information on logical file processing.
- When *NONE is specified as the first key field for any format in a file, records can only be accessed sequentially. When a file is read randomly:
 - If a format name is specified, the first record with the specified format is returned.
 - If a format name is not specified, the first record in the file is returned.In both cases, the value of the RECORD KEY data item is ignored.
- For a program-defined key field:
 - Key fields within each record format must be contiguous.
 - The first key field for each record format must begin at the same relative position within each record.
 - The length of the RECORD KEY data item must be equal to the length of the longest key for any format in the file.
- For an EXTERNALLY-DESCRIBED-KEY:
 - Key fields within each record format can be noncontiguous.
 - The key fields can begin at different positions in each record format.

Figure 80 on page 243 and Figure 81 on page 244 show examples of how to use DDS to describe the access path for indexed files.

File		Keying Instruction	Graphic				Description	Page of
Programmer	Date		Key					

Sequence Number	Form Type	Alt/Dr/Comment	Conditioning				Name	Length	Reference	Data Type/Key/Shift	Decimal Position	Location		Functions
			Indicator	Not (N)	Indicator	Not (N)						Line	Pos	
1														
2														
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														
24														
25														
26														
27														
28														
29														
30														
31														
32														
33														
34														
35														
36														
37														
38														
39														
40														
41														
42														
43														
44														
45														
46														
47														
48														
49														
50														
51														
52														
53														
54														
55														
56														
57														
58														
59														
60														
61														
62														
63														
64														
65														
66														
67														
68														
69														
70														
71														
72														
73														
74														
75														
76														
77														
78														
79														
80														

Figure 80. Using Data Description Specifications to Define the Access Path for an Indexed File

Data description specifications must be used to create the access path for a program-described indexed file.

In the DDS for the record format FORMATA for the logical file ORDDTLL, the field ORDERN, which is five digits long, is defined as the key field. The definition of ORDERN as the key field establishes the keyed access for this file. Two other fields, FLDA and FLDB, describe the remaining positions in this record as character fields.

The program-described input field ORDDTLL is described in the FILE-CONTROL section in the SELECT clause as an indexed file.

The COBOL descriptions of each field in the FD entry must agree with the corresponding description in the DDS file. The RECORD KEY data item must be defined as a five-digit numeric integer beginning in position 15 of the record.

File		Keying Instruction	Graphic	Description		Page	of
Programmer	Date		Key				
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88
89	90	91	92	93	94	95	96
97	98	99	100				

Figure 81. Data Description Specifications for Defining the Access Path (a Composite Key) of an Indexed File

In this example, the data description specifications define two key fields for the record format **FORMAT** in the logical file **ORDDTLL**. For the two fields to be used as a composite key for a program-described indexed file, the key fields must be contiguous in the record.

The COBOL description of each field must agree with the corresponding description in the DDS file. A 10-character item beginning in position 15 of the record must be defined in the **RECORD KEY** clause of the file-control entry. The COBOL descriptions of the DDS fields **ORDERN** and **ITEM** would be subordinate to the 10-character item defined in the **RECORD KEY** clause.

COBOL Relative Files

A COBOL relative file is a file to be processed by a relative record number. To process a file by relative record number, you must specify **ORGANIZATION IS RELATIVE** in the **SELECT** statement for the file. A relative file can be accessed sequentially, randomly by record number, or dynamically.

To write standard ANSI X3.23-1985 COBOL programs that access a relative file, you must create the file with certain characteristics. The following table lists these characteristics and what controls them.

Characteristic	Control
The file must be a physical file.	The CL command CRTPF
The file cannot be a shared file.	The CL command CRTPF
No key can be specified for the file.	DDS
A starting position for retrieving records cannot be specified.	The CL command OVRDBF
Select/omit level keywords cannot be used for the file.	DDS
Records in the file cannot be reused.	The CL command CRTPF

For a COBOL file with an organization of RELATIVE, the Reorganize Physical File Member (RGZPFM) CL command can:

- Remove all deleted records from the file. Because COBOL initializes all relative file records to deleted records, any record that has not been explicitly written will be removed from the file. The relative record numbers of all records after the first deleted record in the file will change.
- Change the relative record numbers if the file has a key and the arrival sequence is changed to match a key sequence (with the KEYFILE parameter).

In addition, a Change Physical File (CHGPF) CL command bearing the REUSEDLT option can change the order of retrieved or written records when the file is operated on sequentially, because it allows the reuse of deleted records.

COBOL Sequential Files

A COBOL sequential file is a file in which records are processed in the order in which they were placed in the file, that is, in arrival sequence. For example, the tenth record placed in the file occupies the tenth record position and is the tenth record to be processed. To process a file as a sequential file, you must specify ORGANIZATION IS SEQUENTIAL in the SELECT clause, or omit the ORGANIZATION clause. A sequential file can only be accessed sequentially.

To write standard ANSI X3.23-1985 COBOL programs that access a sequential file, you must create the file with certain characteristics. The following table lists these characteristics and what controls them.

Characteristic	Control
The file must be a physical file.	The CL command CRTPF
The file cannot be a shared file.	The CL command CRTPF
No key can be specified for the file.	DDS
The file must have a file type of DATA.	The CL command CRTPF
Field editing cannot be used.	DDS
Line and position information cannot be specified.	DDS

Characteristic	Control
Spacing and skipping keywords cannot be specified.	DDS
Indicators cannot be used.	DDS
System-supplied functions such as date, time, and page number cannot be used.	DDS
Select/omit level keywords cannot be used for the file.	DDS
Records in the file cannot be reused.	The CL command CRTPF

To preserve the sequence of records in a file that you open in I/O (update) mode, do not change the file so that you can reuse the records in it. That is, do not use a Change Physical File (CHGPF) CL command bearing the REUSEDLT option.

Note: The COBOL/400 compiler does not check that the device associated with the external file is of the type specified in the device portion of assignment-name. The device specified in the assignment-name must match the actual device to which the file is assigned. See the "ASSIGN Clause" section of the *COBOL/400* Reference* for more information.

COBOL File Organization and AS/400 File Access Path Considerations

A file with a keyed sequence access path can be processed in COBOL as a file with INDEXED, RELATIVE, or SEQUENTIAL organization.

For a keyed sequence file to be processed as a relative file in COBOL, it must be a physical file, or a logical file whose members are based on one physical file member. For a keyed sequence file to be processed as a sequential file in COBOL, it must be a physical file, or a logical file that is based on one physical file member and that does not contain select/omit logic.

A file with an arrival sequence access path can be processed in COBOL as a file with RELATIVE or SEQUENTIAL organization. The file must be a physical file or a logical file where each member of the logical file is based on only one physical file member.

When sequential access is specified for a logical file, records in the file are accessed through the access path created with create file options.

File Processing Methods

Figure 82 on page 248 shows the valid processing methods and expected operation for combinations of organization, access mode, open state, I/O verb, and I/O verb modifiers.

All physical database files that are opened for OUTPUT are cleared. Database files with RELATIVE organization, and with dynamic or random access mode, are also initialized with deleted records.

New relative files opened for OUTPUT in sequential access mode are treated differently. Table 4 on page 247 summarizes conditions affecting them.

<i>Table 4. Initialization of Relative Output Files</i>			
File Access and CL Specifications	Conditions at Opening Time	Conditions at Closing Time	File Boundary
Sequential *INZDLT		Records not written are initialized	All increments
Sequential *INZDLT *NOMAX size		CLOSE succeeds File status is 0Q	Up to boundary of records written
Sequential *NOINZDLT			Up to boundary of records written
Random or dynamic	Records are initialized File is open		All increments
Random or dynamic *NOMAX size	OPEN fails File status is 9Q		File is empty

To extend a file boundary beyond the current number of records, but remaining within the file size, use the INZPFM command to add deleted records before processing the file. You need to do this if you receive a file status of 0Q, and you still want to add more records to the file.

Any attempt to extend a relative file beyond its current size results in a boundary violation.

To recover from a file status of 9Q, use the CHGPF command as described in the associated run-time message text.

Lengthy delays are normal when there remains an extremely large number of records (over 1 000 000) to be initialized to deleted records when the CLOSE statement runs.

When the first OPEN statement for the file is not OPEN OUTPUT, relative files should be cleared and initialized with deleted records before they are used. See the discussion of the CLRPFM and INZPFM commands in the *CL Reference* for more information.

The RECORDS parameter of the INZPFM command must specify *DLT. Overrides are applied when the clear and initialize operations are processed by COBOL, but not when they are processed with CL commands.

Lengthy delays in OPEN OUTPUT processing are normal for extremely large relative files (over 1 000 000 records) that you access in dynamic or random mode.

ORG	ACC	DEV	OPEN	READ	WRITE	START	REWRITE	DELETE	CLOSE	FORMAT	SELECT CLAUSE KEY IS
S S S S	S S S S	ANY ANY ANY ANY	INPUT OUTPUT I-0 EXTEND	X X	 X(F1) X		 X		X X X X	A1	
I I I	S S S	D/DB D/DB D/DB	INPUT OUTPUT I-0	X X	 X(F1) X	X X	 X	 X	X X X	B1 B1 B1	C1 C1 C1
I I I	R R R	D/DB D/DB D/DB	INPUT OUTPUT I-0	X X	 X(F1) X		 X	 X	X X X	B1 B1 B1	D1 D1 D1
I I I	D D D	D/DB D/DB D/DB	INPUT OUTPUT I-0	X X	 X(F1) X	X X	 X	 X	X X X	B1 B1 B1	D1 D1 D1
R R R	S S S	D/DB D/DB D/DB	INPUT OUTPUT I-0	X X	 X(G1)	X X	 X	 X	X X X		C1 C1 C1
R R R	R R R	D/DB D/DB D/DB	INPUT OUTPUT I-0	X X	 X(G1) X		 X	 X	X X X		E1 E1 E1
R R R	D D D	D/DB D/DB D/DB	INPUT OUTPUT I-0	X X	 X(G1) X	X X	 X	 X	X X X		E1 E1 E1
T	S	W	I-0	X	X				X	H1	
T	D	W	I-0	X(K1)	X(K1)		X		X	I1	J1
ORG:				ACC:				DEV:			
S = Sequential R = Relative I = Indexed T = TRANSACTION				S = Sequential R = Random D = Dynamic				ANY = Any Device D = DISK DB = DATABASE W = WORKSTATION			

Figure 82. Processing Methods Summary Chart

The following paragraphs explain the keys used in Figure 82.

X The combination is allowed.

A1 The FORMAT phrase is required for FORMATFILE files with multiple formats, and is not allowed for all other device files.

B1 The FORMAT phrase is optional for DATABASE files, and not allowed for DISK files. If the FORMAT phrase is not specified, the default format name of the file is used. The default format name of the file is the first format name defined in the file.

The special register, DB-FORMAT-NAME, can be used to retrieve the format name used on the last successful I/O operation.

C1 The SELECT clause KEY phrase is ignored except for the START statement. If the KEY phrase is not specified on the START statement, the RECORD KEY phrase or the RELATIVE KEY phrase in the SELECT clause is used and KEY = is assumed.

D1 The SELECT clause KEY phrase is used except for the START statement. If the KEY phrase is not specified on the START statement, the RECORD KEY phrase in the SELECT clause is used and KEY = is assumed.

NEXT, PRIOR, FIRST, or LAST can be specified only for the READ statement for DATABASE files with DYNAMIC access. If NEXT, PRIOR, FIRST, or LAST is specified, the SELECT clause KEY phrase is ignored.

- E1 The SELECT clause RELATIVE KEY phrase is used.
The NEXT phrase can be specified only for the READ statement for a file with DYNAMIC access mode. If NEXT is specified, the SELECT clause KEY phrase is ignored.
The RELATIVE KEY data item is updated with the relative record number for files with sequential access on READ operations.
- F1 A physical file opened for output is cleared.
- G1 A physical file opened for output is cleared and initialized to deleted records. There are some exceptions depending on the file size and the options specified. For more information, refer to Table 4 on page 247.
- H1 The FORMAT phrase is required for the WRITE statement.
- I1 The FORMAT phrase is required to distinguish between the subfile records and the subfile control record. The WRITE FORMAT IS control-record-format-name displays the subfile, but a READ FORMAT IS control-record-format-name is required to allow data to be entered and to cause the operator input for the subfile records on the display to be placed in the subfile.
- J1 The SELECT clause RELATIVE KEY phrase is used for READ, WRITE, and REWRITE statements that use the SUBFILE phrase, except that the READ SUBFILE NEXT MODIFIED uses the current system relative record number rather than the RELATIVE KEY data item. The RELATIVE KEY data item is updated with the relative record number for subfile records for READ statements with the NEXT MODIFIED clause.
- K1 The SUBFILE phrase is required when an I/O operation deals with a particular record rather than an entire file.

Descending File Considerations

Files created with a descending keyed sequence (in DDS) cause the READ statement NEXT, PRIOR, FIRST, and LAST phrases to work in a fashion exactly opposite that of a file with an ascending key sequence. In **descending key sequence**, the data is arranged in order from the highest value of the key field to the lowest value of the key field.

For example, READ FIRST retrieves the record with the highest key value, and READ LAST retrieves the record with the lowest key value. Files with a descending key sequence also cause the START qualifiers to work in the opposite manner. For example, START GREATER THAN positions the current record pointer to a record with a key less than the current key.

Chapter 11. COBOL/400 Programming Considerations

This chapter describes:

- Issuing a CL command from a COBOL program
- The CORRESPONDING phrase
- The LIKE clause
- Reference modification
- De-editing
- Performance considerations.

General-Use Programming Interface

Issuing a CL Command from a COBOL Program

You can issue a CL command from a COBOL program through a CALL to QCMDEXC.

In the following example program, the CALL to QCMDEXC (at sequence number 001600) results in the processing of the Add Library List Entry (ADDLIBLE) CL command (at sequence number 001100). The successful completion of the CL command results in the addition of the library, COBOLTEST, to the library list.

```
-A 1 B..+...2....+...3....+...4....+...5....+...6....+...7..
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. CMDXMPLE.
000300 ENVIRONMENT DIVISION.
000400 CONFIGURATION SECTION.
000500 SOURCE-COMPUTER. IBM-AS400.
000600 OBJECT-COMPUTER. IBM-AS400.
000700 DATA DIVISION.
000800 WORKING-STORAGE SECTION.
000900 01 PROGRAM-VARIABLES.
001000     05 CL-CMD          PIC X(33)
001100         VALUE "ADDLIBLE COBOLTEST".
001200     05 PACK-VAL       PIC 9(10)V9(5) COMP-3
001300         VALUE 18.
001400 PROCEDURE DIVISION.
001500 MAINLINE.
001600     CALL "QCMDEXC" USING CL-CMD PACK-VAL.
001700     STOP RUN.
```

Note: Do not use the Reclaim Resource (RCLRSC) Command in this situation. It cancels all programs higher in the program stack so that the STOP RUN statement in the program will cause a run-time exception.

For more information about QCMDEXC, see the *CL Programmer's Guide*.

End of General-Use Programming Interface

Using the **CORRESPONDING** Phrase

In the following example program, the **ADD CORRESPONDING** statement at sequence number 000270 adds **GROUP1 ITEM1** to **GROUP2 ITEM1**, and adds **GROUP1 ITEM2** to **GROUP2 ITEM2**. The **MOVE CORRESPONDING** statement at sequence number 000290 moves **GROUP1 ITEM1, ITEM2, ITEM3, and ITEM4** to **GROUP2 ITEM1, ITEM2, ITEM3, and ITEM4**.

The **MOVE CORRESPONDING** statement at sequence number 000300 is not processed because there are no corresponding items to move, and an error message is generated.

Figure 83 on page 253 was produced with the **PRTCORR** option in effect.


```

5738CB1 V2R2M0          AS/400 COBOL Source          XPLIB/CORR
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
 1 000010 IDENTIFICATION DIVISION.
 2 000020 PROGRAM-ID.      CORRPHRASE.
 3 000030 AUTHOR.          PROGRAMMER NAME.
 4 000040 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
 5 000050 DATE-WRITTEN. 05/24/91.
 6 000060 DATE-COMPILED. 05/24/91 11:09:11
 7 000070 ENVIRONMENT DIVISION.
 8 000080 CONFIGURATION SECTION.
 9 000090 SOURCE-COMPUTER. IBM-AS400.
10 000100 OBJECT-COMPUTER. IBM-AS400.
11 000110 DATA DIVISION.
12 000120 WORKING-STORAGE SECTION.
13 000130 01 GROUP1.
14 000140 05 ITEM1 PIC 99 VALUE 1.
15 000150 05 ITEM2 PIC 99 VALUE 2.
16 000160 05 ITEM3 PIC X(10) VALUE "GREEN".
17 000170 05 ITEM4 PIC X(10) VALUE "BLUE".
18 000180 01 GROUP2.
19 000190 05 ITEM1 PIC 99 VALUE 8.
20 000200 05 ITEM2 PIC 99 VALUE 9.
21 000210 05 ITEM3 PIC XXBX(8) VALUE SPACES.
22 000220 05 ITEM4 PIC X(10) VALUE SPACES.
23 000230 01 GROUP3.
24 000240 05 SPECIAL PIC XX.
25 000250 PROCEDURE DIVISION.
    000260 MAINLINE.
26 000270 ADD CORRESPONDING GROUP1 TO GROUP2.
    *      ** CORRESPONDING items for statement 26:
    *      **      ITEM1
    *      **      ITEM2
    *      ** End of CORRESPONDING items for statement 26
27 000280 SUBTRACT CORRESPONDING GROUP1 FROM GROUP2.
    *      ** CORRESPONDING items for statement 27:
    *      **      ITEM1
    *      **      ITEM2
    *      ** End of CORRESPONDING items for statement 27
28 000290 MOVE CORRESPONDING GROUP1 TO GROUP2.
    *      ** CORRESPONDING items for statement 28:
    *      **      ITEM1
    *      **      ITEM2
    *      **      ITEM3
    *      **      ITEM4
    *      ** End of CORRESPONDING items for statement 28
29 000300 MOVE CORRESPONDING GROUP3 TO GROUP2.
    *      ** CORRESPONDING items for statement 29:
    *      **      No CORRESPONDING items found
    *      ** End of CORRESPONDING items for statement 29
30 000310 STOP RUN.
          ***** END OF SOURCE *****
5738CB1 V2R2M0          AS/400 COBOL Messages          XPLIB/CORR
STMT
* 29 MSGID: LBL0336 SEVERITY: 10 SEQNBR: 000300
    Message . . . . : No CORRESPONDING items found. Statement
    ignored.
          ***** END OF MESSAGES *****
          Message Summary
Total   Info(0-4)   Warning(5-19)   Error(20-29)   Severe(30-39)   Terminal(40-99)
 1         0             1             0             0             0
Source records read . . . . . : 31
Copy records read . . . . . : 0
Copy members processed . . . . . : 0
Sequence errors . . . . . : 0
Highest severity message issued . . : 10
    LBL0901 00 Program CORR created in library XPLIB.
          ***** END OF COMPILATION *****

```

Figure 83. Example of the CORRESPONDING Phrase

LIKE Clause

The LIKE clause allows you to define the PICTURE, USAGE, SIGN, and BLANK WHEN ZERO characteristics of a data name by copying them from a previously defined data name. LIKE can only refer to a data name or index name, and such names must be uniquely qualified if they have been previously defined. It also allows you to change the length of the data name you define.

This clause is particularly helpful because you can use it to define identifiers in the Working-Storage Section of your program that have the same attributes as variables that you define using the COPY statement.

To create data name DEPTH with the same attributes as data name HEIGHT, write:

```
DEPTH LIKE HEIGHT
```

To create data name PROVINCE with the same attributes as data name STATE, except 1 byte longer, write:

```
PROVINCE LIKE STATE (+1)
```

This example shows how you can create data item WS-KEY3 with the same attributes as data item KEY3 in the Working-Storage Section:

```

S73BCB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B.+. . . . . 2. . . . . 3. . . . . 4. . . . . 5. . . . . 6. . . . . 7. IDENTFCN S

001400 FILE SECTION.
001500 FD FILE1.
001600 01 FILE1-REC.
001700 COPY DDS-ALL-FORMATS OF COPYDDS2.
+000001 05 COPYDDS2-RECORD PIC X(20).
+000002* I-O FORMAT: RECORD1 FROM FILE COPYDDS2 OF LIBRARY COPYLIB
+000003*
+000004*THE KEY DEFINITIONS FOR RECORD FORMAT RECORD1
+000005* NUMBER          NAME          RETRIEVAL  TYPE
+000006* 0001  KEY1-DDS          ASCENDING
+000007*          KEYNAME ORIGINATES FROM PHYSICAL FILE
+000008 05 RECORD1  REDEFINES COPYDDS2-RECORD.
+000009 06 KEY3     PIC X(8).
+000010 06 FILLER  REDEFINES KEY3.
+000011 07 KEY1-DDS  PIC X(4).
+000012 07 FILLER  PIC X(4).
+000013 06 DATA1  PIC X(12).
001000 WORKING-STORAGE SECTION.
001900 01 WS-KEY3 LIKE KEY3.
      * PICTURE IS X(8)

```

Figure 84. COPY DDS with the LIKE Clause

The LIKE clause cannot be used in conjunction with the REDEFINES, SIGN, USAGE, or PICTURE clauses. If you use any of these clauses with the LIKE clause, a duplication error occurs. Similarly, BLANK WHEN ZERO can only be specified in conjunction with the LIKE clause if the BLANK WHEN ZERO attribute has not been inherited by the LIKE clause.

A valid LIKE clause has the format of one of the following:

data-name-1 LIKE-clause xxxxx.

data-name-1 xxxxx LIKE-clause.

data-name-1 xxxxx LIKE-clause xxxxx.

The xxxxx is one or a combination of the following clauses: JUSTIFIED, SYNCHRONIZED, BLANK WHEN ZERO, VALUE, OCCURS.

The following show what the LIKE clause can do:

```
01 INCOME.
   05 ANNUAL-WAGES PIC 9(6)V9(2) COMP-3.
01 YTD-WAGES LIKE ANNUAL-WAGES.
* PICTURE IS 9(6)V9(2)
* USAGE IS PACKED-DECIMAL
```

```
01 RATES.
   05 MONTHLY-RATE PIC 9(3).
66 GROSS-RATE RENAMES MONTHLY-RATE.
01 NET-RATE LIKE GROSS-RATE.
* PICTURE IS 9(3)
```

```
01 FAMILY-NAME PIC X(20) VALUE "JONES".
01 GIVEN-NAME LIKE FAMILY-NAME.
* PICTURE IS X(20)
```

```
01 EMPLOYEE-NUMBER PIC X(6).
01 DEPARTMENT-MEMBERS.
   05 DEPT-EMPLOYEE-NUMBER LIKE EMPLOYEE-NUMBER
      OCCURS 10 TIMES.
* PICTURE IS X(6)
```

Note: DEPARTMENT-MEMBERS in the above example is 60 bytes long.

```
   05 TENANT-NAME PIC X(20) OCCURS 10 TIMES.
01 RENEWAL-RECORD.
   05 RENEWAL-MONTH PIC X(3).
   05 RENEWAL-NAME LIKE TENANT-NAME.
* PICTURE IS X(20)
```

Note: RENEWAL-RECORD in the above example is only 23 bytes long.

The PICTURE portion of the generated comment is shown in a concise format.

Note: A numeric field with the BLANK WHEN ZERO attribute is considered to be a numeric edited field.

```
01 ORDER-DETAILS.  
   05 ORDER-TYPE      PIC XX.  
   05 ORDER-CODE LIKE ORDER-TYPE.  
* PICTURE IS X(2)
```

```
01 FASTENINGS.  
   05 NAILS           PIC 9V99 BLANK WHEN ZERO.  
   05 RIVETS LIKE NAILS.  
* PICTURE IS 9V9(2)  
* BLANK WHEN ZERO
```

```
01 MORTGAGE-PAYMENT.  
   05 MORTGAGE-TOTAL PIC S999V99 SIGN IS LEADING SEPARATE.  
   05 MORTGAGE-INTEREST LIKE MORTGAGE-TOTAL.  
* PICTURE IS S9(3)V9(2)  
* SIGN IS LEADING SEPARATE
```

```
01 PROFIT.  
   05 GROSS-PROFIT  PIC 999(3)PP(5).  
   05 NET-PROFIT LIKE GROSS-PROFIT.  
* PICTURE IS 9(5)P(6)
```

You can use an integer to increase or decrease the length of the field. The following example shows how to increase the field length of WEEKLY-AMOUNT:

```
01 WEEKLY-AMOUNT      PIC 9(3).  
01 ANNUAL-AMOUNT LIKE WEEKLY-AMOUNT (+3).  
* PICTURE IS 9(6)
```

You should also be aware of the following:

- Any field that has attributes of BLANK WHEN ZERO is considered to be an edited field
- If an integer of zero is specified, an informational message is generated.

Only the integer portion of the field length can be increased or decreased. You cannot change the number of decimal places in a data item.

The default attributes, SIGN IS TRAILING and USAGE IS DISPLAY, are never printed as comments following a LIKE operation.

When you use the LIKE clause, the normal data name qualification rules apply to the parent data name; however, the referenced data name must be uniquely qualified if it has previously been defined more than once. For example:

```

01 COMBINATIONS.
   05 PHENOTYPE      PIC XX.
   05 GENOTYPE LIKE PHENOTYPE.
* PICTURE IS X(2)
01 PHENOTYPE-TRAITS.
   05 PHENOTYPE      PIC X(30).
   05 PHENO-GROUP LIKE PHENOTYPE OF COMBINATIONS.
* PICTURE IS X(2)

```

If you do not uniquely qualify the parent data name, the compiler assigns it a picture clause of X(2), and you receive an error message.

The use of the LIKE clause can sometimes result in group items that are not valid. For example, if you define a COMP-4 group item and then use the LIKE clause to define a COMP-3 item that is subordinate to it, an error will result.

The following example is valid:

```

77 SWITCHES-IN-STOCK PIC S99.
01 PARTS-ON-ORDER  SIGN IS LEADING SEPARATE.
   05 SWITCHES-ON-ORDER LIKE SWITCHES-IN-STOCK.
* PICTURE IS S9(2)

```

Note: SWITCHES-ON-ORDER has the same SIGN attribute (SIGN IS TRAILING) as SWITCHES-IN-STOCK.

In the case of B LIKE A where A is a group item, B cannot be subordinate to A. In all other cases, B will be defined as an alphanumeric item with a length in bytes equal to the length of group A.

```

01 GARAGE-1.
   05 STD-PARKING-1 PIC 9(3).
01 GARAGE-2.
   05 STD-PARKING-2 PIC 9(3) COMP-3.
77 VACANCIES-1 LIKE GARAGE-1.
* PICTURE IS X(3)
77 VACANCIES-2 LIKE GARAGE-2.
* PICTURE IS X(2)

```

STD-PARKING-1 is a zoned numeric field, so VACANCIES-1 requires 3 bytes of storage. STD-PARKING-2 is a packed numeric field, so VACANCIES-2 requires only 2 bytes of storage.

You can use the LIKE clause with the USAGE IS POINTER clause:

```
01 CUSTOMER-RECORD.  
  05 CUST-NAME          PIC X(16).  
  05 CUST-ADDR-POINTER POINTER.  
  05 CUST-STATS-POINTER LIKE CUST-ADDR-POINTER.  
* USAGE IS POINTER  
  05 CUST-NUMBER        PIC S9(8).
```

Note: You cannot use the LIKE clause to change the length of a pointer.

For additional information about the LIKE clause, see the *COBOL/400* Reference*.

End of IBM Extension

Reference Modification

Reference modification allows you to reference substrings of a data item. You simply specify the position within the data item at which you want the substring to start, and the length of the substring. The length is optional: if you omit it, it automatically extends to the end of the data item.

You can write both the starting position and the length value as integer literals, data items, or arithmetic expressions.

The starting position must be at least 1, and cannot be greater than the length of the referenced data item. The length must be at least 1.

The result of adding the starting position to the length specification, then subtracting 1, must fall between 1 and the total length of the referenced data item, inclusive. When the length value is greater than the total length of the data item, an error results.

For additional information on reference modification, see the *COBOL/400* Reference*.

The *RANGE generation option produces code to detect out-of-range reference modification conditions, and to flag violations with a run-time message.

Suppose you want to retrieve the current time from the system, and display its value in an expanded format. You can retrieve it with the ACCEPT statement, which returns the hours, minutes, seconds, and hundredths of seconds in the format:

```
HHMMSSss
```

However, you may want to view the current time in the format:

```
HH:MM:SS
```

Without reference modification, you must define the following data items:

```

01 TIME-GROUP.
  05 INTERESTING-FIELDS.
    10 HOURS PIC XX.
    10 MINUTES PIC XX.
    10 SECONDS PIC XX.
  05 UNINTERESTING-FIELDS.
    10 HUNDREDTHS-OF-SECONDS PIC XX.
01 EXPANDED-TIME-GROUP.
  05 INTERESTING-FIELDS.
    10 HOURS PIC XX.
    10 FILLER PIC X VALUE ":".
    10 MINUTES PIC XX.
    10 FILLER PIC X VALUE ":".
    10 SECONDS PIC XX.

```

The following code would retrieve the time value, convert it to its expanded format, and display the new value:

```

ACCEPT TIME-GROUP FROM TIME
MOVE CORRESPONDING
  INTERESTING-FIELDS OF TIME-GROUP TO
  INTERESTING-FIELDS OF EXPANDED-TIME-GROUP
DISPLAY "CURRENT TIME IS: " EXPANDED-TIME-GROUP

```

With reference modification, you do not need to provide names for the subfields that describe the time elements. The only data definition you must have is:

```

01 REFMOD-TIME-ITEM PIC X(8).

```

The code to retrieve and expand the time value appears as follows:

```

ACCEPT REFMOD-TIME-ITEM FROM TIME
DISPLAY "CURRENT TIME IS: "
  REFMOD-TIME-ITEM (1:2)
  ":"
  REFMOD-TIME-ITEM (3:2)
  ":"
  REFMOD-TIME-ITEM (5:2)

```

The following example shows a reference beginning at character position 1, for a length of 2, thus retrieving the portion of the time value that corresponds to the number of hours:

```

REFMOD-TIME-ITEM (1:2)

```

The following example shows a reference beginning at character position 3, for a length of 2, thus retrieving the portion of the time value that corresponds to the number of minutes:

```

REFMOD-TIME-ITEM (3:2)

```

The following example shows a reference beginning at character position 5, for a length of 2, thus retrieving the portion of the time value that corresponds to the number of seconds:

```

REFMOD-TIME-ITEM (5:2)

```

Reference Modification with Variable-length Tables

Suppose you are using variable-length tables to contain names:

```
01 NAME-GROUP.  
  05 NAME-LENGTH PIC 99.  
  05 NAME-PORTION.  
    10 FILLER PIC X  
        OCCURS 1 TO 17 TIMES  
        DEPENDING ON NAME-LENGTH.  
01 NEW-NAME-GROUP.  
  05 NEW-NAME-LENGTH PIC 99.  
  05 NEW-NAME-PORTION.  
    10 FILLER PIC X  
        OCCURS 1 TO 17 TIMES  
        DEPENDING ON NEW-NAME-LENGTH.
```

The OCCURS DEPENDING ON object of the NAME-PORTION table is set to 8 so that only the first eight occurrences of the table are referenced, even though the entire 17 bytes of NAME-PORTION are filled in.

0	8	W	I	L	L	I	A	M	S		T	H	O	M	A	S	J
---	---	---	---	---	---	---	---	---	---	--	---	---	---	---	---	---	---

Suppose you want to change the value in the item NAME-PORTION without changing the portion of the item that is defined beyond the currently defined length. You might try coding:

```
MOVE NEW-NAME-GROUP TO NAME-GROUP
```

in which the contents of NEW-NAME-GROUP are:

0	5	S	M	I	T	H					M	I	C	H	A	E	L
---	---	---	---	---	---	---	--	--	--	--	---	---	---	---	---	---	---

According to the rules for the MOVE statement, the entire contents of the receiving field NAME-GROUP would be replaced. This problem can be avoided by using reference modification in the MOVE statement:

```
MOVE NEW-NAME-GROUP TO NAME-GROUP ( 1 :LENGTH OF NAME-GROUP )
```

By specifying the reference modification with the LENGTH OF special register, the length of NAME-GROUP is now determined by the value in the NAME-LENGTH variable.

The new value of NAME-GROUP will be:

0	5	S	M	I	T	H					T	H	O	M	A	S	J
---	---	---	---	---	---	---	--	--	--	--	---	---	---	---	---	---	---

Reference Modification Using Data Names

So far, all of the reference modification examples have shown simple numeric literals as the reference modification starting position and length values. These values can also be data items or arithmetic expressions.

Suppose a field contains some right-justified characters, and you want to move them to another field, but left-justified instead of right. Using reference modification and an INSPECT statement, you can do it.

The program would have the following data:

```
01 LEFTY PIC X(30).
01 RIGHTY PIC X(30)
JUSTIFIED RIGHT.
01 I PIC 9(9)
USAGE BINARY.
```

The program then counts the number of leading spaces, and, using arithmetic expressions in a reference modification expression, moves the right-justified characters into another (left-justified) field:

```
MOVE SPACES TO LEFTY
MOVE ZERO TO I
INSPECT RIGHTY
TALLYING I FOR LEADING SPACE
IF I IS LESS THAN 30 THEN
MOVE RIGHTY ( I + 1 : 30 - I ) TO LEFTY
END-IF
```

The MOVE statement transfers characters from RIGHTY, beginning at the position computed in I + 1, for a length that is computed in 30 - I, into the field LEFTY.

Reference Modification with Subscripting

define a table like this:

```
01 ANY-TABLE.
05 TABLE-ELEMENT PIC X(10)
OCCURS 3 TIMES
VALUE "ABCDEFGHIJ".
```

You can change both the third and fourth bytes of the first element of TABLE-ELEMENT to the value "??" with the following MOVE statement:

```
MOVE "??" TO TABLE-ELEMENT ( 1 ) ( 3 : 2 )
```

This statement will move the value "??" into table element number 1, beginning at character position 3, for a length of 2.

ANY-TABLE would look like this before the change:

ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ

It would look like this after the change:

AB??EFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ

De-editing

De-editing allows you to move a numeric-edited data item into a numeric or numeric-edited receiving data item. The compiler accomplishes this by first establishing the unedited value of the numeric-edited item. It then moves the unedited value to the receiver.

De-editing can occur in operations such as MOVE and INITIALIZE. A VALUE clause does not de-edit.

Note that unedited numeric values can involve signs.

Suppose that you use a character field to contain a numeric value that displays on the terminal, and also to contain a value that the computer operator supplies. Suppose that this field has the following definition:

- One character position for a sign (to contain a space if the numeric field is positive or zero, or a minus sign if the numeric field is negative);
- Six digit positions, in which leading zeros are represented by spaces;
- A decimal point;
- Two decimal-digit positions.

The data item that you use to define this field would look like this:

```
01 NUM-EDIT PIC -Z(6).9(2) USAGE IS DISPLAY.
```

You could initialize this field using this statement:

```
MOVE ZEROS TO NUM-EDIT
```

and when it displays on the terminal, it would contain the value `bbbbbbb.00`.

Later, the computer operator might use this field for data entry. If the operator puts `bbbb123.45` into the field, you can obtain the numeric value of the field by moving it into a data item defined as:

```
01 NUMERIC-ITEM PIC S9(6)V9(2) USAGE IS PACKED-DECIMAL.
```

This statement:

```
MOVE NUM-EDIT TO NUMERIC-ITEM
```

causes de-editing to take place, whereby the numeric item receives the numeric value of the numeric-edited field NUM-EDIT. As a result, the numeric item contains the value `+123.45`.

De-editing Examples

Table 5 on page 263 and Table 6 on page 263 show examples of COBOL/400 de-editing.

Source Picture	Source Value	Receiving Picture	Receiving Value
\$+++ ,+++ .++	\$bbb+123.45	S9(5)V9(5) USAGE IS DISPLAY	+123.45
\$+++ ,+++ .++	\$b-1,234.56	S9(5)V9(5) USAGE IS BINARY	-1234.56
*****.999+	**123.450-	S9(5)V9(5) USAGE IS PACKED-DECIMAL	-123.45

Source Picture	Source Value	Receiving Picture	Receiving Value
\$+++ ,+++ .++	\$bbb+123.45	\$\$\$\$,\$\$\$,\$\$CR	bbb\$123.45bb
\$+++ ,+++ .++	\$b-1,234.56	-----,----.99	bb-1,234.56
*****.999+	**123.450-	ZZBZZZBVZZZ	bbb123b450
ZZZ999CR	b12345bb	\$++++9999	\$bb+12345
ZZZ999CR	b12345CR	999999.99-	012345.00-

Handling Data Errors

The compiler provides some run-time error checking for move operations that involve de-editing.

The compiler does not perform this checking for source values of zero, and it ignores simple insertion characters (such as / B 0 , .).

Sign Test

The compiler validates signs in numeric-edited source items according to the following rules.

PICTURE Definition	Allowable Contents
Fixed +	+ or -
Fixed -	b or -
CR	bb or CR
DB	bb or DB

If these rules are disobeyed, a sign error occurs, and the program stops.

Float Test

If the source has a string of floating characters, this test verifies the correctness of leading floating characters in the data field.

The rules for the float test are:

- If the source PICTURE clause contains floating \$ symbols, the first non-blank character in the relevant portion of the source field (positions 2 through 7 in the example) must be a \$, and its location must be correct according to the rules for PICTURE clause editing. (See the *COBOL/400* Reference* for more information about these rules.)

For example:

Location of a Leading Floating Character

```
01 A PIC +$$B,$$$.  
.  
.  
/* Note that "b" represents one space */  
/* PIC String:          +$$B,$$$ */  
/* Position indexes:    12345678 */  
MOVE 1 TO A.          /* A = "+bbbb$1" */  
MOVE 12 TO A.         /* A = "+bbbb$12" */  
MOVE 123 TO A.        /* A = "+bbb$123" */  
MOVE 1234 TO A.       /* A = "+$1b,234" */
```

In this example, the \$ must be located at position 2, 5, 6, or 7.

- If the source PICTURE clause contains floating + symbols, the first non-blank character in the relevant portion of the source field must be + or -, and its location must be correct according to the rules for PICTURE clause editing.
- If the source PICTURE clause contains floating - symbols, the relevant portion of the source field must start with:
 - One or more contiguous spaces, the last of which must be correctly located according to the rules for PICTURE clause editing
 - One or more contiguous spaces, with a - immediately following it. The location of the - must be correct according to the rules for PICTURE clause editing.
 - A -.

If these rules are disobeyed, a float error occurs, and the program stops.

Performance Considerations

PICTURE Clauses for Numeric Items

Because hardware instructions use signs, you can improve performance by including an S in a picture clause whenever possible.

You can also improve performance by specifying odd numbers of numeric character positions in the picture clauses for COMP-3 (packed decimal) items. Internally, the rightmost byte of a packed decimal item contains a digit and a sign, and any other bytes contain two digits. If you use the more efficient configuration, the compiler does not need to supply the missing digit.

Eight-Byte Binary Items

Avoid using 8-byte binary items. You can specify these items for convenience, but the compiler must make conversions in order to use them.

Segmentation

Use of segmentation increases the compile and run times of the COBOL program. The segmentation feature is provided only for compatibility with other systems. You do not have to be concerned with storage management when using COBOL/400 programs.

Debugging

COBOL source language debugging is provided to help the COBOL programmer debug a program that is not functioning as expected. Use of this facility increases the compile and run times of a COBOL program.

*NORANGE Option

This GENOPT parameter option of the CRTCLPGM command removes the run-time checks for subscript and reference modification ranges.

This option can improve performance when:

- You make frequent references to tables, and the subscripts always reference elements that are in the tables
- You use reference modification often.

Note: The *RANGE option generates code for checking subscript ranges. For example, it ensures that you are not attempting to access the 21st element of a 20-element array.

The *NORANGE option does not generate code to check subscript or reference modification ranges.

These options do not eliminate the zero subscript checking performed by the operating system. If zero subscripts occur, the operating system will not permit their use and issues message MCH0603.

*DUPKEYCHK Option

This GENOPT parameter option of the CRTCLPGM command indicates that duplicate key checking for INDEXED files will be performed. Using DUPKEYCHK while reading INDEXED files can adversely affect performance.

Relative Files

You can experience lengthy delays if you open or close relative files in which very large volumes of records are being initialized to deleted records.

See Table 4 on page 247 for more information.

Indicators

If you use indicators in a separate indicator area (INDARA keyword specified in DDS) instead of in the record area, the use of the OCCURS clause to specify a table with up to 99 indicators can improve performance. See Figure 60 on page 152 for more information.

Commitment Control

Generally, the use of commitment control increases the run time of a COBOL program. In addition, the record locking that results from the use of commitment control by a job may cause delays for other users attempting to access the same file.

Reading without Record Locks

To avoid unnecessary record locks, you can include the NO LOCK phrase in your READ statement. For more information about this phrase, refer to the section on the READ statement in the *COBOL/400* Reference*.

Initializing Variables

You can reduce program run time by choosing **not** to initialize program variables that have no value clauses associated with them. You can specify no initialization by specifying *NOSTDINZ for the GENOPT parameter of the CRTCLBLPGM command, or by specifying NOSTDINZ in the PROCESS statement. The compiler then initializes only those variables that have value clauses declared. An additional benefit to this option is that you can also compile larger programs with a greater number of variables.

If you specify *NOSTDINZ, you must ensure that all data items contain valid data before you attempt to manipulate the items. If an item does not contain valid data, decimal data errors can occur.

Blocking Records

You can use record blocking to improve your run-time performance. The key benefits for blocking are realized when you read multiple records sequentially, such as a random read followed by sequential reads.

For information on blocking, refer to “Unblocking Input Records and Blocking Output Records” on page 100.

Program Loops

When a program repeatedly processes the same series of instructions, and it is apparent that this will continue indefinitely, the program is in a loop. To identify loops, you can use information known about the program itself, as follows:

- **Time:** If the actual run time is substantially exceeding the expected run time, the program could be in a loop.
- **I/O operations:** If no input/output operations are taking place and I/O is expected to be occurring repeatedly, the program is probably in a loop.

Tracing a Loop in a Program

Frequently, a loop encompasses many instructions in a program. In this case, you can use the COBOL debugging features as described in Chapter 5, “Debugging Your Program” on page 53.

Errors That Can Cause a Loop

A PERFORM statement with an UNTIL clause can cause a loop when the condition specified in the UNTIL clause cannot be met. For example:

```
PERFORM ... UNTIL COUNTR LESS THAN ZERO
```

where COUNTR is an unsigned numeric item.

A GO TO statement that refers to a previous procedure-name can cause a loop when no conditional statement exists to prevent the GO TO statement from being processed again. For example:

```
PARA-1.  
  MOVE ...  
  MOVE ...  
  MOVE ...  
PARA-2.  
  MOVE ...  
  GO TO PARA-1.
```

A possible variation of this case occurs when a conditional statement exists, but the condition cannot be met or the statement does not branch (through a GO TO statement) to a paragraph outside the range of the loop.

Chapter 12. Communicating Between Programs

Sometimes an application is simple enough to be coded as a single, self-sufficient program. In many cases, however, an application's solution will consist of several, separately compiled programs used together.

The AS/400 system provides communication between COBOL programs, and between COBOL and non-COBOL programs.

A COBOL **run unit** is a set of one or more programs that function as a unit at run time to provide a problem solution. A COBOL run unit starts with the first COBOL program in the program stack, and includes all programs (of any type) that are below it. A **program stack** is a list of programs linked together as a result of programs calling other programs, or implicitly from some other event within the same job.

When a run unit consists of several, separately compiled programs that call each other, the programs must be able to communicate with each other. They need to transfer control and usually need to have access to common data. This chapter describes the methods that accomplish this interprogram communication between separately compiled programs.

Transferring Control to Another Program

In the Procedure Division, a program can call another program (generally called a subprogram in COBOL terms), and this called program may itself call another program. The program that calls another program is referred to as the **calling** program, and the program it calls is referred to as the **called** program.

The called COBOL program starts running at the top of the Procedure Division.

When the called program processing is completed, the program can either transfer control back to the calling program or end the run unit.

A called program must not directly or indirectly execute its caller (such as program X calling program Y; program Y calling program Z; and program Z then calling program X). This is called a **recursive** call. COBOL/400 allows recursion in both main programs and subprograms. However, if you want your programs to conform to SAA standards, do not use recursive calls.

Main Programs and Subprograms

The first COBOL program to be executed begins the COBOL run unit, and is the **main program**. No specific source statements or options identify a COBOL program to be a main program or a subprogram. A **subprogram** is a program in the run unit below the main program in the program stack. For more information about program stacks and other terms concerning interprogram communication, see the *CL Programmer's Guide*.

Returning Control from a Called Program

It is important to know if a COBOL program is a main program or a subprogram to determine how control is returned from a called program when an error occurs, or a program ends.

You can issue a STOP RUN, EXIT PROGRAM, or GOBACK statement to return control from a called program.

If execution ends in the main program, either STOP RUN or GOBACK is used. These statements end the run unit, and control is returned to the caller of the main program.

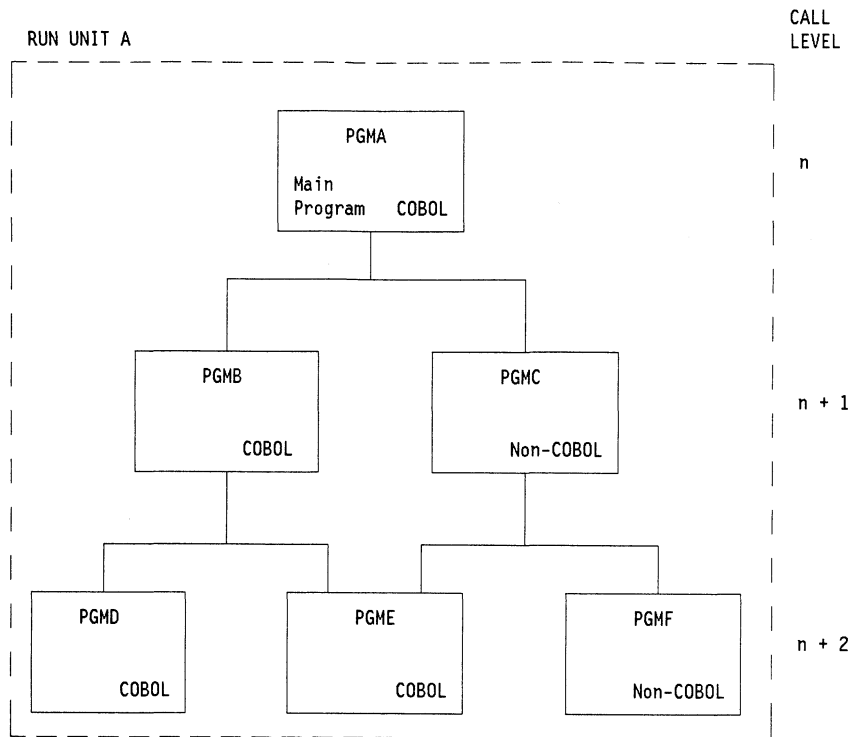
If execution ends in a subprogram, the subprogram may end with an EXIT PROGRAM, a GOBACK, or a STOP RUN statement. If the subprogram ends with an EXIT PROGRAM or a GOBACK statement, control returns to its immediate caller without ending the run unit. An implicit EXIT PROGRAM statement is generated if there is no next executable statement in a called program. If it ends with a STOP RUN statement, the effect is the same as it is in a main program: all COBOL programs in the run unit are terminated, and control returns to the caller of the main program.

A subprogram is left in its **last-used state** when it terminates with EXIT PROGRAM or GOBACK. The next time it is called in the run unit, its internal values will be as they were left, except that return values for PERFORM statements will be reset to their initial values. In contrast, a main program is initialized each time it is called.

The following examples illustrate the use of the EXIT PROGRAM and STOP RUN statements in different parts of a run unit.

- The example in Figure 85 on page 271 shows a single run unit.
- The example in Figure 86 on page 272 shows multiple run units that run consecutively
- The example in Figure 87 on page 273 shows a run unit with a shared program that is both a subprogram and a main program.
- The example in Figure 88 on page 274 shows multiple run units that run concurrently.

Note: You can substitute a GOBACK statement for an EXIT PROGRAM statement that appears in a subprogram, or a STOP RUN statement that appears in a main program.

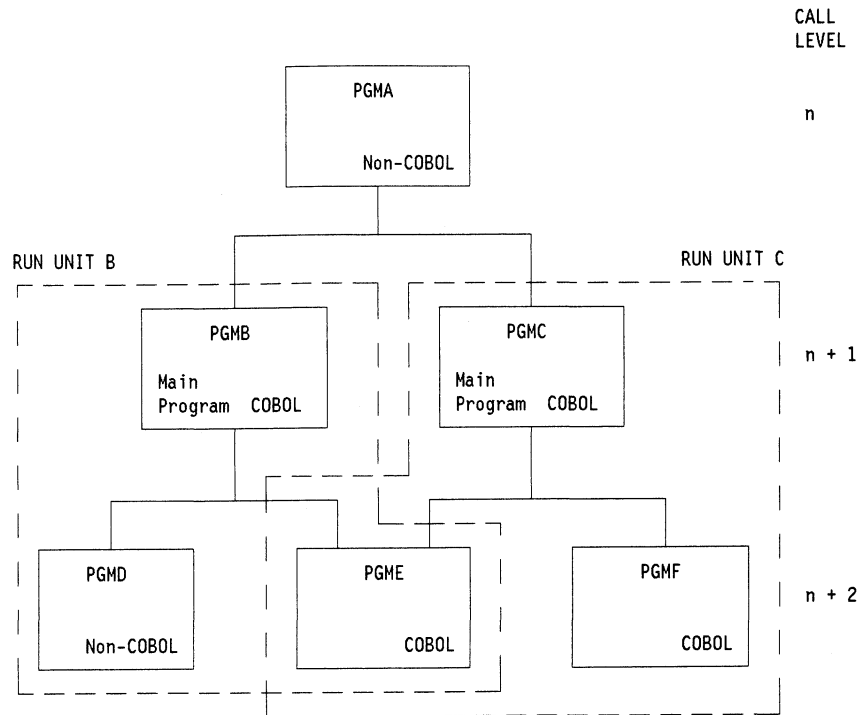


PROGRAM RUNNING STATEMENT

STATEMENT	PGMA	PGMB	PGMD	PGME
EXIT PROGRAM	1	2	2	2
STOP RUN	3	3	3	3

Figure 85. Example of a Single Run Unit

- 1** No operation is processed because the statement is processed in a main program. Processing continues with the next statement in PGMA.
- 2** Control returns to the caller of the program that processes the EXIT PROGRAM statement.
- 3** Run unit A ends. For all programs in the run unit, open files are closed. Storage is freed for all programs in the run unit. Control returns to the program that is at call level n-1. If n = 1, the following considerations apply:
 - Run unit A operates as a job step. See the *CL Programmer's Guide* for more information.
 - For batch jobs, the STOP RUN statement ends the job. For interactive jobs, control returns to the system and the system ends the job step.

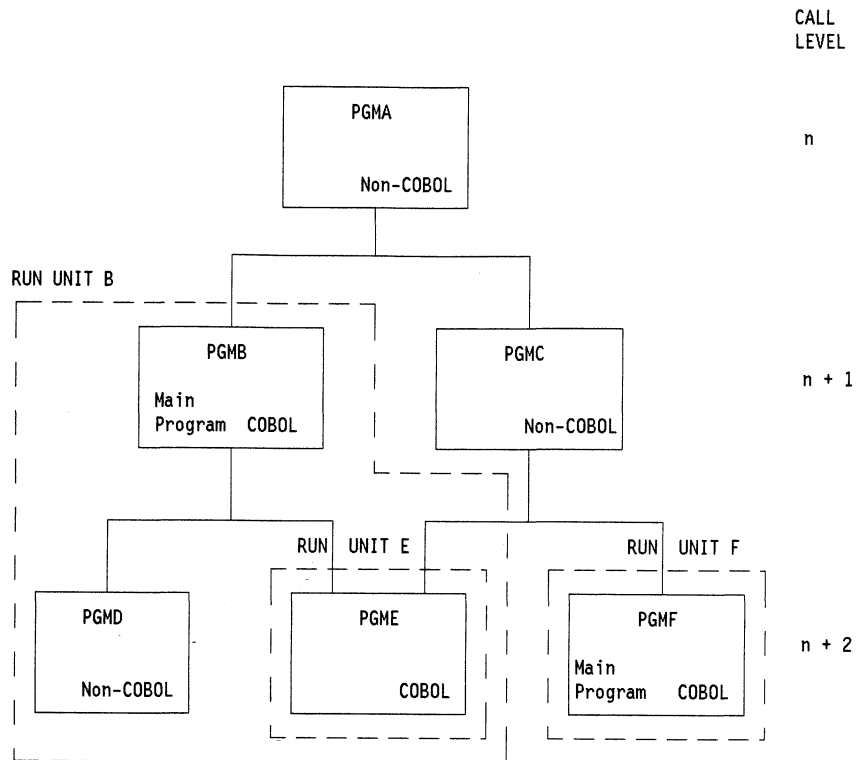


PROGRAM RUNNING STATEMENT

STATEMENT	PGMB	PGMC	PGME (RUN UNIT B)	PGME (RUN UNIT C)	PGMF
EXIT PROGRAM	1	1	2	2	2
STOP RUN	3	4	3	4	4

Figure 86. Example of Multiple Run Units That Run Consecutively

- 1** No operation is processed because the statement is processed in a main program. Processing continues with the next statement in the main program.
- 2** Control returns to the caller of the program that processes the EXIT PROGRAM statement.
- 3** Run unit B ends. All open files in run unit B are closed. Storage is freed for all programs in run unit B. Control returns to the caller of the main program for the run unit (PGMA).
- 4** Run unit C ends. All open files in run unit C are closed. Storage is freed for all programs in run unit C. Control returns to the caller of the main program for the run unit (PGMA).



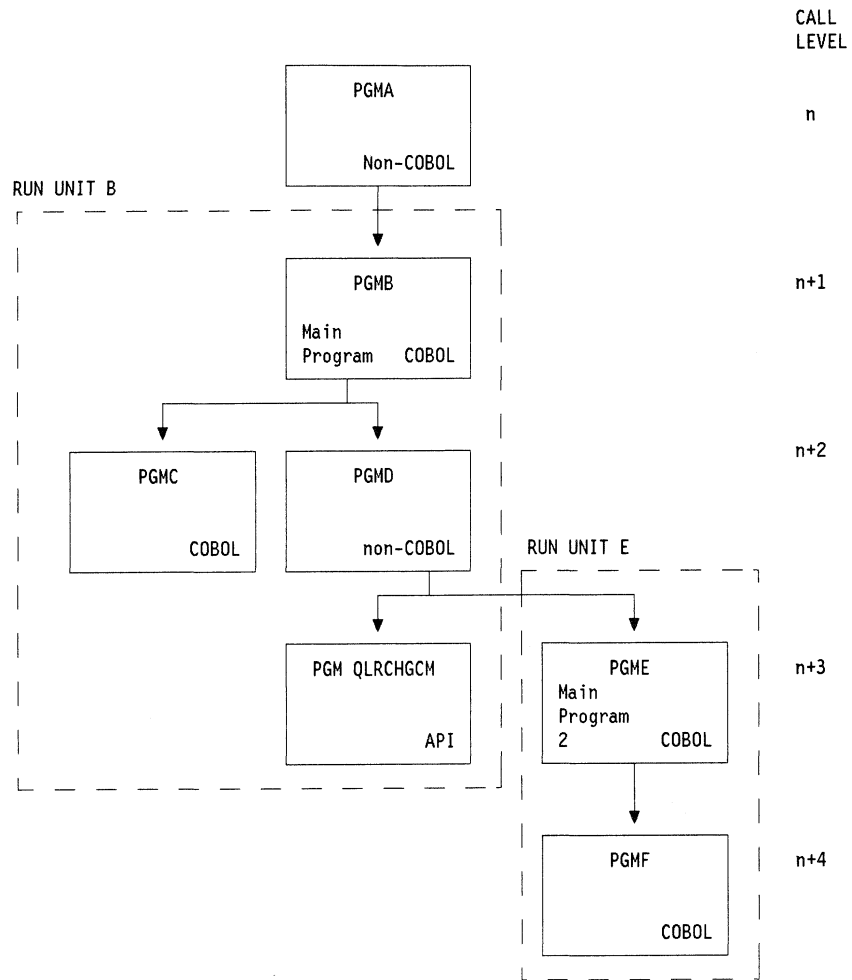
PROGRAM RUNNING STATEMENT

STATEMENT	PGMB	PGME (RUN UNIT B)	PGME (RUN UNIT E)	PGMF
EXIT PROGRAM	1	2	1	1
STOP RUN	3	3	4	5

Figure 87. Example of a Run Unit with a Shared Program that is Both a Subprogram and a Main Program

- 1** No operation is processed because the statement is processed in a main program. Processing continues with the next statement in the main program.
- 2** Control returns to the caller of the program that processes the EXIT PROGRAM statement.
- 3** Run unit B ends. All open files in run unit B are closed. Storage is freed for all programs in run unit B. Control returns to the caller of the main program for the run unit (PGMA).

- 4 Run unit E ends. All open files in run unit E are closed. Storage is freed for PGME. Control returns to the caller of the main program for the run unit (PGMC).
- 5 Run unit F ends. All open files in run unit F are closed. Storage is freed for PGMF. Control returns to the caller of the main program for the run unit (PGMC).



PROGRAM RUNNING STATEMENT

STATEMENT	PGMB	PGMC (RUN UNIT B)	PGME	PGMF (RUN UNIT E)
EXIT PROGRAM	1	2	1	2
STOP RUN	3	3	4	4

Figure 88. Example of Multiple Run Units That Run Concurrently

- 1 No operation is processed because the statement is processed in a main program. Processing continues with the next statement in the main program.

- 2 Control returns to the caller of the program that processes the EXIT PROGRAM statement.
- 3 Run unit B can only end after run unit E completes a STOP RUN. When run unit B ends, all open files in run unit B are closed. Storage is freed for all programs in run unit B, and control returns to the caller of the main program (PGMA).
- 4 Run unit E ends. All open files in run unit E are closed. Storage is freed for all programs in run unit E. Control returns to PGMD in run unit B.

Concurrent run units are achieved by using the QLRCHGCM API. Refer to the *System Programmer's Interface Reference*, SC41-8223 for more information on this API.

Initialization of Storage

The first time a COBOL program in a run unit is called, its storage is initialized. Storage is initialized again under the following conditions:

- The run unit is terminated, then reinitiated.
- The program is canceled (using the CANCEL statement for COBOL, the FREE operation for the RPG/400* programming language, or the Reclaim Resource (RCLRSC) command), and then called again.

If a non-COBOL program is named in a CANCEL statement, its name must conform to the rules for formation of a COBOL program name.

Calling Another Program

You will often want your COBOL programs to communicate with other COBOL and non-COBOL programs.

Passing Data Using **BY REFERENCE** or **BY CONTENT**

BY REFERENCE means that the subprogram is referring to and processing the data items in the calling program's storage, rather than working on a copy of the data.

BY CONTENT means that the calling program is passing only the **contents** of the *literal*, or *identifier*. With a CALL . . . BY CONTENT, the called program cannot change the value of the *literal* or *identifier* in the calling program, even if it modifies the variable in which it received the *literal* or *identifier*.

Whether you pass data items BY REFERENCE or BY CONTENT depends on what you want your program to do with the data:

- If you want the definition of the argument of the CALL statement in the calling program and the definition of the parameter in the called program to share the same memory, specify:

CALL . . . BY REFERENCE *identifier*.

Any changes made by the subprogram to the parameter affect the argument in the calling program.

An identifier in the USING phrase of the CALL . . . BY REFERENCE statement may be a file-name, in addition to a data-name.

File-names as CALL operands are allowed by the compiler as an extension.

- If you want to pass the address of a record area to a called program, specify:
CALL . . . BY REFERENCE ADDRESS OF *record-name*.

The subprogram receives the ADDRESS OF special register for the record-name you specify.

You must define the record name as a level-01 or level-77 item in the Linkage Section of the called and calling programs. A separate ADDRESS OF special register is provided for each record in the Linkage Section.

- If you do not want the definition of the argument of the CALL statement in the calling program and the definition of the parameter in the called subprogram to share the same memory, specify:

CALL . . . BY CONTENT *identifier*.

- If you want to pass a literal value to a called program, specify:

CALL . . . BY CONTENT *literal*.

The called program cannot change the value of the literal. The literal cannot be numeric.

- If you want to pass the length of a data item, specify:

CALL . . . BY CONTENT LENGTH OF *identifier*.

The calling program passes the length of *identifier* from its LENGTH OF special register. When literals are passed BY CONTENT, the called program cannot change their values.

- If you want to pass both a data item and its length to a subprogram, specify a combination of BY REFERENCE and BY CONTENT. For example:

```
CALL 'ERRPROC' USING BY REFERENCE A
                    BY CONTENT LENGTH OF A.
```

Data items in a calling program can be described in the Linkage Section of all the programs it calls directly or indirectly. In this case, storage for these items is allocated in the highest calling program. That is, program A calls program B, which calls program C. Data items in program A can be described in the Linkage Sections of programs B and C, so that one set of data can be made available to all three programs.

Describing Arguments in the Calling Program

In the calling program, the arguments are described in the Data Division in the same manner as other data items in the Data Division. Unless they are in the Linkage Section, storage is allocated for these items in the calling program. If you reference data in a file, the file must be open when the data is referenced. Code the USING clause of the CALL statement to pass the arguments.

Describing Parameters in the Called Program

In the called program, parameters are described in the Linkage Section. Code the USING clause after the PROCEDURE-DIVISION header to receive the parameters.

In the Linkage Section

You must know what is being passed from the calling program and set up the Linkage Section in the called program to accept it. To the called program, it doesn't matter which clause of the CALL statement you use to pass the data (BY REFERENCE or BY CONTENT). In either case, the called program must describe the data it is receiving. It does this in the Linkage Section.

The number of *data-names* in the *identifier* list of a called program must not be greater than the number of *data-names* in the *identifier* list of the calling program. There is a one-to-one positional correspondence; that is, the first *identifier* of the calling program is passed to the first *identifier* of the called program, and so forth. The compiler makes no attempt to match arguments and parameters.

Grouping Data to be Passed

Consider grouping all the data items you want to pass between programs and putting them under one level-01 item. If you do this, you can pass a single level-01 record between programs. For an example of this method, see Figure 89.

To make the possibility of mismatched records even smaller, put the level-01 record in a copy member, and copy it in both programs. (That is, copy it in the Working-Storage Section of the calling program and in the Linkage Section of the called program.)

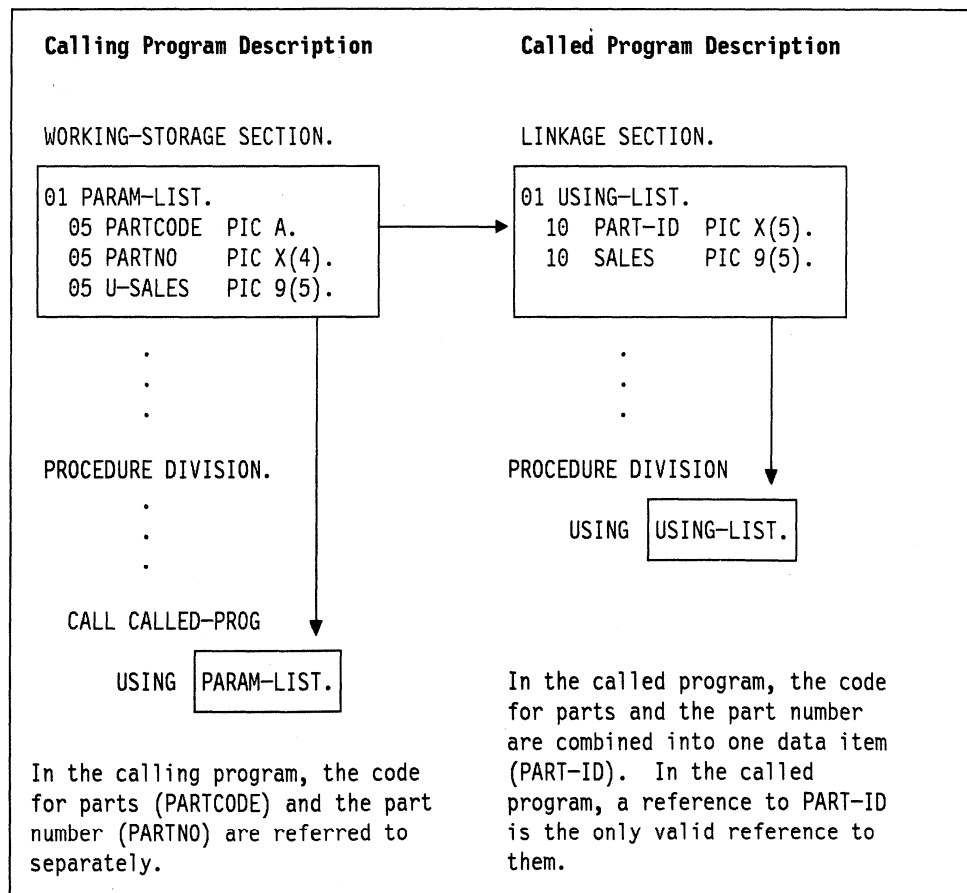


Figure 89. Common Data Items in Subprogram Linkage

Call by Identifier

A system pointer that associates an identifier with an object is set the first time you use the identifier in a CALL statement.

Important for compatibility!

If you carry out a call by an identifier to a program that you subsequently delete or rename, you must use the CANCEL statement to null the system pointer associated with the identifier. This ensures that when you next use the identifier to call your program, the associated system pointer will be set again.

The following example shows how to apply the CANCEL statement to an identifier:

```
MOVE "ABCD" TO IDENT-1.  
CALL IDENT-1.  
CANCEL IDENT-1.
```

If you apply the CANCEL statement directly to the literal "ABCD", you do *not* null the system pointer associated with IDENT-1. Instead, you can continue to call program ABCD simply by using IDENT-1 in your CALL statement.

The value of the system pointer also changes if you change the value of the identifier and perform a call using this new value.

Using Pointers in a COBOL/400 Program

You can use a **pointer** (a data item in which address values can be stored) within a COBOL program when you want to pass and receive addresses of a variably-located data item, and to accomplish limited base addressing.

On the AS/400 system, pointers are 16 bytes long. COBOL pointers are AS/400 **space pointers** since they point to system space objects. One part of the pointer describes its attributes, such as which AS/400 space object it is pointing to. Another part of the pointer contains the offset into the AS/400 system space object.

To define a COBOL pointer, called a **pointer data item**, code a USAGE IS POINTER clause on the data item. A pointer data item is a 16-byte elementary item that can be compared for equality, or used to set the value of other pointer items.

A pointer data item can be used only in:

- A SET statement (Format 5 only)
- A relation condition
- The USING phrase of a CALL statement, or the Procedure Division header.
- The operand for the LENGTH OF and ADDRESS OF special registers.

If pointers are used in a relational condition, the only valid operators are equal to, or not equal to.

Because pointer data items are not simply binary numbers on the AS/400 system, manipulating pointers as integers does not work.

Pointer data items are defined explicitly with the USAGE IS POINTER clause, and are implicit when using an ADDRESS OF special register or the ADDRESS OF an item.

If a group item is described with the USAGE IS POINTER clause, the elementary items within the group item are pointer items. The group itself is not a pointer data item, and cannot be used in the syntax where a pointer data item is allowed. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

Pointer data items can be part of a group that is referred to in a MOVE statement or an input/output statement; however, if a pointer data item is part of a group, there is no conversion of pointer values to another form of internal representation when the statement is executed.

Defining Pointers and Pointer Alignment

Pointer data items can be defined at any level (except 88) in the File, Working-Storage, or Linkage sections of a program.

When a pointer is referenced on the AS/400 system, it must be on a 16-byte storage boundary. **Pointer alignment** refers to the COBOL/400 compiler's process of positioning pointer items within a group item to offsets that are multiples of 16 bytes from the beginning of the record. If a pointer item is not on a 16-byte boundary, a pointer alignment exception (MCH0602) is sent to the COBOL/400 program. In general, pointer alignment exceptions occur in the Linkage Section, where it is up to the user to align these items.

In the File and Working-Storage sections, the compiler ensures that this exception does not occur by adding implicit FILLER items. Every time an implicit FILLER item is added by the compiler, a warning is issued. In the Linkage Section, no implicit FILLER items are added by the compiler; however, warnings are issued indicating how many bytes of FILLER would have been added had the group item appeared in the File or Working-Storage sections.

You can define a data item as a pointer by specifying the USAGE IS POINTER clause as shown in the following example:

```
WORKING-STORAGE SECTION.  
  77 APTR USAGE POINTER.  
  01 AB.  
     05 BPTR  USAGE POINTER.  
     05 BVAR  PIC S9(3) PACKED-DECIMAL.  
LINKAGE SECTION.  
  01 AVAR.  
     05 CVAR  PIC X(30).  
PROCEDURE DIVISION.  
  SET APTR TO ADDRESS OF AVAR.
```

Figure 90. Defining a Pointer Data Item

In the above example, AVAR is an 01-level data item, so the ADDRESS OF AVAR is the ADDRESS OF special register. Because a special register is an actual storage area, the SET statement moves the contents of ADDRESS OF AVAR into pointer data item APTR.

In the above example, if the SET statement used ADDRESS OF CVAR, no special register exists. Instead, the pointer data item APTR is assigned the calculated address of CVAR.

In File and Working-Storage Sections

In the File and Working-Storage sections, all 01-level items (and some 66 and 77-level items) are placed on 16-byte boundaries.

Within a group structure, pointer data items must also occur on a 16-byte boundary. To ensure this, the COBOL/400 compiler adds FILLER items immediately before the pointer data item. To avoid these FILLER items, you should place pointer data items at the beginning of a group item.

If the pointer data item is part of a table, the first item in the table is placed on a 16-byte boundary. To ensure that all subsequent occurrences of the pointer fall on a 16-byte boundary, a FILLER item is added to the end of the table if necessary.

An example of pointer data item alignment follows:

```
WORKING-STORAGE SECTION.  
  77 APTR USAGE POINTER.  
  01 AB.  
    05 ALPHA-NUM PIC X(10).  
    05 BPTR  USAGE POINTER.  
  01 EF.  
    05 ARRAY-1 OCCURS 3 TIMES.  
      10 ALPHA-NUM-TWO PIC X(14).  
      10 CPTR  USAGE POINTER.  
      10 ALPHA-NUM-THREE PIC X(5).
```

Figure 91. Aligning Pointer Data Items

In the above example, APTR is a pointer data item. The 77-level item, therefore, is placed on a 16-byte boundary. The group item AB is an 01-level item and is automatically placed on a 16-byte boundary. Within the group item AB, BPTR is not on a 16-byte boundary. To align it properly, the compiler inserts a 6-byte FILLER item after ALPHA-NUM. Finally, CPTR requires a FILLER of 2 bytes to align its first occurrence. Because ALPHA-NUM-THREE is only 5 bytes long, another 11-byte FILLER must be added to the end of ARRAY-1 to align all subsequent occurrences of CPTR.

When a pointer is defined in the File Section, and a file does not have blocking in effect, each 01-level item will be on a 16-byte boundary. If a file has blocking in effect, only the first record of a block is guaranteed to be on a 16-byte boundary. Thus pointer data items should not be defined for files with blocking in effect. For more information on blocking, refer to "Unblocking Input Records and Blocking Output Records" on page 100.

Pointers and the REDEFINES Clause

A pointer data item may be the subject or object of a REDEFINES clause.

When a pointer is the subject of a REDEFINES clause, the object data item must be on a 16-byte boundary. For example:

```

WORKING-STORAGE SECTION.
01 AB.
   05 ALPHA-NUM PIC X(16).
   05 APTR REDEFINES ALPHA-NUM USAGE POINTER.
   05 BPTR  USAGE POINTER.
   05 CPTR REDEFINES BPTR USAGE POINTER.

```

Figure 92. REDEFINES and Aligned Pointer Data Items

In the above example, both APTR and CPTR are pointer data items that redefine 16-byte aligned items. In the following example, the redefined item would result in a severe compiler error:

```

WORKING-STORAGE SECTION.
01 EF.
   05 ALPHA-NUM PIC X(5).
   05 HI.
      10 ALPHA-NUM-TWO PIC X(11).
      10 APTR  USAGE POINTER.
   05 BPTR REDEFINES HI USAGE POINTER.

```

Figure 93. REDEFINES and Aligned Pointer Data Items - Incorrect Method

In the above example, APTR is aligned on a 16-byte boundary. That is, the COBOL/400 compiler did not need to add FILLER items to align APTR. The group item HI is not on a 16-byte boundary, and so neither is pointer data item BPTR. Because the COBOL/400 compiler cannot add FILLER items to place BPTR on a 16-byte boundary, a severe error will result. In the following example, similar to the above, the COBOL/400 compiler is able to place the pointer data item on a 16-byte boundary:

```

WORKING-STORAGE SECTION.
01 EF.
   05 ALPHA-NUM PIC X(5).
   05 HI.
      10 ALPHA-NUM-TWO PIC X(11).
      10 APTR  USAGE POINTER.
      10 ALPHA-NUM-THREE PIC X(5).
   05 KL REDEFINES HI.
      10 BPTR USAGE POINTER.

```

Figure 94. REDEFINES and Unaligned Pointer Data Items - Correct Method

In the above example, group item KL is not on a 16-byte boundary; however, the compiler adds an 11-byte FILLER before pointer data item BPTR to ensure that it falls on a 16-byte boundary.

Reading and Writing Pointers

Pointer data items can be defined in the File Section, and can be set and used as can any other Working-Storage pointer data items. There are, however, some restrictions:

- If a file has blocking in effect, only the first record of a block is guaranteed to be on a 16-byte boundary. Thus pointer data items should not be defined for files with blocking in effect.
- A record containing pointers can be written to a file; however, on subsequent reading of that record, the pointer data items equal NULL.

Initializing Pointers Using the NULL Figurative Constant

The NULL figurative constant represents a value used to indicate that data items defined with USAGE IS POINTER, ADDRESS OF, or the ADDRESS OF special register do not contain a valid address. For example:

```
WORKING-STORAGE SECTION.  
  77 APTR  USAGE POINTER VALUE NULL.  
PROCEDURE DIVISION.  
  IF APTR = NULL THEN  
    DISPLAY 'APTR IS NULL'  
  END-IF.
```

Figure 95. Using NULL to Initialize a Pointer

In the above example, pointer APTR is set to NULL in the Working-Storage section. The comparison in the procedure division will be true and the display statement is executed.

On the AS/400 system, the initial value of a pointer data item with or without a VALUE clause of NULL, equals NULL.

LENGTH OF Special Register

The LENGTH OF special register contains the number of bytes used by an identifier. It returns a value of 16 for a pointer data item.

You can use LENGTH OF in the Procedure Division anywhere a numeric data item having the same definition as the implied definition of the LENGTH OF special register is used; however, LENGTH OF cannot be used as a subscript or a receiving data item. LENGTH OF has the implicit definition:

```
USAGE IS BINARY, PICTURE 9(9)
```

The following example shows how you can use LENGTH OF with pointers:

```
WORKING-STORAGE SECTION.  
  77 APTR  USAGE POINTER.  
  01 AB.  
    05 BPTR  USAGE POINTER.  
    05 BVAR  PIC S9(3) PACKED-DECIMAL.  
    05 CVAR  PIC S9(3) PACKED-DECIMAL.  
PROCEDURE DIVISION.  
  MOVE LENGTH OF AB TO BVAR.  
  MOVE LENGTH OF BPTR TO CVAR.
```

Figure 96. Using LENGTH OF with Pointers

In the above example, the length of group item AB is moved to variable BVAR. BVAR has a value of 20 because BPTR is 16 bytes long, and both variables BVAR and CVAR are 2 bytes long. CVAR receives a value of 16.

You can also use the LENGTH OF special register to set up data structures within user spaces, or to increment addresses received from another program. To see an example of a program that uses the LENGTH OF special register to define data structures within user spaces, refer to Figure 99 on page 287.

Setting the Address of Linkage Items

Generally, when one COBOL program calls another, data passes between the two programs in the following manner: the calling program uses the CALL USING statement to pass operands to the called program, and the called program specifies the USING phrase in the Procedure Division header. There should be a one-to-one mapping between the operands in the USING phrases of each program.

When using the ADDRESS OF special register, you no longer need to ensure a one-to-one mapping between the USING phrases of the two programs. For those data items in the Linkage Section that are not specified in the USING phrase of the Procedure Division header, you can use a SET statement to specify the starting address of the data structure. Once the SET statement is run, the data item is then treated as if it was passed from another program. For an example of a SET statement used in this manner, refer to Figure 100 on page 288.

16 on page 291 illustrates how the SET statement is used to set the starting address of the data structures *ls-header-record* and *ls-user-space* at the beginning of the user space.

Using ADDRESS OF and the ADDRESS OF Special Register

When you specify ADDRESS OF in a COBOL program, the compiler determines whether to use the calculated address of a data item, referred to as ADDRESS OF, or the ADDRESS OF special register. The ADDRESS OF special register is the starting address of the data structure from which all calculated addresses are determined. Because the ADDRESS OF special register is the starting address of a structure, it must be an 01-level or 77-level data item. If you reference modify this data item, it is no longer the starting address of the data structure. It is a calculated address, or ADDRESS OF. If you are taking the ADDRESS OF an elementary item, and the ADDRESS OF the 01-level item has been set to NULL, a pointer exception (MCH3601) results.

You cannot use the calculated ADDRESS OF where an item can be changed. Only the ADDRESS OF special register can be changed. For example, in Figure 100, the SET statement at **18** on page 291 uses the ADDRESS OF special register because it is an 01-level item. At **19** on page 291 ADDRESS OF is used because, although it is an 01-level item, it is reference-modified.

Using Pointers in a MOVE Statement

Elementary pointer data items cannot be moved using the MOVE statement; a SET statement must be used; however, pointer data items are implicitly moved when they are part of a group item.

When compiling a MOVE statement, the COBOL/400 compiler generates code to maintain (a pointer MOVE) or not maintain (a non-pointer MOVE) pointers within a group item.

A pointer MOVE is done when all of the following conditions are met:

1. The source or receiver of a MOVE statement contains a pointer
2. Both of the items are at least 16 bytes long
3. The data items are properly aligned

4. The data items are alphanumeric or group items.

Of the conditions listed above, determining if two data items are properly aligned can be the most difficult.

If the items being moved are 01-level items, or are part of an 01-level item, they must be on the same offset relative to a 16-byte boundary for a pointer MOVE to occur. (A warning is issued if this is not true.) The following example shows three data structures, and the results when a MOVE statement is issued:

WORKING-STORAGE SECTION.

```
01 A.  
   05 B      PIC X(10).  
   05 C.  
       10 D      PIC X(6).  
       10 E      POINTER.  
  
01 A2.  
   05 B2     PIC X(6).  
   05 C2.  
       10 D2     PIC X(10).  
       10 E2     POINTER.  
  
01 A3.  
   05 B3     PIC X(22).  
   05 C3.  
       10 D3     PIC X(10).  
       10 E3     POINTER.
```

PROCEDURE DIVISION.

```
MOVE A to A2. 1  
MOVE A to A3. 1  
MOVE C to C2. 2  
MOVE C2 to C3. 3
```

- 1** This results in a pointer move because the offset of each group item to be moved is zero. Pointer integrity is maintained.
- 2** This results in a non-pointer move, because the offsets do not match. The offset of group item C is 10, and the offset of group item C2 is 6. Pointer integrity is not maintained.
- 3** This results in a pointer move, because the offset of group item C2 is 6, and the offset of C3 relative to a 16-byte boundary is also 6. (When the offset is greater than 16, the offset relative to a 16-byte boundary is calculated by dividing the offset by 16. The remainder is the relative offset. In this case, the offset was 22, which, when divided by 16, leaves a remainder, or relative offset, of 6.) Pointer integrity is maintained.

If a group item contains a pointer, and the compiler cannot determine the offset relative to a 16-byte boundary, the compiler issues a warning message, and the pointer move is attempted. However, pointer integrity may not be maintained. The compiler cannot determine the offset if the item is defined in the Linkage Section, or if the item is reference-modified with an unknown starting position. You must ensure that pointer alignment is maintained, or MCH0602 may result.

The COBOL/400 compiler places all 01-level items on a 16-byte boundary whether or not they contain pointer data items.

If one of the items in a MOVE statement is an 01-level item with a pointer, and the other a 77-level Working-Storage item, the 77-level Working-Storage item is forced to a 16-byte boundary.

Using Pointers in a CALL Statement

When a pointer data item is passed in a CALL statement, the item is treated as all other USING items. In other words, a pointer to the pointer data item (or copy of the pointer data item) is passed to the called program.

Special consideration must be given when a CALL statement with the BY CONTENT phrase is used to pass pointers and group items containing pointers. This is similar to the case of a MOVE statement. For a CALL BY CONTENT, an implicit MOVE of an item is done to create it in a temporary area. If the compiler can determine the offset of an item relative to a 16-byte boundary, that same offset is used when the implicit MOVE of the BY CONTENT item is done into the temporary area. When the compiler cannot determine the offset of an item relative to a 16-byte boundary, the implicit MOVE of the BY CONTENT item is done into a temporary area that is aligned on a 16-byte boundary.

The compiler is not able to determine the offset of an item relative to a 16-byte boundary when the BY CONTENT item is:

- Reference modified with an unknown starting position, or
- Defined in the Linkage Section.

When an operand is reference = modified, the offset is the reference modification starting position minus one, plus the operand's offset within the data structure. When an operand is in the Linkage Section, its offset can be determined from the calling program.

To avoid pointer alignment problems, pass items by reference.

The following is an example of passing items containing pointers, where pointer integrity is maintained in some cases, and not in others.

WORKING-STORAGE SECTION.

```
01 A. 1
      05 B    PIC X(3).
      05 C. 2
          10 FILLER  PIC X(13).
          10 D      POINTER.
```

PROCEDURE DIVISION.

CALL "B" USING A C.

Figure 97. Program A -- Main Program

WORKING-STORAGE SECTION.

```
01 E.  
   05 F    PIC X(16).  
   05 G    POINTER.  
77 K    PIC S9(3)    VALUE 8.
```

LINKAGE SECTION.

```
01 A.  3  
   05 B    PIC X(3).  
   05 C.  
       10 FILLER    PIC X(13).  
       10 D    POINTER.  
01 C2. 4  
   05 FILLER    PIC X(13).  
   05 D2    POINTER.
```

PROCEDURE DIVISION USING A C2.

```
CALL "C" USING BY CONTENT  
    A, C2, 5 E(5: ), 6 E(K: ), 7 F. 8
```

Figure 98. Program B -- Subprogram

In the previous example, Program A passes two group items to Program B. **1** is an 01-level group item, with an offset of zero. **2** is an 05-level group item, and has an offset of 3. Because the items are passed by reference, pointer integrity is maintained for both group items A and C.

Program B passes five items to another program, C. The items are passed by content to Program C. Because they are passed by content, Program C receives a copy of the items, and pointer integrity is not maintained in all cases.

- **3** Because this item is defined in the Linkage Section, it has an unknown offset. The compiler assumes it is 16-byte aligned, and in this case, when A is passed, pointer integrity of D is maintained, but a compiler warning message is issued on the CALL.
- **4** This item contains a pointer, and a pointer move is accomplished by **5**. However, because the item is defined in the Linkage Section and the offset is unknown, pointer integrity is not maintained. The compiler attempts to move C2 to a 16-byte aligned area, and a compiler warning message is issued.
- **6** Because E contains a pointer, a pointer move is accomplished. The offset can be calculated because the reference modification start position is a numeric literal. In this case, pointer integrity is maintained, and the item is placed at an offset of 4 from the 16-byte boundary.
- **7** Because E contains a pointer, a pointer move is attempted. Because E is reference-modified with an unknown starting position (K), the compiler cannot calculate the offset, and assumes it is aligned on a 16-byte boundary. A compiler warning message is issued. If the value of K causes E to be aligned on a 16-byte boundary, pointer integrity is maintained. For this to occur, K must be 1 or 17.
- **8** F is an item defined in the Working-Storage Section, and contains no pointers, so no pointer moves are expected.

Using Pointers and APIs to Access User Spaces

The following example shows how you can use pointers to access user spaces and to chain records together.

POINTA is a program that reads customer names and addresses into a user space, and then displays the information in a list. The program assumes that the customer information exists in a file called POINTACU.

The customer address field is a variable-length field, to allow for lengthy addresses.

```
A* THIS IS THE CUSTOMER INFORMATION FILE - POINTACUST
A
A
A      R FSCUST          TEXT('CUSTOMER MASTER RECORD')
A      FS_CUST_NO      8S00    TEXT('CUSTOMER NUMBER')
A                                  ALIAS(FS_CUST_NUMBER)
A      FS_CUST_NM      20      TEXT('CUSTOMER NAME')
A                                  ALIAS(FS_CUST_NAME)
A      FS_CUST_AD      100     TEXT('CUSTOMER ADDRESS')
A                                  ALIAS(FS_CUST_ADDRESS)
A                                  VARLEN
A
```

Figure 99. Example Using Pointers to Access User Spaces -- DDS

```

Program . . . . . : POINTA
Library . . . . . : TESTER
Source file . . . . . : QLBSRC
Library . . . . . : TESTER
Source member . . . . . : POINTA 05/01/92 17:55:27
Generation severity level . . . . . : 29
Text 'description' . . . . . : *BLANK
Source listing options . . . . . : *NONE
Generation options . . . . . : *NONE
Conversion options . . . . . : *NONE
Message limit:
  Number of messages . . . . . : *NOMAX
  Message limit severity . . . . . : 29
Print file . . . . . : QSYSVRT
Library . . . . . : *LIBL
FIPS flagging . . . . . : *NOFIPS *NOSEG *NODEB *NOBSOLETE
SAA flagging . . . . . : *NOFLAG
Extended display options . . . . . :
Flagging severity . . . . . : 0
Replace program . . . . . : *YES
Target release . . . . . : *CURRENT
User profile . . . . . : *USER
Authority . . . . . : *LIBCRTAUT
Compiler . . . . . : IBM SAA COBOL/400

```

```

STMT SEQNBR -A 1 B.+...2....+...3....+...4....+...5....+...6....+...7..IDENTFCN S COPYNAME CHG DATE

```

```

1 000010 PROCESS extaccdsp varchar 2
2 000020 ID DIVISION. CBT00010

```

```

000040* This program reads in a file of variable length records
000050* into a user space. It then shows the records on
000060* the display.
3 000070 PROGRAM-ID. pointa.
4 000080 ENVIRONMENT DIVISION.
5 000090 CONFIGURATION SECTION.
6 000100 SPECIAL-NAMES. CONSOLE IS CRT,
7 000110 CRT STATUS IS ws-crt-status. 3
8 000120 INPUT-OUTPUT SECTION.
9 000130 FILE-CONTROL.
10 000140 SELECT cust-file ASSIGN TO DATABASE-pointacu
11 000150 ORGANIZATION IS SEQUENTIAL
12 000160 FILE STATUS IS ws-file-status.
13 000170 DATA DIVISION.
14 000180 FILE SECTION.
15 000190 FD cust-file.
16 000200 01 fs-cust-record.
000210* copy in field names turning underscores to dashes
000220* and using alias names
17 000230 COPY DDR-ALL-FORMATS-1 OF pointacu.
18 +000001 05 POINTACU-RECORD PIC X(130). <-ALL-FMTS
+000002* I-O FORMAT:FSCUST FROM FILE POINTACU OF LIBRARY TESTER <-ALL-FMTS
+000003* CUSTOMER MASTER RECORD <-ALL-FMTS
19 +000004 05 FSCUST REDEFINES POINTACU-RECORD. <-ALL-FMTS
20 +000005 06 FS-CUST-NUMBER PIC S9(8). <-ALL-FMTS
+000006* CUSTOMER NUMBER <-ALL-FMTS
21 +000007 06 FS-CUST-NAME PIC X(20). <-ALL-FMTS
+000008* CUSTOMER NAME <-ALL-FMTS
22 +000009 06 FS-CUST-ADDRESS. 4 <-ALL-FMTS
+000010* (Variable length field) <-ALL-FMTS
23 +000011 49 FS-CUST-ADDRESS-LENGTH <-ALL-FMTS
24 +000012 PIC S9(4) COMP-4. <-ALL-FMTS
25 +000013 49 FS-CUST-ADDRESS-DATA <-ALL-FMTS
26 +000014 PIC X(100). <-ALL-FMTS
+000015* CUSTOMER ADDRESS <-ALL-FMTS
27 000240 WORKING-STORAGE SECTION.
28 000250 01 ws-file-status.
29 000260 05 ws-file-status-1 PIC X.
30 000270 88 ws-file-stat-good VALUE "0".
31 000280 88 ws-file-stat-at-end VALUE "1".
32 000290 05 ws-file-status-2 PIC X.
33 000300 01 ws-crt-status. 5
34 000310 05 ws-status-1 PIC 9(2).
35 000320 88 ws-status-1-ok VALUE 0.
36 000330 88 ws-status-1-func-key VALUE 1.

```

Figure 100 (Part 1 of 7). Example Using Pointers to Access User Spaces

```

Customer Information Display
5738CB1 V2R2M0 001000 AS/400 COBOL Source TESTER/POINTA AS400SYS 05/01/92 18:01:14 Page 3
STMT SEQNBR -A 1 B. ....2.....3.....4.....5.....6.....7..IDENTFCN S COPYNAME CHG DATE
37 000340 88 ws-status-1-error VALUE 9.
38 000350 05 ws-status-2 PIC 9(2).
39 000360 88 ws-func-03 VALUE 3.
40 000370 88 ws-func-07 VALUE 7.
41 000380 88 ws-func-08 VALUE 8.
42 000390 05 ws-status-3 PIC 9(2).
43 000400 01 ws-params. 6
44 000410 05 ws-space.
45 000420 10 ws-space-name PIC X(10) VALUE "MYSPACE".
46 000430 10 ws-space-lib PIC X(10) VALUE "QTEMP".
47 000440 05 ws-attr PIC X(10) VALUE "PF".
48 000450 05 ws-init-size PIC S9(5) VALUE 32000 BINARY.
49 000460 05 ws-init-char PIC X VALUE SPACE.
50 000470 05 ws-auth PIC X(10) VALUE "**ALL".
51 000480 05 ws-text PIC X(50) VALUE
52 000490 "Customer Information Records".
53 000500 05 ws-replace PIC X(10) VALUE "**YES".
54 000510 05 ws-err-data. 7
55 000520 10 ws-input-1 PIC S9(6) BINARY VALUE ZERO.
56 000530 10 ws-output-1 PIC S9(6) BINARY VALUE ZERO.
57 000540 10 ws-exception-id PIC X(7).
58 000550 10 ws-reserved PIC X(1).
59 000560 10 ws-exception-data PIC X(87).
60 000570 05 ws-space-ptr POINTER. 8
61 000580 05 ws-map-ptr POINTER.
000590
62 000600 77 ws-accept-data PIC X.
63 000610 88 ws-acc-create-space VALUE "Y", "y".
64 000620 88 ws-acc-delete-space VALUE "Y", "y".
65 000630 88 ws-acc-no-space VALUE "N", "n".
000640
66 000650 77 ws-prog-indicator PIC X VALUE "G".
67 000660 88 ws-prog-continue VALUE "G".
68 000670 88 ws-prog-end VALUE "C".
69 000680 88 ws-prog-loop VALUE "L".
000690
70 000700 77 ws-line PIC S99.
000710* error message line
71 000720 77 ws-error-msg PIC X(50) VALUE SPACES.
000730* more address information indicator
72 000740 77 ws-plus PIC X.
000750* length of address information to display
73 000760 77 ws-temp-size PIC 9(2).
000770
74 000780 77 ws-current-rec PIC S9(4) VALUE 1.
75 000790 77 ws-old-rec PIC S9(4) VALUE 1.
76 000800 77 ws-old-space-ptr POINTER.
000810* max number of lines to display
77 000820 77 ws-displayed-lines PIC S99 VALUE 20.
000830* line on which to start displaying records
78 000840 77 ws-start-line PIC S99 VALUE 5.
000850* variables to create new record in space
79 000860 77 ws-addr-inc PIC S9(4) PACKED-DECIMAL.
80 000870 77 ws-temp PIC S9(4) PACKED-DECIMAL.
81 000880 77 ws-temp-2 PIC S9(4) PACKED-DECIMAL.
000890* pointer to previous record
82 000900 77 ws-cust-prev-ptr POINTER VALUE NULL.
83 000910 LINKAGE SECTION.
84 000920 01 ls-header-record. 9
85 000930 05 ls-hdr-cust-ptr USAGE POINTER.
000940* number of records read in from file
86 000950 05 ls-record-counter PIC S9(3) BINARY.
87 000960 05 FILLER PIC X(14). 10
88 000970 01 ls-user-space. 11
89 000980 05 ls-customer-rec.
000990* pointer to previous customer record
90 001000 10 ls-cust-prev-ptr USAGE POINTER.
91 001010 10 ls-cust-rec-length PIC S9(4) BINARY.
92 001020 10 ls-cust-name PIC X(20).
93 001030 10 ls-cust-number PIC S9(8).

```

Figure 100 (Part 2 of 7). Example Using Pointers to Access User Spaces

```

Customer Information Display
5738CB1 V2R2M0 001000 AS/400 COBOL Source TESTER/POINTA AS400SYS 05/01/92 18:01:14 Page 4
STMT SEQNBR -A 1 B...+....2.....+....3.....+....4.....+....5.....+....6.....+....7..IDENTFCN S COPYNAME CHG DATE
001040* total length of this record including filler bytes
001050* to make sure next record on 16 byte boundary
94 001060 10 ls-cust-address-length PIC S9(4) BINARY.
95 001070 05 ls-cust-address-data PIC X(116).
001080
001090* Size of ls-user-space is 16 more than actually needed. This
001100* allows the start address of the next record
001110* record to be established without exceeding the declared size
001120* The size is 16 bigger to allow for pointer alignment
001130
96 001140 PROCEDURE DIVISION.
001150* note no need for "USING" entry on PROC... DIV.
001160 DECLARATIVES.
001170 cust-file-para SECTION.
001180 USE AFTER ERROR PROCEDURE ON cust-file.
001190 cust-file-para-2.
97 001200 MOVE "Error XX on file pointacu" TO ws-error-msg.
98 001210 MOVE ws-file-status TO ws-error-msg(7:2).
001220 END DECLARATIVES.
001230 main-section section.
001240 main-proc.
001250* keep reading initial display until entered data correct
99 001260 SET ws-prog-loop to TRUE.
100 001270 PERFORM initial-display THRU read-initial-display
001280 UNTIL NOT ws-prog-loop.
001290* if want to continue with program and want to create
001300* customer information area, fill the space with
001310* records from the customer file
101 001320 IF ws-prog-continue and
001330 ws-acc-create-space THEN
102 001340 PERFORM read-customer-file
103 001350 MOVE 1 TO ws-current-rec
001360* set ptr to header record
104 001370 SET ADDRESS OF ls-header-record TO ws-space-ptr
001380* set to first customer record in space
105 001390 SET ADDRESS OF ls-user-space TO ls-hdr-cust-ptr
001400 END-IF.
106 001410 IF ws-prog-continue THEN
107 001420 PERFORM main-loop UNTIL ws-prog-end
001430 END-IF.
001440 end-program.
108 001450 PERFORM clean-up.
109 001460 STOP RUN.
001470 initial-display. 12
110 001480 DISPLAY "Create Customer Information Area" AT 0118 WITH
001490 BLANK SCREEN REVERSE-VIDEO
001500 "Create customer information area (Y/N)=> <="
001510 AT 1015
001520 "F3=Exit" AT 2202.
111 001530 IF ws-error-msg NOT = SPACES THEN
112 001540 DISPLAY ws-error-msg at 2302 with beep highlight
113 001550 MOVE SPACES TO ws-error-msg
001560 END-IF.
001570 read-initial-display. 13
114 001580 ACCEPT ws-accept-data AT 1056 WITH REVERSE-VIDEO
001590 ON EXCEPTION
115 001600 IF ws-status-1-func-key THEN
116 001610 IF ws-func-03 THEN
117 001620 SET ws-prog-end TO TRUE
001630 ELSE
118 001640 MOVE "Invalid Function Key" TO ws-error-msg
001650 END-IF
001660 ELSE
119 001670 MOVE "Unknown Error" TO ws-error-msg
001680 END-IF
001690 NOT ON EXCEPTION

```

Figure 100 (Part 3 of 7). Example Using Pointers to Access User Spaces

```

Customer Information Display
5738CB1 V2R2M0 001000 AS/400 COBOL Source TESTER/POINTA AS400SYS 05/01/92 18:01:14 Page 5
120 001700 IF ws-acc-create-space THEN
121 001710 PERFORM create-space THRU get-space
122 001720 SET ws-prog-continue TO TRUE
001730 ELSE
123 001740 IF NOT ws-acc-no-space THEN
124 001750 MOVE "Invalid Character Entered" TO ws-error-msg
001760 ELSE
125 001770 SET ws-prog-continue TO TRUE
126 001780 PERFORM get-space
001790 END-IF
001800 END-IF
001810 END-ACCEPT.
001820 create-space.
127 001830 CALL "QUSCRTUS" 14
001840 USING ws-space, ws-attr, ws-init-size,
001850 ws-init-char, ws-auth, ws-text,
001860 ws-replace, ws-err-data.
001870* check for errors in creating space
001880 get-space.
128 001890 CALL "QUSPTRUS" USING ws-space, ws-space-ptr. 15
001900* set header record to beginning of space
129 001910 SET ADDRESS OF ls-header-record 16
001920 ADDRESS OF ls-user-space 17
001930 TO ws-space-ptr.
001940* set first customer record after header record
130 001950 SET ADDRESS OF ls-user-space TO 18
001960 ADDRESS OF ls-user-space(LENGTH OF ls-header-record 19
001970 + 1:1).
001980* save ptr to first record in header record
131 001990 SET ls-hdr-cust-ptr TO ADDRESS OF ls-user-space.
002000 delete-space.
132 002010 CALL "QUSDLTUS" USING ws-space, ws-err-data. 20
002020 read-customer-file.
002030* read all records from customer file and move into space
133 002040 OPEN INPUT cust-file.
134 002050 IF ws-file-stat-good THEN
135 002060 READ cust-file AT END CONTINUE
136 002070 END-READ
137 002080 PERFORM VARYING ls-record-counter FROM 1 BY 1
002090 UNTIL not ws-file-stat-good
138 002100 SET ls-cust-prev-ptr TO ws-cust-prev-ptr
002110* Move information from file into space
139 002120 MOVE fs-cust-name TO ls-cust-name
140 002130 MOVE fs-cust-number TO ls-cust-number
141 002140 MOVE fs-cust-address-length TO ls-cust-address-length
142 002150 MOVE fs-cust-address-data(1:fs-cust-address-length)
002160 TO ls-cust-address-data(1:ls-cust-address-length)
002170* Save ptr to current record
143 002180 SET ws-cust-prev-ptr TO ADDRESS OF ls-user-space
002190* Make sure next record on 16 byte boundary
144 002200 ADD LENGTH OF ls-customer-rec 21
002210 ls-cust-address-length TO 1 GIVING ws-addr-inc
145 002220 DIVIDE ws-addr-inc BY 16 GIVING ws-temp
002230 REMAINDER ws-temp-2
146 002240 SUBTRACT ws-temp-2 FROM 16 GIVING ws-temp
002250* Save total record length in user space
147 002260 ADD ws-addr-inc TO ws-temp GIVING ls-cust-rec-length
148 002270 SET ADDRESS OF ls-user-space
002280 TO ADDRESS OF ls-user-space(ls-cust-rec-length + 1:1)
002290* Get next record from file
149 002300 READ cust-file AT END CONTINUE
150 002310 END-READ
002320 END-PERFORM
002330* At the end of the loop have one more record than really
002340* have
151 002350 SUBTRACT 1 FROM ls-record-counter
002360 END-IF.
152 002370 CLOSE cust-file.
002380
002390 main-loop. 22
002400* write the records to the display until F3 entered

```

Figure 100 (Part 4 of 7). Example Using Pointers to Access User Spaces

```

Customer Information Display
5738CB1 V2R2M0 001000 AS/400 COBOL Source TESTER/POINTA AS400SYS 05/01/92 18:01:14 Page 6
153 002410 DISPLAY "Customer Information" AT 0124 WITH
002420 BLANK SCREEN REVERSE-VIDEO
002430 "Cust Customer Name Customer"
002440 AT 0305
002450 " Address"
002460 "Number" AT 0405
002470 "F3=Exit" AT 2202.
002480* if a pending error put on the display
154 002490 IF ws-error-msg NOT = SPACES THEN
155 002500 DISPLAY ws-error-msg at 2302 with beep highlight
156 002510 MOVE SPACES TO ws-error-msg
002520 END-IF.
002530* if in the middle of the list put F7 on the display
157 002540 IF ws-current-rec > 1 THEN 23
158 002550 DISPLAY "F7=Back" AT 2240
002560 END-IF.
002570* save the current record
159 002580 MOVE ws-current-rec TO ws-old-rec.
160 002590 SET ws-old-space-ptr TO ADDRESS OF ls-user-space. 24
002600* move each record to the display
161 002610 PERFORM VARYING ws-line FROM ws-start-line BY 1
002620 UNTIL ws-line > ws-displayed-lines or
002630 ws-current-rec > ls-record-counter
002640* if address is greater than display width show "+"
162 002650 IF ls-cust-address-length > 40 THEN
163 002660 MOVE "+" TO ws-plus
164 002670 MOVE 40 TO ws-temp-size
002680 ELSE
165 002690 MOVE ls-cust-address-length TO ws-temp-size
166 002700 MOVE SPACE TO ws-plus
002710 END-IF
167 002720 DISPLAY ls-cust-number at line ws-line column 5
002730 ls-cust-name ls-cust-address-data with
002740 size ws-temp-size ws-plus at line
002750 ws-line column 78
002760* get next record in the space
168 002770 ADD 1 TO ws-current-rec
169 002780 SET ADDRESS OF ls-user-space
002790 TO ADDRESS OF ls-user-space
002800 (ls-cust-rec-length + 1:1)
002810 END-PERFORM.
002820* if can go forward put F8 on the display
170 002830 IF ws-current-rec < ls-record-counter THEN 23
171 002840 DISPLAY "F8=Forward" AT 2250
002850 END-IF.
002860* check to see if continue, exit, or get next records or
002870* previous records
172 002880 ACCEPT ws-accept-data WITH SECURE 25
002890 ON EXCEPTION
173 002900 IF ws-status-1-func-key THEN
174 002910 IF ws-func-03 THEN
175 002920 SET ws-prog-end TO TRUE
002930 ELSE
176 002940 IF ws-func-07 THEN
177 002950 PERFORM back-screen
002960 ELSE
178 002970 IF ws-func-08 THEN
179 002980 PERFORM forward-screen
002990 ELSE
180 003000 MOVE "Invalid Function Key" TO ws-error-msg
181 003010 MOVE ws-old-rec TO ws-current-rec
182 003020 SET ADDRESS OF ls-user-space TO ws-old-space-ptr
003030 END-IF
003040 END-IF
003050 ELSE
183 003060 MOVE "Unknown Error" TO ws-error-msg
184 003070 MOVE ws-old-rec TO ws-current-rec
185 003080 SET ADDRESS OF ls-user-space TO ws-old-space-ptr
003090 END-IF
003100 NOT ON EXCEPTION
186 003110 MOVE ws-old-rec TO ws-current-rec
187 003120 SET ADDRESS OF ls-user-space TO ws-old-space-ptr
003130 END-ACCEPT.
003140 clean-up.
003150* do clean up for program

```

Figure 100 (Part 5 of 7). Example Using Pointers to Access User Spaces


```

Customer Information Display
5738CB1 V2R2M0 001000      AS/400 COBOL Source      TESTER/POINTA      AS400SYS 05/01/92 18:01:14      Page 7

003160* keep reading end display until entered data correct
188 003170      SET ws-prog-loop to TRUE.
189 003180      PERFORM end-display THRU read-end-display 26
003190      UNTIL NOT ws-prog-loop.
003200 end-display.
190 003210      DISPLAY "Delete Customer Information Area" AT 0118 WITH 27
003220      BLANK SCREEN REVERSE-VIDEO
003230      "Delete customer information area (Y/N)=> <="
003240      AT 1015
003250      "F3=Exit" AT 2202.
191 003260      IF ws-error-msg NOT = SPACES THEN
192 003270      DISPLAY ws-error-msg at 2302 with beep highlight
193 003280      MOVE SPACES TO ws-error-msg
003290      END-IF.
003300 read-end-display.
194 003310      ACCEPT ws-accept-data AT 1056 WITH REVERSE-VIDEO
003320      ON EXCEPTION
195 003330      IF ws-status-1-func-key THEN
196 003340      IF ws-func-03 THEN
197 003350      SET ws-prog-end TO TRUE
003360      ELSE
198 003370      MOVE "Invalid Function Key" TO ws-error-msg
003380      END-IF
003390      ELSE
199 003400      MOVE "Unknown Error" TO ws-error-msg
003410      END-IF
003420      NOT ON EXCEPTION
200 003430      IF ws-acc-delete-space THEN
201 003440      PERFORM delete-space
202 003450      SET ws-prog-continue TO TRUE
003460      ELSE
203 003470      IF NOT ws-acc-no-space THEN
204 003480      MOVE "Invalid Character Entered" TO ws-error-msg
003490      ELSE
205 003500      SET ws-prog-continue TO TRUE
003510      END-IF
003520      END-IF
003530      END-ACCEPT.
003540 back-screen. 28
206 003550      IF ws-old-rec <= 1 THEN
207 003560      MOVE "Top of customer records" TO ws-error-msg
208 003570      MOVE ws-old-rec TO ws-current-rec 29
209 003580      SET ADDRESS OF 1s-user-space TO ws-old-space-ptr
003590      ELSE
210 003600      MOVE ws-old-rec TO ws-current-rec 29
211 003610      SET ADDRESS OF 1s-user-space TO ws-old-space-ptr
212 003620      PERFORM VARYING ws-line FROM ws-start-line BY 1
003630      UNTIL ws-line > ws-displayed-lines or
003640      ws-current-rec <= 1
003650* Back up one record at a time
213 003660      SET ws-cust-prev-ptr TO 1s-cust-prev-ptr
214 003670      SET ADDRESS OF 1s-user-space TO ws-cust-prev-ptr 30
215 003680      SUBTRACT 1 FROM ws-current-rec
003690      END-PERFORM
003700      END-IF.
003710 forward-screen. 31
003720* if current record greater or equal to the max records
003730* print error, have reached max records
216 003740      IF ws-current-rec >= 1s-record-counter
217 003750      MOVE "No more customer records" TO ws-error-msg
218 003760      MOVE ws-old-rec TO ws-current-rec
219 003770      SET ADDRESS OF 1s-user-space TO ws-old-space-ptr
003780      ELSE
220 003790      MOVE ws-current-rec TO ws-old-rec
221 003800      SET ws-old-space-ptr TO ADDRESS OF 1s-user-space
003810      END-IF.
***** END OF SOURCE *****

Customer Information Display
5738CB1 V2R2M0 001000      AS/400 COBOL Messages      TESTER/POINTA      AS400SYS 05/01/92 18:01:14      Page 8
STMT
* 15 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000190
Message . . . . : Blocking/Deblocking for file 'CUST-FILE'
will be performed by compiler-generated code.
***** END OF MESSAGES *****

```

Figure 100 (Part 6 of 7). Example Using Pointers to Access User Spaces

Total	Info(0-4)	Warning(5-19)	Message Summary		
1	1	0	Error(20-29)	Severe(30-39)	Terminal(40-99)
0	0	0	0	0	0
Source records read	:	381			
Copy records read	:	15			
Copy members processed	:	1			
Sequence errors	:	0			
Highest severity message issued	:	0			
LBL0901 00 Program POINTA created in library TESTER.					
***** END OF COMPILATION *****					

Figure 100 (Part 7 of 7). Example Using Pointers to Access User Spaces

- 1** The compiler directive TITLE is used to create this title that appears at the beginning of each page.
- 3** CRT STATUS IS specifies a data name into which a status value is placed after the termination of an extended ACCEPT statement. In this example, the STATUS key value is used to determine which function key was pressed.
- 4** *fs-cust-address* is a variable-length field. To see meaningful names here rather than FILLER, specify *VARCHAR for the CVTOPT parameter of the CRTCPBLPGM command, or VARCHAR in the PROCESS statement, as shown in **2**. For more information about variable-length fields, refer to “Declaring Data Items Using CVTOPT Data Types” on page 127.
- 5** CRT STATUS as mentioned in **3** is defined here.
- 6** The *ws-params* structure contains the parameters used when calling the APIs to access user spaces.
- 7** *ws-err-data* is the structure for the error parameter for the user space APIs. Note that the *ws-input-1* is zero, meaning that any exceptions are signalled to the program, and not passed in the error code parameter. For more information on error code parameters, refer to the *System Programmer’s Interface Reference*, SC41-8223.
- 8** *ws-space-ptr* defines a pointer data item set by the API QUSPTRUS. This points to the beginning of the user space, and is used to set the addresses of items in the Linkage Section.
- 9** The first data structure (*ls-header-record*) to be defined in the user space.
- 10** FILLER is used to maintain pointer alignment, because it makes *ls-header-record* a multiple of 16 bytes long.
- 11** The second data structure (*ls-user-space*) to be defined in the user space.
- 12** *initial-display* shows the Create Customer Information Area display. This display is shown in Figure 101 on page 296.
- 13** *read-initial-display* reads the first display, and determines if the user chooses to continue or end the program. If the user continues the program by pressing Enter, then the program checks *ws-accept-data* to see if the customer information area is to be created.
- 14** QUSCRTUS is an API used to create user spaces.
- 15** QUSPTRUS is an API used to return a pointer to the beginning of a user space.

- 16** Maps the first data structure (*ls-header-record*) over the beginning of the user space.
- 17** Maps the second data structure (*ls-user-space*) over the beginning of the user space.
- 18** Uses ADDRESS OF special register
- 19** Uses ADDRESS OF, not the ADDRESS OF special register, because it is reference modified.
- 20** QUSDLTUS is an API used to delete a user space.
- 21** The following four arithmetic statements calculate the total length of each record, and ensure that each record is a multiple of 16 bytes in length.
- 22** *main-loop* puts up the Customer Information display. Refer to Figure 102 on page 296.
- 23** These statements determine if the program should display function keys F7 and F8.
- 24** Saves a pointer to the first customer record on the display.
- 25** This ACCEPT statement waits for input from the Customer Information display. Based on the function key pressed, it calls the appropriate paragraph to display the next set of records (*forward-screen*), or the previous set of records (*back-screen*), or sets an indicator to end the routine if F3 is pressed.
- 26** The clean up routine displays the Delete Customer Information Area display until an appropriate key is pressed.
- 27** This statement puts up the Delete Customer Information Area display.
- 28** Each record contains a pointer to the previous customer record. The ADDRESS OF special register points to the current customer record. By changing the ADDRESS OF special register, the current customer record is changed.
- back-screen* moves the current record pointer backward one record at a time **30**, by moving the pointer to the previous customer record into the pointer to the current customer record (ADDRESS OF). Before moving backward one record at a time, the program sets the current customer record to the first record currently displayed **29**.
- 31** *forward-screen* sets *ws-old-space-ptr* (which points to the first record in the display) to point to the current record (which is after the last record displayed.)
- A user space always begins on a 16-byte boundary, so the method illustrated here ensures that **all** records are aligned. *ls-cust-rec-length* is also used to chain the records together.

When you run POINTA, you see the following displays:

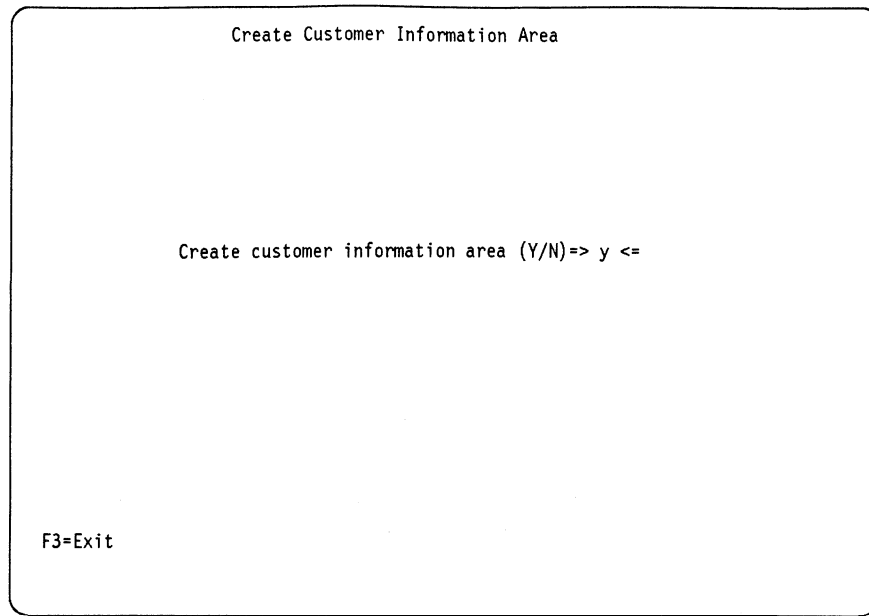


Figure 101. Create Customer Information Area Display

If you specify Y to create the user space, the program reads the customer records from the file and puts the information in the user space. The records are chained together.

When you press enter from the previous display, the Customer Information display appears:

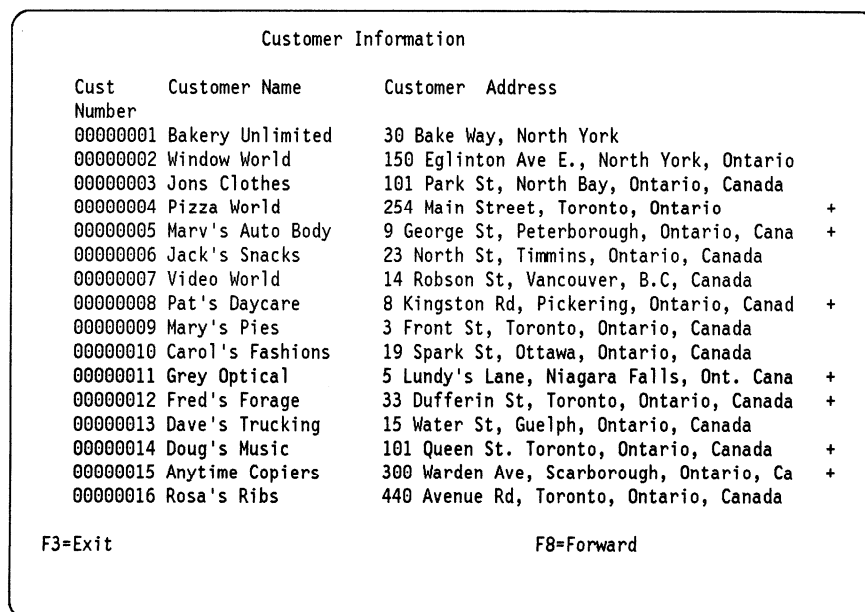


Figure 102. Customer Information Area Display

If there are more than 16 records in the user space (based on the starting line in *ws-start-line*), the program enables the F8=Forward key, to allow the user to page forward in the list. Once the user has rolled forward, the F7=Backward

Processing a Chained List

A typical application for using pointer data items is in processing a chained list (a series of records where each one points to the next).

For this example, picture a chained list of data that is composed of individual salary records. Figure 105 shows one way to visualize how these records are linked in storage:

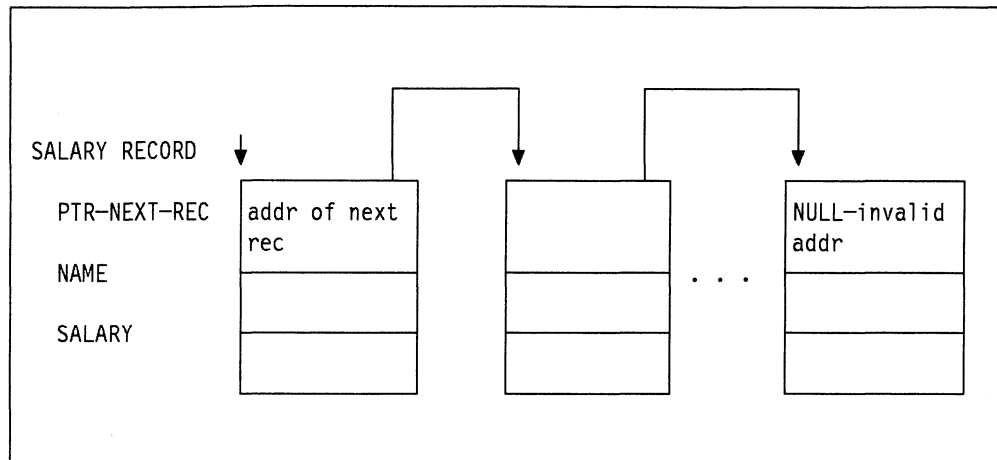


Figure 105. Representation of a Chained List Ending with NULL

The first item in each record (except for the last record) points to the next record. The first item in the last record, in order to indicate that it is the last record, contains a null value instead of an address.

The high-level logic of an application that processes these records might look something like this:

```
OBTAIN ADDRESS OF FIRST RECORD IN CHAINED LIST FROM ROUTINE
CHECK FOR END OF THE CHAINED LIST
DO UNTIL END OF THE CHAINED LIST
    PROCESS RECORD
    GO ON TO THE NEXT RECORD
END
```

Figure 106 on page 299 contains an outline of the processing program, `LISTS`, used in this example of processing a chained list.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. LISTS.
ENVIRONMENT DIVISION.
DATA DIVISION.
*****
WORKING-STORAGE SECTION.
77 PTR-FIRST          POINTER VALUE IS NULL.
77 DEPT-TOTAL        PIC 9(4) VALUE IS 0.
*****
LINKAGE SECTION.
01 SALARY-REC.
   02 PTR-NEXT-REC    POINTER.
   02 NAME            PIC X(20).
   02 DEPT            PIC 9(4).
   02 SALARY          PIC 9(6).
01 DEPT-X            PIC 9(4).
*****
PROCEDURE DIVISION USING DEPT-X.
*****
* FOR EVERYONE IN THE DEPARTMENT RECEIVED AS DEPT-X,
* GO THROUGH ALL OF THE RECORDS IN THE CHAINED LIST BASED ON THE
* ADDRESS OBTAINED FROM THE PROGRAM CHAIN-ANCH
* AND ACCUMULATE THE SALARIES.
* IN EACH RECORD, PTR-NEXT-REC IS A POINTER TO THE NEXT RECORD
* IN THE LIST; IN THE LAST RECORD, PTR-NEXT-REC IS NULL.
* DISPLAY THE TOTAL.
*****
      CALL "CHAIN-ANCH" USING PTR-FIRST
      SET ADDRESS OF SALARY-REC TO PTR-FIRST
*****
      PERFORM WITH TEST BEFORE UNTIL ADDRESS OF SALARY-REC = NULL
      IF DEPT = DEPT-X
      THEN ADD SALARY TO DEPT-TOTAL
      ELSE CONTINUE
      END-IF
      SET ADDRESS OF SALARY-REC TO PTR-NEXT-REC
      END-PERFORM
*****
      DISPLAY DEPT-TOTAL
      GOBACK.

```

Figure 106. Program for Processing a Chained List

Passing Addresses between Programs

To obtain the address of the first SALARY-REC record area, the LISTS program calls the program CHAIN-ANCH:

```
CALL "CHAIN-ANCH" USING PTR-FIRST
```

PTR-FIRST is defined in WORKING-STORAGE in the calling program (LISTS) as a pointer data item:

```
WORKING-STORAGE SECTION.
77 PTR-FIRST          POINTER VALUE IS NULL.
```

Upon return from the call to CHAIN-ANCH, PTR-FIRST contains the address of the first record in the chained list.

PTR-FIRST is initially defined as having a null value as a logic check. If an error occurs with the call, and PTR-FIRST never receives the value of the address of the first record in the chain, a null value remains in PTR-FIRST and, according to the logic of the program, the records will not be processed.

NULL is a figurative constant used to assign the value of a non-valid address to pointer items. It can be used in the VALUE IS NULL clause, in the SET statement, and as an operand in a relation condition with a pointer data item.

The Linkage Section of the calling program contains the description of the records in the chained list. It also contains the description of the department code that is passed through the USING phrase of the CALL statement.

```
LINKAGE SECTION.  
01 SALARY-REC.  
    02 PTR-NEXT-REC    POINTER.  
    02 NAME            PIC X(20).  
    02 DEPT           PIC 9(4).  
    02 SALARY         PIC 9(6).  
01 DEPT-X            PIC 9(4).
```

To "base" the record description SALARY-REC on the address contained in PTR-FIRST, use a SET statement:

```
CALL "CHAIN-ANCH" USING PTR-FIRST  
SET ADDRESS OF SALARY-REC TO PTR-FIRST
```

Check for the End of the Chained List

The chained list in this example is set up so that the last record contains a non-valid address. To do this, the pointer data item in the last record would be assigned the value NULL.

A pointer data item can be assigned the value NULL in two ways:

- A pointer data item can be defined with a VALUE IS NULL clause in its data definition.
- NULL can be the sending field in a SET statement.
- The initial value of a pointer data item with or without a VALUE clause of NULL equals NULL.

In the case of a chained list in which the pointer in the last record contains a null value, the code to check for the end of the list would be:

```
IF PTR-NEXT-REC = NULL  
:  
  
(logic for end of chain)
```

If you have not reached the end of the list, process the record and move on to the next record.

In the program `LISTS`, this test for the end of the chained list is accomplished with a “do while” structure:

```
PERFORM WITH TEST BEFORE UNTIL ADDRESS OF SALARY-REC = NULL
  IF DEPT = DEPT-X
    THEN ADD SALARY TO DEPT-TOTAL
    ELSE CONTINUE
  END-IF
  SET ADDRESS OF SALARY-REC TO PTR-NEXT-REC
END-PERFORM
```

Continuing Processing the Next Record

To move on to the next record, set the address of the record in the Linkage Section to be equal to the address of the next record. This is accomplished through the pointer data item sent as the first field in `SALARY-REC`:

```
SET ADDRESS OF SALARY-REC TO PTR-NEXT-REC
```

Then repeat the record-processing routine, which will process the next record in the chained list.

Incrementing Addresses Received from Another Program

The data passed from a calling program might contain header information that you want to ignore (for example, in data received from a CICS application that is not migrated to the command level).

Because pointer data items are not numeric, you cannot directly perform arithmetic on them. You can, however, use the `SET` verb to increment the passed address in order to bypass header information.

You could set up the Linkage Section as follows:

```
LINKAGE SECTION.
01 RECORD-A.
  02 HEADER          PIC X(16).
  02 REAL-SALARY-REC PIC X(30).
:
01 SALARY-REC.
  02 PTR-NEXT-REC   POINTER.
  02 NAME           PIC X(20).
  02 DEPT          PIC 9(4).
  02 SALARY        PIC 9(6).
```

Within the Procedure Division, base the address of `SALARY-REC` on the address of `REAL-SALARY-REC`:

```
SET ADDRESS OF SALARY-REC TO ADDRESS OF REAL-SALARY-REC
```

`SALARY-REC` is now based on the address of `RECORD-A + 16`.

Data Areas

A data area is an object used to communicate data such as variable values between programs within a job and between jobs. A data area can be created and declared to a program before it is used in that program or job. For information on how to create and declare a data area, see the *CL Programmer's Guide*.

Local Data Area

The local data area can be used to pass any desired information between programs in a job. This information may be free-form data, such as informal messages, or may consist of a fully structured or formatted set of fields.

The system automatically creates a local data area for each job. The local data area is defined outside the COBOL program as an area of 1024 bytes.

When a job is submitted, the submitting job's local data area is copied into the submitted job's local data area. If there is no submitting job, the local data area is initialized to blanks.

A COBOL program can access the local data area for its job with the ACCEPT and DISPLAY statements, using a mnemonic name associated with the function-name LOCAL-DATA.

There is only one local data area associated with each job. Even if several work stations are acquired by a single job, only one local data area exists for that job. There is *not* a local data area for each work station.

Program Initialization Parameters (PIP) Data Area

The PIP data area is used by a prestart job. Generally, a prestart job is a job from a remote system under ICF that you start and keep ready to run until you call it.

If you use a prestart job, you do not have to wait for a program that you call to go through job initiation processing. Job initiation is performed before a program can actually start. Because job initiation has already taken place, a prestart job allows your program to start more quickly after the program start request is received.

A COBOL program can access the PIP data area for its job with the ACCEPT statement, using a mnemonic name associated with the function-name PIP-DATA.

The PIP data area is a 2 000-byte alphanumeric item and contains parameters received from a calling program. It provides the program initialization parameters that, in non-prestart jobs, is provided through standard COBOL parameters.

You use a Format 5 ACCEPT statement to access the PIP data area, similar to the way in which you use a Format 4 ACCEPT statement to read from the local data area. Note that you cannot update the PIP data area using COBOL. See the *COBOL/400* Reference* for detailed syntax information.

For more information regarding prestart jobs and the PIP data area, refer to the *Work Management Guide* and the *CL Programmer's Guide*.

File Considerations

You can pass a file name as a parameter in a COBOL program, but you cannot use that file in the called program. If a file is defined in both a calling program and a called program, it is treated as two separate files. The contents of the record area and the current record pointer in each program are independent,

unless shared files are specified in CL commands. See the *Data Management Guide* for further information on shared files.

The following statements affect file status differently:

- An EXIT PROGRAM statement does not change the status of any of the files in a run unit.
- A STOP RUN statement closes all of the files in a run unit.

IBM Extension

- A GOBACK statement issued from a main program closes all of the files in a run unit. A GOBACK statement issued from a subprogram does not change the status of any of the files in a run unit.

End of IBM Extension

- A CANCEL statement does not change the status of any of the files in the program that is canceled. It does free the storage that contains information about the file. If the program has files that are open when the CANCEL statement is processed, those files are closed when that program is cancelled. The program can no longer use the file. If the canceled program is called again, the program considers the file closed. If the program opens the file, a new linkage to the file is established. This can cause additional system storage to be used.

Appendix A. Segmentation Feature

You do not have to be concerned with storage management when writing COBOL/400 programs. Storage segmentation is, however, available for compatibility with other systems.

The segmentation feature provides programmer-controlled storage optimization of the Procedure Division by allowing that division to be subdivided both physically and logically.

Segmentation Concepts

Although it is not required, the Procedure Division of a source program is often written as a consecutive group of sections, each of which is made up of a series of related operations that perform a particular function. Thus, the entire Procedure Division is made up of a number of logical subdivisions. Segmentation allows the programmer to physically divide the Procedure Division into segments, each of which has specific physical and logical attributes.

When Segmentation is used, the entire Procedure Division must be divided into sections. Each section must then be classified as to its physical and logical attributes. Classification is specified by means of segment numbers. All sections given the same segment number make up one program segment.

Segment numbers must be integers from 0 through 99.

Program Segments

There are three types of program segments; fixed permanent, fixed overlayable, and independent.

Fixed Segments

Fixed-permanent segments and fixed-overlayable segments make up the fixed portion, the part of the Procedure Division that is logically treated as if it were always physically present in main storage. Fixed-portion segment numbers must be integers from 0 through 49.

A fixed-permanent segment is always made available in its last-used state.

A fixed-overlayable segment is logically always in main storage during program processing; therefore, it is always available in its last-used state. Any overlay of such a segment is transparent to the user. Thus, a fixed-overlayable segment is logically identical with a fixed-permanent segment.

Independent Segments

Logically, an independent segment can overlay and be overlaid by other segments during a program's run.

An independent segment is made available in its initial state the first time control is passed to it (explicitly or implicitly) during a program's run.

An independent segment is made available in its initial state during subsequent transfers of control when:

- The transfer is the result of an implicit transfer of control between consecutive statements that are in different segments (that is, when control drops through into the independent segment from the physically preceding segment).
- The transfer is the result of an implicit transfer from a SORT or MERGE statement in one segment to a SORT input procedure or SORT/MERGE output procedure in an independent segment.
- An explicit transfer of control from a section with a different segment number takes place (as, for example, during the transfer of control in a PERFORM n TIMES statement).

An independent segment is made available in its last-used state during subsequent transfers of control when:

- With the exception of the two preceding kinds of implied transfers, an implicit transfer from a section with a different priority takes place (as, for example, when control is returned to the independent segment from a Declarative procedure).
- An explicit transfer results from an EXIT PROGRAM or GOBACK statement.

Independent segments must be assigned segment numbers 50 through 99.

Segmentation Logic

In a segmented program, the sections are classified by a system of segment numbers according to the following criteria:

- *Frequency of Reference*—Much-referenced sections, or those that must be available for reference at all times, should be placed within fixed permanent segments. Less frequently used sections can be within either fixed overlayable or independent segments, depending on the program logic.
- *Frequency of Use*—The more frequently a section is used, the lower its segment number; the less frequently it is referred to, the higher its segment number.
- *Logical Relationships*—Sections that frequently communicate with each other should be given identical segment numbers.

Segmentation Control

Except for specific transfers of control, the logical sequence and the physical sequence of program instructions are the same. The compiler inserts any instructions necessary to initialize a segment. It is not necessary to transfer control to the beginning of a segment, or to the beginning of a section within a segment. Instead, control can be transferred to any paragraph in the Procedure Division.

COBOL Source Program Considerations

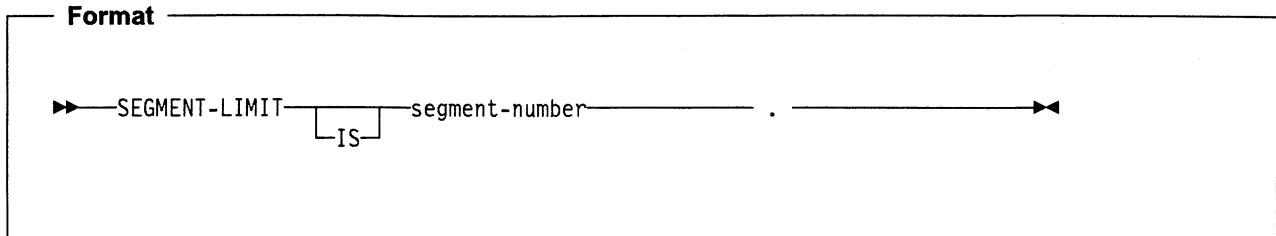
The following elements of a COBOL source program implement the Segmentation feature:

- The SEGMENT-LIMIT clause in the OBJECT-COMPUTER paragraph of the Environment Division. This clause allows you to control the specification of fixed-permanent and fixed-overlayable segments.
- Procedure Division segment numbers, which group sections into segments. The segment numbering scheme also allows specifications of independent

segments, fixed-permanent segments, and (in conjunction with the SEGMENT-LIMIT clause) of fixed-overlayable segments.

Segmentation – Environment Division

In the OBJECT-COMPUTER paragraph, the SEGMENT-LIMIT clause allows the user to reclassify fixed permanent segments while retaining the properties of fixed portion segments for the reclassified segments.



The SEGMENT-LIMIT clause allows the programmer to specify certain permanent segments as capable of being overlaid by independent segments without losing the logical properties of fixed portion segments.

segment-number must be an integer ranging in value from 1 through 49.

When the SEGMENT-LIMIT clause is specified:

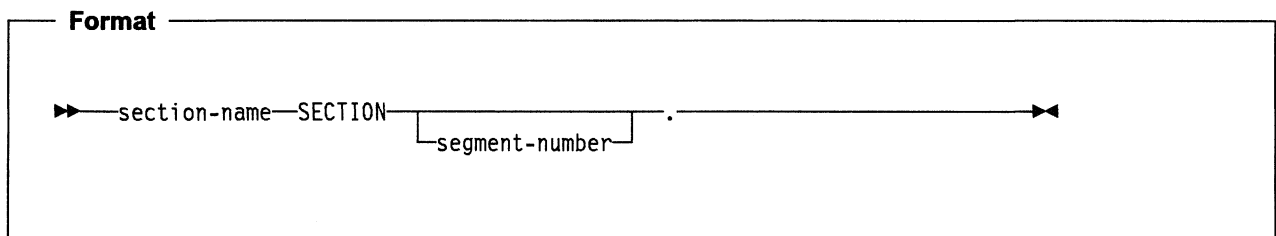
- Fixed-permanent segments are those with segment numbers from 0 up to, but not including, the segment number specified.
- Fixed-overlayable segments are those with segment numbers from the segment number specified through 49.

For example, if SEGMENT-LIMIT IS 25 is specified, sections with segment numbers 0 through 24 are fixed-permanent segments, and sections with segment numbers 25 through 49 are fixed-overlayable segments.

When the SEGMENT-LIMIT clause is omitted, all sections with segment numbers 0 through 49 are fixed-permanent segments.

Segmentation – Procedure Division

In the Procedure Division of a segmented program, section classification is specified through segment numbers in the section headers. The segment number must be an integer from 0 through 99.



All sections with the same segment number make up one program segment. Such sections need not be contiguous in the source program.

Segments with segment numbers 0 through 49 are in the fixed portion of the program. Declarative sections can be assigned only these segment numbers. Segments with segment numbers from 50 through 99 are independent segments. If the segment number is omitted from the section header, the segment number is assumed to be 0.

Segmentation – Special Considerations

When segmentation is used, there are restrictions on the ALTER, PERFORM, SORT, and MERGE statements. There are also special considerations for calling and called programs.

ALTER Statement

A GO TO statement in an independent segment must not be referred to by an ALTER statement in a different segment. All other uses of the ALTER statement are valid and are performed, even if the GO TO statement referred to is in a fixed-overlayable segment.

PERFORM Statement

A PERFORM statement in the fixed portion can have in its range, in addition to any Declarative procedures, the processing of which is caused within that range, only one of the following:

- Sections and/or paragraphs in the fixed portion
- Sections and/or paragraphs contained within a single independent segment.

A PERFORM statement in an independent segment can have within its range, in addition to any Declarative procedures, the processing of which is caused within that range, only one of the following:

- Sections and/or paragraphs in the fixed portion
- Sections and/or paragraphs wholly contained in the same independent segment as the PERFORM statement.

SORT and MERGE Statements

If a SORT or MERGE statement appears in the fixed portion, any SORT input procedures or SORT/MERGE output procedures must appear completely in one of the following:

- The fixed portion
- A single independent segment.

If a SORT or MERGE statement appears in an independent segment, any SORT input procedures or SORT/MERGE output procedures must appear completely in one of the following:

- The fixed portion
- The same independent segment as the SORT or MERGE statement.

Calling and Called Programs

The CALL statement can appear anywhere within a segmented program. When a CALL statement appears in an independent segment, that segment is in its last-used state when control is returned to the calling program.

Appendix B. Debugging Features

The debugging features specify the conditions under which procedures are to be monitored during program run time.

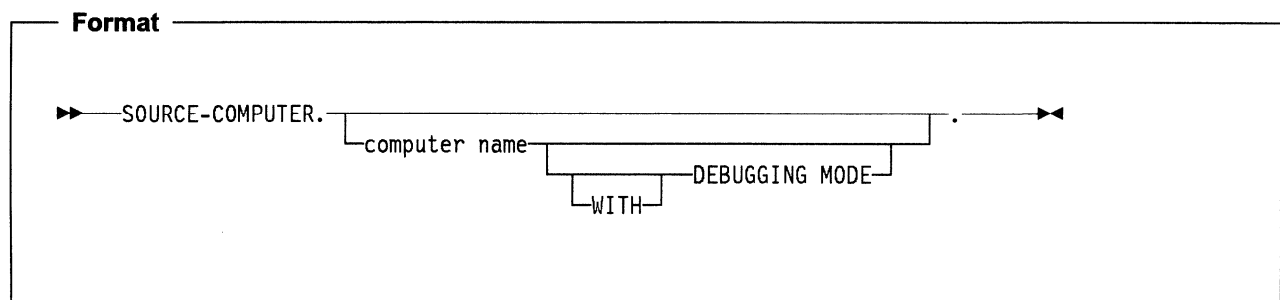
COBOL source language debugging statements are provided. You must decide what to monitor, and what information you need to retrieve for debugging purposes. The COBOL debugging features simply provide access to pertinent information.

COBOL Source Language Debugging

COBOL language elements that implement the Debugging Feature are a compile-time switch (WITH DEBUGGING MODE), a run-time switch, a USE FOR DEBUGGING Declarative, the special register DEBUG-ITEM, and debugging lines that can be written in the Environment, Data, and Procedure Divisions.

Compile-Time Switch

In the SOURCE-COMPUTER paragraph of the Configuration Section, the WITH DEBUGGING MODE clause acts as a compile-time switch.



The WITH DEBUGGING MODE clause serves as a compile-time switch for the debugging statements written in the source program.

When WITH DEBUGGING MODE is specified, all debugging sections and debugging lines are compiled as specified in this appendix. When WITH DEBUGGING MODE is omitted, all debugging sections and debugging lines are treated as documentation.

Run-Time Switch

The run-time switch dynamically activates the debugging code that is generated when WITH DEBUGGING MODE is specified.

Two commands are provided to control the run-time switch. To set the run-time switch on, enter the command:

STRCBLDBG

and press F4.

You see the following display:

```

Start COBOL Debug (STRCBLDBG)

Type choices, press Enter.

Program .....          Name
Library .....          Name, *LIBL, *CURLIB

F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display  Bottom
F24=More keys
  
```

The following diagram shows the syntax of the STRCBLDBG command:

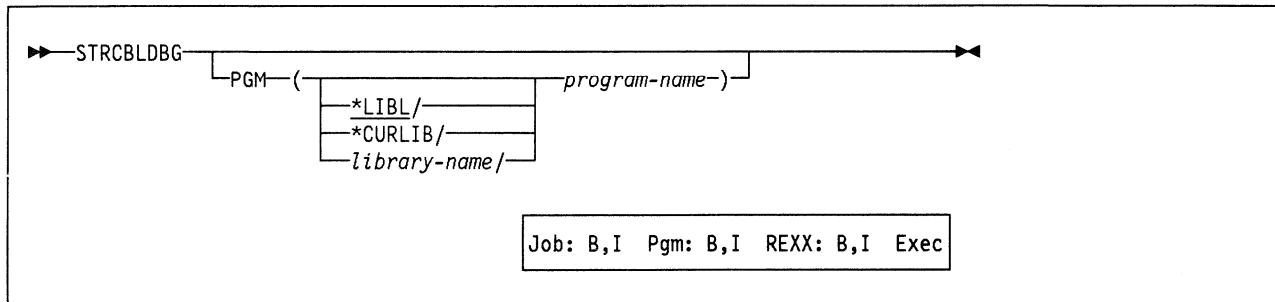


Figure 107. Syntax of the STRCBLDBG Command

This command is allowed in interactive and batch processing, and in CL programs.

General-Use Programming Interface

You can use this command in QCMDXEC.

End of General-Use Programming Interface

To set the run-time switch off, enter the command:

ENDCBLDBG

and press F4.

You see the following display:

```
End COBOL Debug (ENDCBLDBG)

Type choices, press Enter.

Program . . . . . Name
Library . . . . . *LIBL Name, *LIBL, *CURLIB

F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display Bottom
F24=More keys
```

The following diagram shows the syntax of the ENDCBLDBG command:

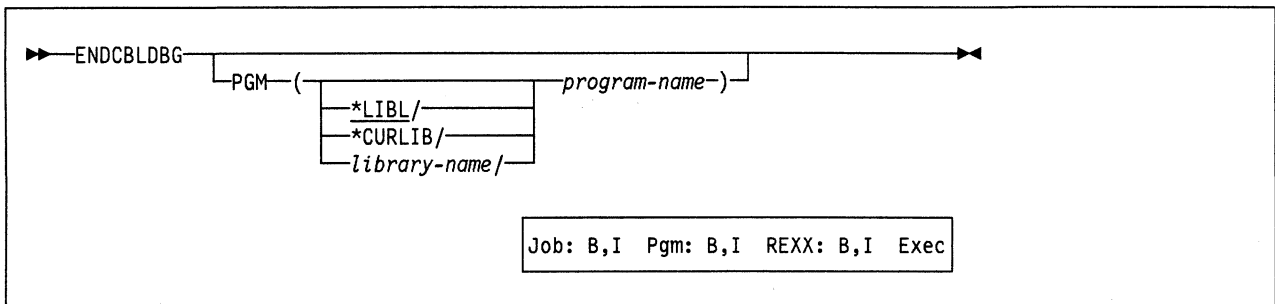


Figure 108. Syntax of the ENDCBLDBG Command

This command is allowed in interactive and batch processing, and in CL programs.

General-Use Programming Interface

You can use this command in QCMDXEC.

End of General-Use Programming Interface

The default for the run-time switch is off.

When debugging mode is specified through the run-time switch, all the debugging sections and debugging lines (D in column 7) compiled into the program are activated.

You must enter the STRCBLDBG command for each COBOL program (main program or called program) to be debugged in the next COBOL run unit. At the end of the run unit, all run-time switches that are on are set off. If a switch must be set off before starting a COBOL run unit, use the ENDCBLDBG command. Run-time switches for up to 15 programs can be on at once.

When the STRCBLDBG or ENDCBLDBG command is issued in a CL program, concatenation expressions can be used for all parameter values. See the *CL Programmer's Guide* for more information about concatenation expressions.

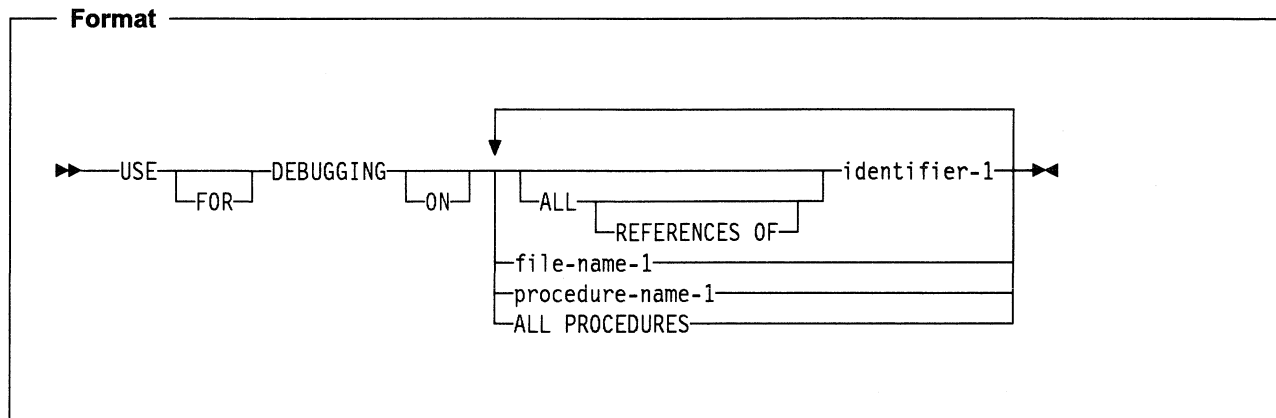
When debugging mode is suppressed, through the run-time switch, any USE FOR DEBUGGING Declarative procedures are inhibited. All debugging lines (D in column 7) remain in effect.

Recompilation of the source program is not required to activate or deactivate the run-time switch.

When WITH DEBUGGING MODE is not specified in the SOURCE-COMPUTER paragraph, the run-time switch has no effect on the running of the program.

USE FOR DEBUGGING Declarative

The USE FOR DEBUGGING sentence in the Procedure Division identifies the items in the source program that are to be monitored by the associated debugging declarative procedure.



Identifier-1 cannot be reference modified.

When specified, all debugging sections must be written immediately after the DECLARATIVES header. Except for the USE FOR DEBUGGING sentence there must be no reference to any non-declarative procedure within the debugging procedure.

Note that the USE FOR DEBUGGING declarative causes all subsequent statements to be ignored up to a valid USE AFTER EXCEPTION/ERROR statement, or END DECLARATIVES delimiter. Entire programs can be ignored because of this.

Automatic running of a debugging section is not caused by a statement appearing in a debugging section.

A debugging section for a specific operand is processed only once as the result of the running of a single statement, no matter how many times the operand is

specified in the statement. An exception to this rule is that each specification of a subscripted or indexed identifier where the subscripts or indexes are different causes the calling of the debugging Declarative. For a PERFORM statement that causes repeated running of a procedure, any associated procedure name debugging Declarative section is run only once for each processing of the procedure.

For debugging purposes, each separate occurrence of an imperative verb within an imperative statement begins a separate statement.

Statements appearing outside the debugging sections must not refer to procedure names defined within the debugging sections.

Except for the USE FOR DEBUGGING sentence itself, statements within a debugging Declarative section can only refer to procedure names defined in a different USE procedure through the PERFORM statement. Procedure names within debugging Declarative sections must not appear in USE FOR DEBUGGING sentences.

Table 7 defines the points during program run time when the USE FOR DEBUGGING procedures are processed. Identifier-n, file-name-n, and procedure-name-n refer to the first and all subsequent specifications of that type of operand in one USE FOR DEBUGGING sentence. Any particular identifier, file name, or procedure name can appear in only one USE FOR DEBUGGING sentence, and only once in that sentence.

An identifier in a USE FOR DEBUGGING sentence:

- Must be specified without the subscripting or indexing normally required if it contains an OCCURS clause or is subordinate to an entry containing an OCCURS clause. (A SEARCH or SEARCH ALL statement that refers to such an identifier does not call the USE FOR DEBUGGING procedures.)
- Must not be a special register.

When ALL PROCEDURES is specified in a USE FOR DEBUGGING sentence, procedure-name-1, procedure-name-2, procedure-name-3, and so on, must not be specified in any USE FOR DEBUGGING sentence. The ALL PROCEDURES phrase can be specified only once in a program.

When a USE FOR DEBUGGING operand is used as a qualifier, such a reference in the program does not activate the debugging procedures.

References to the DEBUG-ITEM special register can be made only from within a debugging Declarative procedure.

<i>Table 7. Running Debugging Declaratives</i>	
USE FOR DEBUGGING Operand	The USE FOR DEBUGGING procedures run immediately after the following:
identifier-n	<p>Before REWRITE/WRITE identifier-n and after FROM phrase move, if applicable.</p> <p>After each initialization, modification, or evaluation of identifier-n in PERFORM/VARYING/AFTER/UNTIL identifier-n.</p> <p>After any other COBOL statement that explicitly refers to identifier-n and could change its contents. (See note.)</p>
ALL REFERENCES OF identifier-n	<p>Before GO TO DEPENDING ON identifier-n, control is transferred, and before any associated debugging section for the procedure name runs.</p> <p>Before REWRITE/WRITE identifier-n and FROM phrase move, if applicable.</p> <p>After each initialization, modification or evaluation of identifier-n in PERFORM/VARYING/AFTER/UNTIL identifier-n.</p> <p>After any other COBOL statement explicitly referring to identifier-n. (See note.)</p>
file-name-n	<p>After CLOSE/DELETE/OPEN/START file-name-n.</p> <p>After READ file-name-n where AT END/INVALID KEY was not run.</p>
procedure-name-n	<p>Before each running of the named procedure.</p> <p>After running an ALTER statement referring to the named procedure.</p>
ALL PROCEDURES	<p>Before each running of every non-debugging procedure.</p> <p>After running every ALTER statement (except ALTER statements in Declarative procedures).</p>

Note: Operands acted upon but not explicitly named in such statements as ADD, MOVE, or SUBTRACT CORRESPONDING never cause activation of a USE FOR DEBUGGING procedure when such statements are run. If identifier-n is specified in a phrase that is not processed, the associated debugging section is not run.

DEBUG-ITEM Special Register

The DEBUG-ITEM special register provides information for a debugging Declarative procedure. DEBUG-ITEM has the following implicit description:

01	DEBUG-ITEM.	
02	DEBUG-LINE	PICTURE IS X(6).
02	FILLER	PICTURE IS X VALUE SPACE.
02	DEBUG-NAME	PICTURE IS X(30).
02	FILLER	PICTURE IS X VALUE SPACE.
02	DEBUG-SUB-1	PICTURE IS S9999 SIGN IS LEADING SEPARATE CHARACTER.
02	FILLER	PICTURE IS X VALUE SPACE.
02	DEBUG-SUB-2	PICTURE IS S9999 SIGN IS LEADING SEPARATE CHARACTER.
02	FILLER	PICTURE IS X VALUE SPACE.
02	DEBUG-SUB-3	PICTURE IS S9999 SIGN IS LEADING SEPARATE CHARACTER.
02	FILLER	PICTURE IS X VALUE SPACE.
02	DEBUG-CONTENTS	PICTURE IS X(n).

The DEBUG-ITEM special register provides information about the conditions causing the running of a debugging section.

Before each debugging section is processed, DEBUG-ITEM is filled with spaces. The contents of the DEBUG-ITEM subfields are then updated according to the rules for the MOVE statement, with one exception: DEBUG-CONTENTS is updated as if the move were an alphanumeric-to-alphanumeric elementary move without conversion of data from one form of internal representation to another. After updating, each field contains:

- **DEBUG-LINE:** The compiler-generated statement number, right justified and padded on the left with zeros. For example, 000112.
- **DEBUG-NAME:** The first 30 characters of the name causing the debugging section to run. All qualifiers are separated by the word OF (subscripts or indexes are not entered in DEBUG-NAME).
- **DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3:** If the DEBUG-NAME is subscripted or indexed, the occurrence number of each level is entered in the respective DEBUG-SUB-n. If the item is not subscripted or indexed, these fields remain spaces.
- **DEBUG-CONTENTS:** Data is moved into DEBUG-CONTENTS as shown in Table 8. DEBUG-CONTENTS is the same size as the largest identifier in the program.

Table 8. DEBUG-ITEM Subfield Contents

Item Causing Debug Section To Run	DEBUG-LINE Contains Number of COBOL Statement Referring to	DEBUG-NAME Contains	DEBUG-CONTENTS Contains
identifier-n	identifier-n	identifier-n	Contents of identifier-n when control passes to debug section.
file-name-n	file-name-n	file-name-n	For READ: contents of record retrieved. Other references: spaces.
procedure-name-n ALTER reference	ALTER statement	procedure-name-n	procedure-name-n in TO PROCEED TO phrase
GO TO procedure-name-n	GO TO statement	procedure-name-n	
procedure-name-n in SORT/MERGE INPUT/OUTPUT PROCEDURE	SORT/MERGE statement	procedure-name-n	"SORT INPUT" "SORT OUTPUT" "MERGE OUTPUT" as applicable
PERFORM statement transfer of control	This PERFORM statement	procedure-name-n	"PERFORM LOOP"
procedure-name-n in a USE procedure	Statement causing USE procedure running	procedure-name-n	"USE PROCEDURE"
Implicit transfer from previous sequential procedure	Previous statement processed in previous sequential procedure (see note)	procedure-name-n	"FALL THROUGH"
First entry into first non-declarative procedure	Line number of first statement in the procedure	First non-declarative procedure name	"START PROGRAM"

Note: If this paragraph is preceded by a section header and control is passed through the section header, the statement number refers to the section header.

Debugging Lines

Debugging lines can help determine the cause of an error. A debugging line is any line in a source program with a **D** coded in column 7 (the continuation area). If a debugging line contains nothing but spaces in Area A and Area B, it is considered a blank line.

Each debugging line must be written so that a syntactically correct program results whether the debugging lines are compiled into the program or syntax-checked, but are treated as documentation.

Successive debugging lines are permitted. Debugging lines can be continued. However, each continuation line must contain a **D** in column 7, and character-strings must not be broken across two lines.

Debugging lines can be specified only after the OBJECT-COMPUTER paragraph.

When the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph, all debugging lines are compiled as part of the object program.

When the WITH DEBUGGING MODE clause is omitted, all debugging lines are syntax-checked, but are treated as documentation.

Appendix C. Level of Language Support

ANSI X3.23-1985 COBOL Standard

The ANSI X3.23-1985 COBOL standard consists of eleven functional processing modules, seven of which are required and four of which are optional.

The seven required modules are: Nucleus, Sequential I-O, Relative I-O, Indexed I-O, Inter-Program Communication, Sort-Merge, and Source Text Manipulation. The four optional modules are: Report Writer, Communication, Debug and Segmentation.

Language elements within the modules may be classified as level 1 elements and level 2 elements. Elements within nine of the modules are divided into level 1 elements and level 2 elements. Two of the modules (SORT-MERGE and REPORT WRITER) contain only level 1 elements. For instance, Nucleus level 1 elements perform basic internal operations. Nucleus level 2 elements provide for more extensive and sophisticated internal processing.

The three subsets of Standard COBOL are the high subset, the intermediate subset, and the minimum subset. Each subset is composed of a level of the seven required modules: Nucleus, Sequential I-O, Relative I-O, Indexed I-O, Inter-Program Communication, Sort-Merge, and Source Text Manipulation. The four optional modules (Report Writer, Communication, Debug and Segmentation) are not required in the three subsets of Standard COBOL.

The high subset is composed of all language elements of the highest level of all required modules. That is:

- Level 2 elements from Nucleus, Sequential I-O, Relative I-O, Indexed I-O, Inter-Program Communication, and Source Text Manipulation
- Level 1 elements from Sort-Merge.

The intermediate subset is composed of all language elements of level 1 of all required modules. That is:

- Level 1 elements from Nucleus, Sequential I-O, Relative I-O, Indexed I-O, Inter-Program Communication, Sort-Merge, and Source Text Manipulation.

The minimum subset is composed of all language elements of level 1 of the Nucleus, Sequential I-O, and Inter-Program Communication modules.

The four optional modules are not an integral part of any of the subsets. However, none, all, or any combination of the optional modules may be associated with any of the subsets.

COBOL/400 Level of Language Support

The COBOL/400 compiler supports:

- Level 1 of the Nucleus, Sequential I-O, Relative I-O, Indexed I-O, Inter-Program Communication, Sort-Merge, and Source Text Manipulation modules
- Level 2 of the Debug and Segmentation modules.

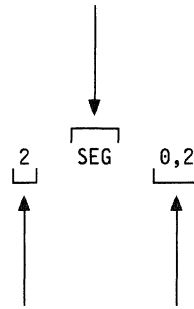
The Report Writer and Communication modules of ANSI X3.23-1985 COBOL are not supported by the COBOL/400 compiler.

The level of support provided by the COBOL/400 compiler is represented in the table below. The table:

- Shows the level of COBOL/400 compiler support for each functional processing module of the ANSI X3.23-1985 COBOL standard
- Describes each module.

Following is an explanation of the notation used within the table:

A 3-character code that identifies the module.
In this example, the Segmentation module, is referenced.



The level of this module supported by the COBOL/400 compiler. In this example, support is provided for the higher of the two levels of the Segmentation module.

The **range** of levels of support defined by the ANSI X3.23-1985 COBOL standard. A level of 0 means a **minimum** standard COBOL does not need to support this module to conform to the standard.

COBOL/400 Level of Language Supported	Module Description
Nucleus 1 NUC 1,2	Contains the language elements necessary for internal processing of data within the four basic divisions of a program and the capability for defining and accessing tables.
Sequential I-O 1 SEQ 1,2	Provides access to file records by the established sequence in which they were written to the file.
Relative I-O 1 REL 0,2	Provides access to records in either a random or sequential manner. Each record is uniquely identified by an integer that represents the record's logical position in the file.
Indexed I-O 1 INX 0,2	Provides access to records in either random or sequential manner. Each record in an indexed file is uniquely identified by a record key.
Inter-program Communication 1 IPC 1,2	Allows a COBOL program to communicate with other programs through transfers of control and access to common data items.
Sort-Merge 1 SRT 0,1	Orders one or more files of records, or combines two or more identically ordered files according to user-specified keys.
Source-Text Manipulation 1 STM 0,2	Allows insertion of predefined COBOL text into a program at compile time.

<i>Table 9 (Page 2 of 2). Level of COBOL/400 Compiler Support</i>	
COBOL/400 Level of Language Supported	Module Description
Report Writer 0 RPW 0,1	Provides semiautomatic production of printed reports.
Communications 0 COM 0,2	Provides the ability to access, process, and create messages or portions of messages; also allows communication through a Message Control System with local and remote communication devices.
Debug 2 DEB 0,2	Allows you to specify statements and procedures for debugging.
Segmentation 2 SEG 0,2	Provides the overlaying at object time of Procedure Division sections.

SAA Common Programming Interface (CPI) Support

Source file QILBINC in product libraries QLBL and QLBLP contains members that hold specifications for multiple SAA Common Programming Interfaces. These specifications describe parameter interfaces. This file is IBM-owned and should not be changed.

If you want to customize any of the specifications, you must copy any members that you want to change to a source file in one of your libraries. You can use the Copy File (CPYF) command to do this. For more information about the CPYF command, refer to the *CL Reference*.

If you copy these specifications to your library, you must refresh your copies when a new product release is installed, or when any changes are made using a Program Temporary Fix (PTF). IBM provides maintenance for these specifications only in the libraries in which they are distributed.

Appendix D. COBOL/400 Messages, the FIPS Flagger, and SAA Flagging

COBOL/400 Messages

This appendix provides a general description of messages that IBM supplies with the COBOL/400 licensed program.

Interactive Messages

In an interactive environment, messages are displayed on the work station display. They can appear on the current display as a result of the running of the program or in response to your keyed input to prompts, menus, command entry displays, or Application Development Tools (Appl Dev Tools). The messages can also appear on request, as a result of a display command or an option on a menu.

The messages for the COBOL/400 licensed program begin with an LSC, LBE, or LBL prefix.

The LSC messages are issued by the COBOL/400 syntax checker when the Source Entry Utility (SEU) is used to enter your COBOL/400 source. For example, you see the following display after incorrectly entering the program name in the PROGRAM-ID paragraph.

```
Columns . . . : 1 71          Edit          XMPLIB/QLBLSRC
SEU==> _____ TESTPR
FMT CB .....-A+++B+*****
***** Beginning of data *****
0000.10 IDENTIFICATION DIVISION.
0000.20 PROGRAM-ID. #TESTPR.
0000.70 ENVIRONMENT DIVISION.
0000.90 SOURCE-COMPUTER. IBM-AS400.
***** End of data *****

F3=Exit  F4=Prompt  F5=Refresh  F9=Retrieve  F10=Cursor
F16=Repeat find  F17=Repeat change  F24=More keys
# not in COBOL character set. Line rejected.
```

Figure 109. Example of a COBOL/400 Syntax Checker Message

LBE messages provide you with additional information about system operation during run time. For example, you might see the following display if you have a run-time error:

```

                                Display Program Messages

Job 011111/PGMRS/E34 started on 03/04/90 at 14:35:02 in subsystem QINTER in
Message CPF4101 in XMPLDUMP in COBOLEX (C D F G).

Type reply, press Enter.
Reply . . . _____

F3=Exit  F12=Cancel

```

Figure 110. Run-Time Error Message

If you move the cursor to the line on which message number CPF4101 is indicated and press either the HELP key or F1, the LBE message information is displayed as shown:

```

                                Additional Message Information

Message ID . . . . . : LBE7200          Severity . . . . . : 99
Message type . . . . . : INQUIRY
Date sent . . . . . : 03/04/90          Time sent . . . . . : 14:37:15
From program . . . . . : QLREXHAN       Instruction . . . . . : 0000
To program . . . . . : *EXT             Instruction . . . . . : 0000

Message . . . . . : Message CPF4101 in XMPLDUMP in COBOLEX (C D F G).
Cause . . . . . : Message CPF4101 was detected in COBOL statement .OPEN (MI
instruction 007E) in program XMPLDUMP in COBOLEX.
Recovery . . . . . : Enter a G to continue the program at the next MI
instruction, or a C if no dump is wanted, a D if a dump of the COBOL
identifiers is wanted, or and F to dump both the COBOL identifiers and the
compiler-generated variables. The message text for CPF4101 follows: File
SALES in library *LIBL not found or inline data file missing.
Possible choices for replying to message . . . . . :
C -- No formatted dump is given
D -- A dump of the COBOL identifiers is given
F -- A dump of all variables is given
G -- To continue the program at the next MI instruction.

Press Enter to continue.

F3=Exit          F10=Display messages in job log          F12=Cancel

```

Figure 111. Run-Time Error Message—Second-Level Text

LBE messages 7900 to 7999 are used as headings for information printed during a COBOL/400 formatted dump.

The LBL messages are described under “Compilation Messages” below.

“Responding to Messages” on page 325 explains how to display second-level message text and how to reply to messages.

Compilation Messages

LBL messages are printed in the program listing when errors are found during program compilation. The LBL messages include the message issued when Federal Information Processing Standard (FIPS) flagging is requested; for more information on the FIPS messages, refer to page 327 in this appendix.

Program Listings

In the compiler output, the COBOL/400 messages listing follows the source listing. The COBOL/400 messages listing gives the message identifier, severity, text, usually the location of the error, and the messages summary.

For more information about Program Listings, see “Source Listing” on page 40.

Responding to Messages

In an interactive environment, a message is indicated by one or several of these conditions:

- A brief message (called first-level text) on the message line
- Reverse image highlighting of the input field in error
- A locked keyboard
- The sound of an alarm (if the alarm option is installed).

The following paragraphs briefly describe some methods of responding to error messages; more information is available in the *New User’s Guide* and the *Application Development Tools* publications.

If the necessary correction is obvious from the initial display, you can press the Error Reset key (if the keyboard is locked), enter the correct information, and continue your work.

If the message requires that you choose a reply (such as **C** to cancel, **D** to dump COBOL identifiers, **F** to dump all variables, or **G** to resume processing at the next COBOL statement), the reply options are shown in parentheses in the first-level message text. For an example, see Figure 110 on page 324.

If the information on the initial information display does not provide sufficient data for you to handle the error, you can press the HELP key (after positioning the cursor to the message line, if required) to get a second-level display with additional information about how to correct this error. To return to the initial display, press the Enter key; then press the Error Reset key (if the keyboard is locked), and make your correction or response.

If the error occurs when you are compiling or running a program, you might need to modify your COBOL/400 source statements or control language (CL) commands. Refer to the *SEU User’s Guide and Reference* for information on how to change the statements.

COBOL Message Descriptions

The messages for the COBOL/400 licensed program begin with prefixes LSC, LBE, or LBL.

The LSC messages are issued by the COBOL syntax checker when SEU is used to enter your COBOL source.

The LBE messages provide you with additional information about system operation during run time.

The LBL messages are compiler-generated messages.

Message numbers are assigned as follows:

Error Message	Description
LBE7000 through LBE7199	Escape Messages
LBE7200 through LBE7999	Run-time messages
LBE9001	Escape message
LBL0000 through LBL0999	Messages with severity less than 30
LBL1000 through LBL1999	Messages with severity greater than or equal to 30
LBL8000 through LBL8799	FIPS Flagger messages
LBL8800 through LBL8899	SAA Flagging messages
LSC0000 through LSC1999	Syntax checker messages

Severity Levels

The COBOL/400 licensed program provides the following message severity levels:

Severity	Meaning
00	Informational: This level is used to convey information to the user. No error has occurred. Informational messages are listed only when the FLAG (00) option is specified.
10	Warning: This level indicates that an error was detected but is not serious enough to interfere with the running of the program.
20	Error: This level indicates that an error was made, but the compiler is taking a recovery that might yield the desired code.
30	Severe Error: This level indicates that a serious error was detected. Compilation is completed, but running of the program cannot be attempted.
40	Unrecoverable: This level usually indicates a user error that forces termination of processing.
50	Unrecoverable: This level usually indicates a compiler error that forces termination of processing.
99	Action: Some manual action is required, such as entering a reply, changing printer forms, or replacing diskettes.

Note: 00, 10, and 20 messages are suppressed when the FLAG(30) option of the PROCESS statement is used or the CRTCLPGM command specifies FLAG(30) and is not overridden by the PROCESS statement. See "Using the PROCESS Statement to Specify Compiler Options" on page 31 for further information.

The compiler always attempts to provide full diagnostics of all source text in the program, even when errors have been detected. If the compiler cannot continue on a given statement, the message states that the compiler cannot continue and that it will ignore the rest of the statement. When this error occurs, the programmer should examine the entire statement.

The OS/400 message facility is used to produce all messages. The COBOL/400 compiler messages reside in the message file, QLBLMSG, and the run-time messages reside in the message file, QLBLMSGE.

Substitution variables and valid reply values are determined by the program sending the message, *not* by the message description stored in the message file. However, certain elements of a message description can be changed: for example, the text, severity level, default response, or dump list. To effect such changes, you need to define another message description using an Add Message Description (ADDMSGD) command, place the modified description in a user-created message file,¹ and specify that file in the Override Message File (OVRMSGF) command. Using the OVRMSGF command allows the compiler to retrieve messages from the specified file. See the ADDMSGD and OVRMSGF commands in the *CL Reference* for additional information.

CAUTION: Overriding an IBM-supplied message with a user-created message can produce results you do not anticipate. If reply values are not retained, the program might not respond to any replies. Changing default replies on *NOTIFY type messages could affect the ability of the program to run in unattended mode. Changing the severity could cancel a job not previously canceled. Be cautious when overriding IBM-supplied messages with user-created messages.

The Federal Information Processing Standard (FIPS) Flagger

The FIPS flagger can be specified to monitor a FIPS COBOL subset, any of the optional modules, all of the obsolete language elements, or a combination of a FIPS COBOL subset, optional modules and all obsolete elements.

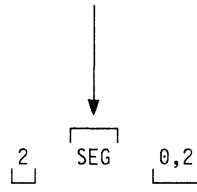
The monitoring is an analysis that compares the syntax used in the source program with the syntax included in the user-selected FIPS subset and optional modules. Any syntax used in the source program that does not conform to the selected FIPS COBOL subset and optional modules is identified. Any syntax for an obsolete language element used in the source program will also be identified (depending on the compiler option chosen). See page 25 for more information on the parameters for FIPS flagging.

1986 FIPS COBOL specifications are the language specifications contained in ANSI X3.23-1985 COBOL. FIPS COBOL is subdivided into three subsets and four optional modules. The three subsets are identified as Minimum, Intermediate and High. The four optional modules are Report Writer, Communication, Debug, and Segmentation. These four optional modules are not an integral part of any of the subsets; however, none, all, or any combination of the optional modules may be associated with any of the subsets. Any program written to conform to the 1986 FIPS standard must conform to one of the subsets of 1986 FIPS COBOL. Table 10 on page 328 shows the 1985 ANSI Standard COBOL processing modules included in each of the subsets of 1986 FIPS COBOL.

¹ If an IBM-supplied message must be changed and replaced in *its* message file, call your service representative.

Following is an explanation of the notation used within the table:

A 3-character code that identifies the module.
In this example, the Segmentation module, is
referenced.



The level of this module supported by
the 1986 FIPS COBOL standard. In this
example, support is provided for the
higher of the two levels of the
Segmentation module.

The **range** of levels of support
defined by the ANSI X3.23-1985 COBOL
standard. A level of 0 means a **minimum**
standard COBOL does not need to support
this module to conform to the standard.

1985 ANSI Module Name	High FIPS	Intermediate FIPS	Minimum FIPS
Nucleus	2 NUC 1,2	1 NUC 1,2	1 NUC 1,2
Sequential I-O	2 SEQ 1,2	1 SEQ 1,2	1 SEQ 1,2
Relative I-O	2 REL 0,2	1 REL 0,2	0 REL 0,2
Indexed I-O	2 INX 0,2	1 INX 0,2	0 INX 0,2
Source-Text Manipulation	2 STM 0,2	1 STM 0,2	0 STM 0,2
Sort-Merge	1 SRT 0,1	1 SRT 0,1	0 SRT 0,1
Inter-Program Communication	2 IPC 1,2	1 IPC 1,2	1 IPC 1,2
Report Writer	0, or 1 RPW 0,1	0, or 1 RPW 0,1	0, or 1 RPW 0,1
Segmentation	0,1 or 2 SEG 0,2	0,1 or 2 SEG 0,2	0,1 or 2 SEG 0,2
Debug	0,1 or 2 DEB 0,2	0,1 or 2 DEB 0,2	0,1 or 2 DEB 0,2
Communications	0,1 or 2 COM 0,2	0,1 or 2 COM 0,2	0,1 or 2 COM 0,2

Note: The COBOL/400 compiler supports the Segmentation and Debug optional modules.

Elements that are specified in the COBOL/400 source program and that are not included in 1986 FIPS COBOL are flagged as described in Appendix C, "Level of Language Support" on page 319.

SAA Flagging

You can choose to perform SAA flagging to determine if the COBOL/400 functions that you are using are portable to other SAA COBOL environments.

Flagging is performed on those COBOL/400 functions that are *outside* of SAA COBOL, such as:

- COBOL/400 extensions
- COBOL/400 compiler limits
- Non-SAA reserved words
- Compiler options.

In this way, you can write programs that conform to the SAA COBOL definition.

For an example of SAA flagging in a compiler listing, see Figure 12 on page 46. To perform SAA flagging through the CRTCLPGM CL command, specify SAAFLAG(*FLAG). To perform SAA flagging through a PROCESS statement, specify SAAFLAG.

To compile a program to conform to the SAA definition, using the CRTCLPGM command, specify the following:

```
OPTION(*QUOTE *NOSEQUENCE *NONUMBER)
GENOPT(*CRTF *DUPKEYCHK *SYNC)
SAAFLAG(*FLAG)
```

If you use the PROCESS statement, specify the following:

```
QUOTE, NOSEQUENCE, NONUMBER, CRTF,
DUPKEYCHK, SYNC, SAAFLAG.
```

For more information about specifying the option for SAA flagging, see the SAAFLAG parameter on page 25, and the "Using the PROCESS Statement to Specify Compiler Options" on page 31.

For information about compiler limits, see the Compiler Limits appendix in the *COBOL/400* Reference*.

Appendix E. Differences Between ANSI 74 COBOL and ANSI 85 COBOL

This appendix identifies the ANSI 85 COBOL language elements that are incompatible with ANSI 74 COBOL. These items identify the changes and conditions that ANSI 74 COBOL users need to be aware of when migrating to ANSI 85 COBOL.

See "Industry Standards Used in Compiler Design" on page xiii for more information on ANSI 85 COBOL.

Migrating ANSI 74 COBOL Programs to ANSI 85 COBOL

The following are some of the new features or changes to ANSI 85 COBOL that could affect ANSI 74 COBOL programs:

- The keyword ALPHABET must precede alphabet-name within the alphabet-name clause of the SPECIAL-NAMES paragraph. An alphabet-name is a user-defined word in the SPECIAL-NAMES paragraph that names a character set or collating sequence.
- The relative key data item specified in the RELATIVE KEY phrase must not contain the PICTURE symbol "P."
- The ALPHABETIC class test is true for uppercase letters, lowercase letters, and the space character.
- When there is no next statement to be processed in a called program, an implicit EXIT PROGRAM is run.
- No two files in a MERGE statement can be specified in the SAME AREA or SAME SORT-MERGE AREA clause. The only files in a MERGE statement that can be specified in the SAME RECORD AREA clause are those associated with the GIVING phrase.
- Within the READ statement, the INTO phrase cannot be specified unless:
 - All records associated with the file and the data item specified in the INTO phrase are group items or elementary alphanumeric items, or only one record description is subordinate to the file description entry.
- Within the RETURN statement, the INTO phrase cannot be specified unless:
 - All records associated with the file and data item specified in the INTO phrase are group items or elementary alphanumeric items, or only one record description is subordinate to the sort-merge file description entry.
- File position indicator - the concept of a current record pointer has been changed to a file position indicator.
- Reserved words - new reserved words have been added.
- I/O status - new I/O status values have been added.
- Pseudo-text-1 on the COPY statement must not consist entirely of a separator comma or a separator semicolon.
- A data item appearing in the USING phrase of the Procedure Division header must not have a REDEFINES clause in its data description entry.

- If the FOOTING phrase is not specified, no end-of-page condition independent of the page overflow condition exists.
- The NO REWIND phrase cannot be specified in a CLOSE statement having the REEL/UNIT phrase.
- The CANCEL and STOP RUN statements close all open files.
- When a receiving item is a variable-length data item and contains the object of the DEPENDING ON phrase, the maximum length of the item will be used.
- Within the VARYING ... AFTER phrase of the PERFORM statement, identifier-2 is augmented before identifier-5 is set.
- Any subscripts for identifier-4 in the DIVIDE statement REMAINDER phrase are evaluated after the result of the DIVIDE operation is stored in identifier-3 of the GIVING phrase.
- The phrase ADVANCING PAGE and END-OF-PAGE must not both be in a single WRITE statement.
- The picture character-string of an alphabetic item can contain only the symbol "A." No editing is allowed for the alphabetic data category.

Note: An alphabetic character is a letter or a space character.
- When a data item described by a PICTURE containing the character "P" is referenced, the digit positions specified by "P" are considered to contain zeros in the following operations:
 - Any operation requiring a numeric sending operand
 - A MOVE statement where the sending operand is numeric and its PICTURE character-string contains the symbol "P"
 - A MOVE statement where the sending operand is numeric edited and its PICTURE character-string contains the symbol "P" and the receiving operand is numeric or numeric edited
 - A comparison operation where both operands are numeric.
- The literal in the CURRENCY SIGN clause cannot be a figurative constant.
- If the COPY statement appears in a comment-entry, it is considered part of the comment-entry.
- The following special cases of exponentiation are defined:
 - If an expression having a zero value is raised to a negative or zero power, the size error condition exists.
 - If the evaluation of the exponentiation yields both a positive and a negative real number, the positive number is returned.
 - If no real number exists as the result of the evaluation, the size error condition exists.
- When the figurative constant ALL literal is not associated with another data item, the length of the string is the length of the literal.

Appendix F. Supporting International Languages with Double-Byte Character Sets

IBM Extension

This appendix describes only those enhancements made to the COBOL programming language for writing programs that process double-byte characters.

Specifically, this appendix describes where you can use Double-Byte Character Set (DBCS) characters in each portion of a COBOL program, and considerations for working with DBCS data in the COBOL/400 language.

There are two ways to specify DBCS characters:

- Bracketed-DBCS
- DBCS-graphic data

In general, COBOL handles bracketed-DBCS characters in the same way it handles alphanumeric characters. **Bracketed-DBCS** is a character string in which each character is represented by two bytes. The character starts with a shift-out (SO) character, and ends with a shift-in (SI) character. It is up to you to know (or have the COBOL program check) which data items contain DBCS characters, and to make sure the program receives and processes this information correctly.

You can now use DDS descriptions that define DBCS-graphic data fields with your COBOL/400 programs. **DBCS-graphic** pertains to a character string where each character is represented by two bytes. The character string does not contain shift-out or shift-in characters. You cannot use source programs containing graphic data. For information on specifying graphic data items with your COBOL/400 programs, refer to "DBCS-Graphic Fields" on page 129.

Using DBCS Characters in Literals

Types of Literals

There are two types of literals in which you can use DBCS characters: the DBCS literal and the mixed literal. A mixed literal consists of Double-Byte Character Set (DBCS) and Single-Byte Character Set (SBCS) characters.

DBCS Literals: The COBOL compiler recognizes DBCS characters in DBCS literals when you use the GRAPHIC option on the PROCESS statement.

Note: The GRAPHIC option on the PROCESS statement is not to be confused with the *GRAPHIC value in the CVTOPT parameter of the CRTCLPGM command and the CVTGRAPHIC option on the PROCESS statement, which are used to specify double-byte graphic data from a DDS description. For more information on specifying graphic data, refer to "DBCS-Graphic Fields" on page 129.

DBCS/SBCS Literals: The COBOL compiler recognizes DBCS characters in DBCS/SBCS (mixed) literals, when you are on a DBCS system and the GRAPHIC option on the PROCESS statement is not specified.

How to Specify Literals Containing DBCS Characters

When you specify any literal that contains DBCS characters, follow the same rules that apply in specifying alphanumeric literals, as well as the following rules specific to the literal types:

How to Specify a DBCS Literal: When you specify a DBCS literal, keep in mind the following:

The format for a DBCS literal is:

```
"0E K1K20F"
```

- A quotation mark opens and closes the literal.
- A shift-out character (0E) immediately follows the initial quotation mark and occupies 1 byte. A **shift-out** character is a control character (hex 0E) that indicates the start of a string of double-byte characters.
- A shift-in character (0F) immediately precedes the final quotation mark and occupies 1 byte. A **shift-in** character is a control character (hex 0F) that indicates the end of a string of double-byte characters.
- All DBCS characters appear between the shift-out and shift-in characters.
- Only DBCS characters may appear in the literal (null strings are valid).

The maximum length of a DBCS literal is 80 DBCS characters, including the shift control characters. (These counted together are equivalent in length to one DBCS character.) The shift control characters are part of the literal, and take part in all operations.

See "How to Continue DBCS Literals on a New Line" on page 335 for information on how to extend DBCS literals.

How to Specify a DBCS/SBCS Literal: When you specify a DBCS/SBCS literal, keep in mind the following:

- DBCS/SBCS literals can take many different forms. The following is only one possible example:

```
"SINGLE0EK1K2K30F BYTES"
```
- USAGE DISPLAY must be either explicit or implicit.
- A quotation mark opens and closes the literal.
- EBCDIC characters can appear before or after any DBCS string in the mixed literal.
- All DBCS strings appear between shift-out and shift-in characters.
- Double all SBCS quotation marks that occur within the literal. DBCS quotation marks within the literal do not require doubling.
- You can use null DBCS strings (shift-out and shift-in characters without any DBCS characters) *only* when the literal contains at least one SBCS character.

The shift-out and shift-in characters cannot be nested.

The shift control characters are part of the literal, and take part in all operations.

DBCS/SBCS literals cannot continue across lines. They are restricted to the space of AREA B on one line.

Other Considerations

Quotation Marks: Although the preceding discussion uses the term a *quotation mark* to describe the character that identifies a literal, the character actually used can vary depending upon the option specified on the CRTCLPGM CL command, or on the PROCESS statement. If you specify the APOST option, an apostrophe (') is used. Otherwise, a quotation mark (") is used. In this appendix, a *quotation mark* refers to both an apostrophe and a quotation mark. The character that you choose does not affect the rules for specifying a literal.

Shift Characters: The shift-out and shift-in characters separate EBCDIC characters from DBCS characters. They are part of both the DBCS and the DBCS/SBCS literal. Therefore, the shift code characters participate in all operations when they appear in either DBCS or DBCS/SBCS literals.

How the COBOL Compiler Checks DBCS Characters

When the COBOL compiler finds a DBCS string, it checks the DBCS string by scanning it one DBCS character at a time.

The following conditions cause the COBOL compiler to diagnose a literal containing DBCS characters as *not valid*:

- The syntax for the literal is incorrect.
- The DBCS literal is longer than one line and does *not* follow the rules for continuing nonnumeric literals. (See "How to Continue DBCS Literals on a New Line" for more information.)
- The DBCS/SBCS literal is longer than one line.

When the COBOL compiler finds a DBCS literal that is not valid, it generates an error message, and then processes the literal as an alphanumeric literal.

For each DBCS or SBCS literal that is not valid, the compiler generates an error message and accepts or ignores the literal.

How to Continue DBCS Literals on a New Line

To continue a DBCS literal onto another line of source code, do *all* of the following:

- Place a shift-in character in either column 71 or column 72 of the line to be continued (If you put the shift-in character in column 71, the blank in column 72 is ignored)
- Place a hyphen (-) in column 7 (the continuation area) of the new line
- Place a quotation mark, then a shift-out character, and then the rest of the literal in Area B of the new line.

For example:

```
-A 1 B
  ⋮
01 DBCS1          PIC X(12)          VALUE "0_K1K2K30_
-   "0_K4K50_".
  ⋮
```

The value of DBCS1 is "0_K1K2K3K4K50_".

The shift-in character, quotation mark, and shift-out character used to continue a line are not counted in the length of the DBCS literal. The first shift-out and final shift-in characters are counted.

Where You Can Use DBCS Characters in a COBOL Program

In general, you can use DBCS, or DBCS/SBCS literals wherever nonnumeric literals are allowed. Literals for the following, however, cannot include double-byte characters:

- ALPHABET-NAME clause
- CURRENCY SIGN clause
- ASSIGN clause
- CLASS clause
- CALL statement
- CANCEL statement.

Note: You cannot use DBCS characters for COBOL words or names. See the *COBOL/400* Reference* for information on rules for formatting COBOL system-names, reserved words, and user-defined words such as data names and file names.

How to Write Comments

You can write comments containing DBCS characters in a COBOL program by putting an asterisk (*) or slash (/) in column seven of the program line. Either symbol causes the compiler to treat any information following column seven as documentation. The slash also causes a page eject. Because the COBOL compiler does not check the contents of comment lines, DBCS characters in comments are not detected. DBCS characters that are not valid can cause the compiler listing to print improperly.

Identification Division

You can put comment entries that contain DBCS characters in any portion of the Identification Division except the PROGRAM-ID paragraph. The program name specified in the PROGRAM-ID paragraph must be alphanumeric.

Environment Division

Configuration Section

You can use DBCS characters in comment entries only in the Configuration Section paragraph. All function-names, mnemonic-names, condition-names, and alphabet-names must be specified with alphanumeric characters. For the SOURCE-COMPUTER and the OBJECT-COMPUTER entry, use the alphanumeric computer name:

IBM-AS400

You cannot use DBCS or DBCS/SBCS literals in the Configuration Section. Instead, use alphanumeric literals to define an alphabet-name and the literal in the CURRENCY SIGN clause of the SPECIAL-NAMES paragraph. There is no DBCS alphabet. Use the EBCDIC character set instead.

Input-Output Section

Specify all data names, file names, and assignment names using alphanumeric characters. You can use DBCS characters in comments.

For indexed files, the data name in the RECORD KEY clause can refer to a DBCS or DBCS/SBCS data item within a record. The number of fields in the record, plus the number of positions occupied by the record key, together cannot be greater than 120.

Note: Each DBCS character occupies two positions, and the shift control characters each occupy one position. Ensure that both the data description of the key and the key position within the file match those specified when you created the file.

You cannot use DBCS and DBCS/SBCS data as the RELATIVE KEY in relative files.

File Control Paragraph

ASSIGN Clause: You cannot use literals containing DBCS characters in the ASSIGN clause to specify an external medium such as a printer or a database.

Data Division

File Section

For the FD (File Description) Entry, you can use DBCS or DBCS/SBCS data items or literals in the VALUE OF clause. The DATA RECORDS clause can refer to data items only. Because the COBOL/400 compiler treats both the VALUE OF clause and the DATA RECORDS clause in the File Section as documentation, neither clause has any effect when you run the program. However, the COBOL compiler checks all literals in the VALUE OF clause to make sure they are valid.

For magnetic tapes, the system can only read DBCS characters from, or write DBCS characters to, the tape in the EBCDIC format. The system cannot perform tape functions involving a tape in the ASCII format. Define the alphabet-name in the CODE-SET clause as NATIVE. Use alphanumeric characters to specify the alphabet-name.

Working-Storage Section

REDEFINES Clause: The existing rules for redefining data also apply to data that contains DBCS characters. When you determine the length of a redefining or redefined data item, remember that each DBCS character is twice as long as an alphanumeric character.

Also, ensure that redefined data items contain the shift control characters when and where necessary.

OCCURS Clause: Use this clause to define tables for storing DBCS or DBCS/SBCS data. If you specify the ASCENDING/DESCENDING KEY phrase, COBOL assumes the contents of the table are in the EBCDIC program collating sequence. The shift control characters in DBCS and DBCS/SBCS data take part in the collating sequence.

For more information about handling tables that contain DBCS characters, see “Table Handling – SEARCH Statement” on page 344.

JUSTIFIED RIGHT Clause: Use the JUSTIFIED RIGHT clause to align DBCS or DBCS/SBCS data at the rightmost position of an elementary receiving field. If the receiving field is shorter than the sending field, COBOL truncates the rightmost characters. If the receiving field is longer than the sending field, COBOL pads (fills) the unused space on the left of the receiving field with blanks.

The JUSTIFIED clause does not affect the initial setting in the VALUE clause.

VALUE Clause: You can use DBCS or DBCS/SBCS literals to specify an initial value for a data item that is not numeric, or to define values for level-88 condition-name entries.

Any shift control characters in the literal are considered part of the literal's picture string, except when used to continue a new line. When you continue a DBCS literal, the compiler does *not* include the shift-in character in column 71 or 72, or the initial quotation mark (") and shift-out character on the continued line as part of the DBCS literal. Make certain, however, that the DBCS literal does not exceed the size of the data item specified in the PICTURE clause, otherwise truncation occurs.

Note: DBCS/SBCS mixed literals cannot be continued to a new line.

When you use literals that contain DBCS characters in the VALUE clause for level-88 condition-name entries, COBOL treats the DBCS characters as alphanumeric. Therefore, follow the rules for specifying alphanumeric data, including allowing a THROUGH option. This option uses the normal EBCDIC collating sequence, but remember that shift control characters in DBCS and DBCS/SBCS data take part in the collating sequence.

PICTURE Clause: Use the PICTURE symbol X to define DBCS and DBCS/SBCS data items. Because DBCS characters are twice as long as alphanumeric, and are enclosed within shift control characters, you would define a DBCS data item containing n DBCS characters as

```
PICTURE X(2n+2)
```

A DBCS/SBCS data item containing m SBCS characters, and one string of n DBCS characters would be defined as

```
PICTURE X(m+2n+2)
```

You can use all edited alphanumeric PICTURE symbols for DBCS and DBCS/SBCS data items. The editing symbols have the same effect on the DBCS data in these items as they do on alphanumeric data items. Check that you have obtained the desired results.

RENAMES Clause: Use this clause to specify alternative groupings of elementary data items. The existing rules for renaming alphanumeric data items also apply to DBCS and DBCS/SBCS data items.

Procedure Division

Declaratives

An identifier in the USE FOR DEBUGGING sentence of the DECLARATIVES section can refer to a DBCS or a DBCS/SBCS data item.

You cannot use DBCS characters for file names or procedure names in the USE FOR DEBUGGING sentence.

Conditional Expressions

Because condition-names (level-88 entries) can refer to data items that contain DBCS characters, you can use the condition-name condition to test this data. (See "VALUE Clause" on page 338.) Follow the rules listed in the *COBOL/400* Reference* for using conditional variables and condition-names.

You can use DBCS or DBCS/SBCS data items or literals as the operands in a relation condition. Because COBOL treats DBCS data as alphanumeric, all comparisons occur according to the rules for alphanumeric operands. Keep the following in mind:

- The system does not recognize the mixed content.
- The system uses the shift codes in comparisons of DBCS and DBCS/SBCS data.
- The system compares the data using either the EBCDIC collating sequence, or a user-defined sequence.
- In a comparison of DBCS or DBCS/SBCS items with similar items of unequal size, the smaller item is padded on the right with EBCDIC spaces.

See "SPECIAL-NAMES Paragraph" section in the *COBOL/400* Reference* for more information.

You can use class conditions and switch status conditions as described in the *COBOL/400* Reference*.

Input/Output Statements

ACCEPT Statement: The input data received from a device by using a Format 1 ACCEPT statement can include DBCS or DBCS/SBCS data. All DBCS and DBCS/SBCS data must be identified by the proper syntax. The input data, including shift control characters, replaces the existing contents of the identifier. COBOL does not perform editing or error checking on the data.

If you use the Format 3 ACCEPT statement to get OPEN-FEEDBACK information about a file, that information includes a field showing whether the file has DBCS or DBCS/SBCS data.

Information received from the local data area by a Format 4 ACCEPT statement can include DBCS or DBCS/SBCS character strings. Information received replaces the existing contents. COBOL does not perform any editing or checking for errors. This also applies to information received from the PIP data area by a Format 5 ACCEPT statement.

Using the Format 6 ACCEPT statement, you can get the attributes of a work station display and its keyboard. For display stations that can display DBCS characters, the system sets the appropriate value in the ATTRIBUTE-DATA data item. You cannot use DBCS characters to name a device.

If you use an extended (Format 7) ACCEPT statement for field-level work station input, you must ensure that DBCS data is not split across lines. COBOL does not perform any editing or checking for errors.

DISPLAY Statement: You can specify DBCS or DBCS/SBCS data items or literals in the DISPLAY statement. You can mix the types of data. DBCS and DBCS/SBCS data, from either data items or literals, is sent as it appears to the program device or local data area that is the target named on the DISPLAY statement.

Because COBOL does not know the characteristics of the device on which data is being displayed, you must make sure that the DBCS and DBCS/SBCS data is correct.

Note: ALL is a valid option for mixed literals.

If you use an extended (Format 3) DISPLAY statement for field-level work station output, you must ensure that DBCS data is not split across lines.

READ Statement: You can use DBCS or DBCS/SBCS data items as the RECORD KEY for an indexed file. See "Input-Output Section" on page 337 for more information.

INTO Phrase: You can read a record into a DBCS or a DBCS/SBCS data item using the INTO phrase. This phrase causes a MOVE statement (without the CORRESPONDING option) to be performed. The compiler moves DBCS and DBCS/SBCS data in the same manner that it moves alphanumeric data. It does not make sure that this data is valid.

REWRITE Statement: Use the FROM phrase of this statement to transfer DBCS or DBCS/SBCS data from a DBCS or a DBCS/SBCS data item to an existing record. The FROM phrase causes both types of data to be moved in the same manner as the INTO phrase with the READ statement. (See "READ Statement.")

START Statement: If you use DBCS characters in the key of an indexed file, specify a corresponding data item in the KEY phrase of the START statement.

One of the following must be true:

- The data item must be the same as the data item specified in the RECORD KEY clause of the FILE-CONTROL paragraph.
- The data item has the same first character as the record key and is not longer than the record key.

You can specify valid operators (such as EQUAL, GREATER THAN, NOT LESS THAN) in the KEY phrase. The system can follow either the EBCDIC or a user-defined collating sequence.

WRITE Statement: Use the FROM phrase of this statement to write DBCS or DBCS/SBCS data to a record. This phrase moves the data in the same manner as the REWRITE statement. (See "REWRITE Statement.")

You must include the shift control characters when you write the data into a device file.

Data Manipulation Statements

Arithmetic Statements: Because COBOL treats DBCS characters in the same manner that it treats alphanumeric characters, do not use DBCS characters in numeric operations, nor manipulate them with arithmetic statements.

INSPECT Statement: You can use any DBCS or DBCS/SBCS data item as an operand for the INSPECT statement. The system tallies and replaces on each half of a DBCS character, including the shift control characters in these operations. Therefore, the data may not be matched properly.

You can use any combination of double-byte character and alphanumeric operands and double-byte character literals or data items. If you use the REPLACING phrase, you might cause parts of the inspected item to be replaced by alphanumeric data, or vice versa.

You cannot replace a character string with a string of a different length. Consider this when replacing alphanumeric characters with DBCS characters, or vice versa.

If you want to control the use of the INSPECT statement with items containing DBCS characters, define data items containing shift control characters. Use the shift-out and shift-in characters as BEFORE/AFTER operands in the INSPECT statement.

The following example shows how you can use the INSPECT statement to replace one DBCS character with another.

```
01 SUBJECT-ITEM          PICTURE X(50).
01 DBCS-CHARACTERS      VALUE "0_K1K20_F".
   05 SHIFT-OUT         PICTURE X.
   05 DBCS-CHARACTER-1  PICTURE XX.
   05 DBCS-CHARACTER-2  PICTURE XX.
   05 SHIFT-IN         PICTURE X.
```

The INSPECT statement would be coded as follows:

```
INSPECT SUBJECT-ITEM
  REPLACING ALL DBCS-CHARACTER-1
    BY DBCS-CHARACTER-2
  AFTER INITIAL SHIFT-OUT.
```

Note: Using the AFTER INITIAL SHIFT-OUT phrase helps you to avoid the risk of accidentally replacing two consecutive alphanumeric characters that have the same EBCDIC values as DBCS-CHARACTER-1 (in cases where SUBJECT-ITEM contains DBCS/SBCS data).

You can also use the INSPECT statement to determine if a data item contains DBCS characters, so that appropriate processing can occur. For example:

```
01 SUBJECT-FIELD        PICTURE X(50).
01 TALLY-FIELD          PICTURE 9(3) COMP.
01 SHIFTS               VALUE "0_E0_F".
   05 SHIFT-OUT         PICTURE X.
   05 SHIFT-IN         PICTURE X.
```

In the Procedure Division you might enter the following:

```

MOVE ZERO TO TALLY-FIELD.
INSPECT SUBJECT-FIELD TALLYING TALLY-FIELD
                        FOR ALL SHIFT-OUT.
IF TALLY-FIELD IS GREATER THAN ZERO THEN
    PERFORM DBCS-PROCESSING
ELSE
    PERFORM A-N-K-PROCESSING.

```

MOVE Statement: All DBCS characters are moved as alphanumeric character strings. The system does not convert the data or examine it.

You can move DBCS/SBCS literals to group items and alphanumeric items.

If the length of the receiving field is different from that of the sending field, COBOL does one of the following:

- Truncates characters from the sending item if it is longer than the receiving item. This operation can reduce data integrity.
- Pads the sending item with blanks if it is shorter than the receiving item.

To understand more about the effect of editing symbols in the PICTURE clause of the receiving data item, see the *COBOL/400* Reference*.

SET Statement (Condition-Name Format): When you set the condition name to TRUE on this statement, COBOL moves the literal from the VALUE clause to the associated data item. You can move a literal with DBCS characters.

STRING Statement: You can use the STRING statement to construct a data item that contains DBCS or DBCS/SBCS subfields. All data in the source data items or literals, including shift control characters, is moved to the receiving data item, one-half of a DBCS character at a time.

UNSTRING Statement: The UNSTRING statement treats DBCS data and DBCS/SBCS data the same as alphanumeric data. The UNSTRING operation is performed on one-half of a DBCS character at a time.

Data items can contain both alphanumeric and DBCS characters within the same field.

Use the DELIMITED BY phrase to locate double-byte and alphanumeric subfields within a data field. Identify the data items containing shift control characters, and use those data items as identifiers on the DELIMITED BY phrase. See the following examples for more information on how to do this. Use the POINTER variable to continue scanning through subfields of the sending field.

After the system performs the UNSTRING operation, you can check the delimiters stored by the DELIMITER IN phrases against the shift control character values to see which subfields contain DBCS and which contain alphanumeric characters.

The following example shows how you might set up fields to prepare for the unstring operation on a character string that contain DBCS/SBCS data:

```

01 SUBJECT-FIELD      PICTURE X(40)
01 FILLER.
   05 UNSTRING-TABLE OCCURS 4 TIMES.
       10 RECEIVER    PICTURE X(40).
       10 DELIMTR     PICTURE X.
       10 COUNTS      PICTURE 99 COMP.
01 SHIFTS              VALUE "0E0F".
   05 SHIFT-OUT       PICTURE X.
   05 SHIFT-IN        PICTURE X.

```

Code the UNSTRING statement as follows:

```

UNSTRING SUBJECT-FIELD  DELIMITED BY SHIFT-OUT
                        OR SHIFT-IN
INTO RECEIVER (1) DELIMITER IN DELIMTR (1)
                   COUNT      IN COUNTS (1)
INTO RECEIVER (2) DELIMITER IN DELIMTR (2)
                   COUNT      IN COUNTS (2)
INTO RECEIVER (3) DELIMITER IN DELIMTR (3)
                   COUNT      IN COUNTS (3)
INTO RECEIVER (4) DELIMITER IN DELIMTR (4)
                   COUNT      IN COUNTS (4)
ON OVERFLOW PERFORM UNSTRING-OVERFLOW-MESSAGE.

```

This UNSTRING statement divides a character string into its alphanumeric and DBCS parts. Assuming that the data in the character string is valid, a delimiter value of shift-out indicates that the corresponding receiving field contains alphanumeric data, while a value of shift-in indicates that corresponding receiving field has DBCS data. You can check the COUNT data items to determine whether each receiving field received any characters. The following figure is an example that shows the results of the UNSTRING operation just described:

```

SUBJECT-FIELD = ABC0EK1K2K30FD0EK4K5K60F
RECEIVER (1) = ABC      DELIMTR (1) = 0E   COUNTS (1) = 3
RECEIVER (2) = K1K2K3   DELIMTR (2) = 0F   COUNTS (2) = 6
RECEIVER (3) = D        DELIMTR (3) = 0E   COUNTS (3) = 1
RECEIVER (4) = K4K5K6   DELIMTR (4) = 0F   COUNTS (4) = 6

```

```

SUBJECT-FIELD = 0EK1K2K30FABC0EK40F
RECEIVER (1) = (blanks) DELIMTR (1) = 0E   COUNTS (1) = 0
RECEIVER (2) = K1K2K3   DELIMTR (2) = 0F   COUNTS (2) = 6
RECEIVER (3) = ABC      DELIMTR (3) = 0E   COUNTS (3) = 3
RECEIVER (4) = K4       DELIMTR (4) = 0F   COUNTS (4) = 2

```

Procedure Branching Statements

You can use either a DBCS or a DBCS/SBCS literal as the operand for the STOP statement. When you do, the system displays the literal as you entered it at your work station for interactive jobs. For batch jobs, the system displays underscores where the literal would normally appear on the system operator's message queue. The system does not edit or check the contents of the literal.

Table Handling – SEARCH Statement

You can perform a Format 1 SEARCH statement (sequential search of a table) on a table that contains DBCS or DBCS/SBCS data half a DBCS character at a time.

You can also perform a Format 2 SEARCH statement (SEARCH ALL) against a DBCS or DBCS/SBCS table as well. Order the table according to the chosen collating sequence.

Note: The shift control characters in DBCS and DBCS/SBCS data participate in the comparison.

SORT/MERGE

You cannot perform a DBCS alphabet sort using COBOL. However, you can use DBCS or DBCS/SBCS data items as keys in a SORT or MERGE statement. The sort operation orders data according to the collating sequence specified in the SORT, MERGE, or SPECIAL NAMES paragraph. The system orders any shift control characters contained in DBCS and DBCS/SBCS keys.

Use the RELEASE statement to transfer records containing DBCS characters from an input/output area to the initial phase of a sort operation. The system performs the FROM phrase with the RELEASE statement in the same way it performs the FROM phrase with the WRITE statement. (See “WRITE Statement” on page 340.)

You can also use the RETURN statement to transfer records containing DBCS characters from the final phase of a sort or merge operation to an input/output area. The system performs the INTO phrase with the RETURN statement in the same manner that it performs the INTO phrase with the READ statement. (See “READ Statement” on page 340.)

Compiler-Directing Statements

COPY Statement

You can use the COPY statement to copy source text that contains DBCS characters into a COBOL program. When you do, make sure that you specify the member name, file name, and library name using alphanumeric data, and that you specify these names according to the rules stated in the *COBOL/400* Reference*.

Use the Format 2 COPY statement to copy fields defined in the data description specifications (DDS). DBCS and DBCS/SBCS data items (the value in column 35 of the DDS form is O) are copied into a COBOL program in the PICTURE X(n) format. The compiler listing does not indicate that these fields contain DBCS characters, unless a field is a key field. In those cases, the system prints an O in the comment table for keys.

DBCS-graphic data items are copied into a COBOL program in the PICTURE X(N) format. The compiler listing indicates that these fields contain graphic data. See “DBCS-Graphic Fields” on page 129 for a description of the DBCS-graphic data type.

You can put DBCS characters in text comments that are copied from DDS if the associated DDS field has comments.

If you specify the REPLACING phrase of the COPY statement, consider the following:

- Pseudo-text can contain any combination of DBCS and alphanumeric characters.
- You can use literals with DBCS or DBCS/SBCS content.
- Identifiers can refer to data items that contain DBCS characters.

TITLE Statement

You can use DBCS/SBCS literals as the literal in the TITLE statement.

Communications between Programs

You can specify entries for DBCS or DBCS/SBCS data items in the Linkage Section of the Data Division.

You can pass DBCS characters from one program to another program by specifying those data items in the USING phrase. You cannot use DBCS characters in the CALL statement for the program-name of the called program.

You cannot use DBCS characters in the CANCEL statement because they specify program-names.

FIPS Flagger

Enhancements to the COBOL language that let you use DBCS characters are flagged (identified) by the FIPS (Federal Information Processing Standard) flagger provided by the compiler as IBM extensions.

COBOL Program Listings

DBCS characters can appear in listings that originate from DBCS-capable source files, and that are produced on DBCS-capable systems.

DBCS characters that appear in a program listing originate from the source file, from source text generated by the COPY statement, or from COBOL compiler messages.

A listing containing DBCS characters should be output to a printer file that is capable of processing DBCS data. Listings containing DBCS characters are handled correctly if one of the following conditions is true:

- The printer file specified by the PRTFILE parameter of the CRTCLPGM command is defined with the required attribute, using the CRTPRTF or CHGPRTF command.
- The source file is defined as capable of containing DBCS data using the IGCDTA parameter of the CRTSRCPF command. In this case, the program overrides the existing value of the attribute for the output printer file.
- The user has specified the required attribute for the output printer, using the IGCDTA parameter of the OVRPRTF command, before compiling the program.

Note: The IGCDTA parameter is only available on DBCS systems, and it cannot be defined or displayed on non-DBCS systems. You can, however, create objects with DBCS attributes on a non-DBCS system by copying them from a DBCS system. You should check for possible incompatibilities if you do this.

The compiler may use characters from your source program as substitution parameters in compiler and syntax checker messages. The system does not check or edit the substitution parameters. If you do not specify DBCS characters properly, the system may print or display parts of messages incorrectly.

_____ End of IBM Extension _____

Appendix G. AS/400 File Processing Examples

This appendix contains sample programs that illustrate the fundamental programming techniques associated with each type of AS/400 file organization. These examples are intended to be used for planning purposes only, and to illustrate the input/output statements necessary for certain access methods. Other COBOL features (the use of the PERFORM statement, for example) are used only incidentally. The programs illustrated are:

- Sequential File Creation
- Sequential File Updating and Extension
- Indexed File Creation
- Indexed File Updating
- Relative File Creation
- Relative File Updating
- Relative File Retrieval.

Sequential File Creation

This program creates a sequential file of employee salary records. The input records are arranged in ascending order of employee number. The output file has the identical order. (An **output file** is a file that is opened in either the output mode or the extend mode.)

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B. ....2....3....4....5....6....7..IDENTFCN S COPYNAME  CHG DATE
 1 000010 IDENTIFICATION DIVISION.
 2 000020 PROGRAM-ID. CRTSEQ.
   000030
 3 000040 ENVIRONMENT DIVISION.
 4 000050 CONFIGURATION SECTION.
 5 000060 SOURCE-COMPUTER. IBM-AS400.
 6 000070 OBJECT-COMPUTER. IBM-AS400.
 7 000080 SPECIAL-NAMES. CONSOLE IS TYPEWRITER.
 8 000090 INPUT-OUTPUT SECTION.
 9 000100 FILE-CONTROL.
10 000110     SELECT INPUT-FILE ASSIGN TO DISK-FILEA
11 000120     FILE STATUS IS INPUT-FILE-STATUS.
12 000130     SELECT OUTPUT-FILE ASSIGN TO DISK-FILEB
13 000140     FILE STATUS IS OUTPUT-FILE-STATUS.
14 000150 DATA DIVISION.
15 000160 FILE SECTION.
16 000170 FD INPUT-FILE LABEL RECORDS STANDARD.
17 000180 01 INPUT-RECORD.
18 000190     05 INPUT-EMPLOYEE-NUMBER     PICTURE 9(6).
19 000200     05 INPUT-EMPLOYEE-NAME         PICTURE X(28).
20 000210     05 INPUT-EMPLOYEE-CODE     PICTURE 9.
21 000220     05 INPUT-EMPLOYEE-SALARY    PICTURE 9(6)V99.
22 000230 FD OUTPUT-FILE LABEL RECORDS STANDARD.
23 000240 01 OUTPUT-RECORD.
24 000250     05 OUTPUT-EMPLOYEE-NUMBER    PICTURE 9(6).
25 000260     05 OUTPUT-EMPLOYEE-NAME        PICTURE X(28).
26 000270     05 OUTPUT-EMPLOYEE-CODE     PICTURE 9.
27 000280     05 OUTPUT-EMPLOYEE-SALARY    PICTURE 9(6)V99.
28 000290 WORKING-STORAGE SECTION.
29 000300 77 INPUT-FILE-STATUS           PICTURE XX.
30 000310 77 OUTPUT-FILE-STATUS         PICTURE XX.
31 000320 01 INPUTEND                   PICTURE X VALUE SPACE.
32 000330     88 THE-END-OF-INPUT        VALUE "E".
33 000340 01 DISP-RECORD.
34 000350     05 OP-NAME                   PICTURE X(7).
35 000360     05 FILLER                     PICTURE XX VALUE SPACE.
36 000370     05 FILE-NAME                   PICTURE X(11).
37 000380     05 FILLER                     PICTURE XX VALUE SPACE.
38 000390     05 FILLER                     PICTURE X(14)
39 000400     VALUE "FILE STATUS IS".
40 000410     05 FILLER                     PICTURE XX VALUE SPACE.
41 000420     05 SK                         PICTURE XX.
42 000430 PROCEDURE DIVISION.
   000440 DECLARATIVES.
   000450 I-O-ERROR SECTION.
   000460     USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE,
   000470     OUTPUT-FILE.
   000480 I-O-ERROR-PARA.
   000490*****
   000500* DUMMY DECLARATIVES TO ENSURE CONTROL IS RETURNED TO THIS *
   000510* PROGRAM WHEN AN ERROR OCCURS DURING FILE PROCESSING.      *
   000520* ERROR HANDLING IS DONE AFTER EACH I/O STATEMENT.        *
   000530*****
   000540 END DECLARATIVES.
   000550 MAIN-PROGRAM SECTION.
   000560 OPEN-FILES.
43 000570     OPEN INPUT INPUT-FILE
   000580     OUTPUT OUTPUT-FILE.
44 000590     IF INPUT-FILE-STATUS NOT = "00"
45 000600         MOVE "OPEN" TO OP-NAME
46 000610         MOVE "INPUT-FILE" TO FILE-NAME
47 000620         MOVE INPUT-FILE-STATUS TO SK
48 000630         PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2.
49 000640     IF OUTPUT-FILE-STATUS NOT = "00"
50 000650         MOVE "OPEN" TO OP-NAME
51 000660         MOVE "OUTPUT-FILE" TO FILE-NAME
52 000670         MOVE OUTPUT-FILE-STATUS TO SK
53 000680         PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2.
54 000690     PERFORM BUILD-FILE UNTIL THE-END-OF-INPUT.
   000700 CLOSE-FILES.
55 000710     CLOSE INPUT-FILE
   000720     OUTPUT-FILE.
56 000730     STOP RUN.
   000740 BUILD-FILE.

```

Figure 112 (Part 1 of 2). Example of a Sequential File of Employee Salary Records


```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B.+. . . . 2. . . . +. . . . 3. . . . +. . . . 4. . . . +. . . . 5. . . . +. . . . 6. . . . +. . . . 7. . IDENTFCN S COPYNAME  CHG DATE
57 000750      READ INPUT-FILE INTO OUTPUT-RECORD
58 000760      AT END SET THE-END-OF-INPUT TO TRUE.
59 000770      IF INPUT-FILE-STATUS NOT = "00"
60 000780          MOVE "WRITE" TO OP-NAME
61 000790          MOVE "OUTPUT-FILE" TO FILE-NAME
62 000800          MOVE OUTPUT-FILE-STATUS TO SK
63 000810          PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2
64 000820          GO TO CLOSE-FILES.
65 000830      WRITE OUTPUT-RECORD.
66 000840      IF OUTPUT-FILE-STATUS NOT = "00"
67 000850          MOVE "WRITE" TO OP-NAME
68 000860          MOVE "OUTPUT-FILE" TO FILE-NAME
69 000870          MOVE OUTPUT-FILE-STATUS TO SK
70 000880          PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2
71 000890          GO TO CLOSE-FILES.
000900 ERROR-OUT-1.
72 000910          DISPLAY "FILE PROCESSING ERROR" UPON TYPEWRITER.
73 000920          DISPLAY DISP-RECORD UPON TYPEWRITER.
74 000930          CLOSE INPUT-FILE
000940              OUTPUT-FILE.
75 000950          STOP RUN.
000960 ERROR-OUT-2.
000970          EXIT.

          * * * * *  E N D  O F  S O U R C E  * * * * *
5738CB1 V2R2M0          AS/400 COBOL Messages
STMT
* 16 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000170
    Message . . . . : Blocking/Deblocking for file 'INPUT-FILE'
    will be performed by compiler-generated code.
* 22 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000230
    Message . . . . : Blocking/Deblocking for file 'OUTPUT-FILE'
    will be performed by compiler-generated code.
* 43 MSGID: LBL0335 SEVERITY: 00 SEQNBR: 000540
    Message . . . . : Empty paragraph or section precedes 'END
    DECLARATIVES' paragraph or section.
          * * * * *  E N D  O F  M E S S A G E S  * * * * *
          Message Summary
          Total      Info(0-4)      Warning(5-19)      Error(20-29)      Severe(30-39)      Terminal(40-99)
          3          3          0          0          0          0
Source records read . . . . . : 97
Copy records read . . . . . : 0
Copy members processed . . . . . : 0
Sequence errors . . . . . : 0
Highest severity message issued . . : 0
    LBL0901 00 Program CRTSEQ created in library XMPLIB.
          * * * * *  E N D  O F  C O M P I L A T I O N  * * * * *

```

Figure 112 (Part 2 of 2). Example of a Sequential File of Employee Salary Records

Sequential File Updating and Extension

This program updates and extends the file created by the CRTSEQ program. The INPUT-FILE and the MASTER-FILE are each read. When a match is found between INPUT-EMPLOYEE-NUMBER and MST-EMPLOYEE-NUMBER, the input record replaces the original record. After the MASTER-FILE is processed, new employee records are added to the end of the file.

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7...IDENTFCN S COPYNAME  CHG DATE
 1 000010 IDENTIFICATION DIVISION.
 2 000020 PROGRAM-ID. UPDTSEQ.
 3 000030 ENVIRONMENT DIVISION.
 4 000040 CONFIGURATION SECTION.
 5 000050 SOURCE-COMPUTER. IBM-AS400.                                05/24/91
 6 000060 OBJECT-COMPUTER. IBM-AS400.                                05/24/91
 7 000070 INPUT-OUTPUT SECTION.
 8 000080 FILE-CONTROL.
 9 000090     SELECT INPUT-FILE ASSIGN TO DISK-FILES
10 000100         FILE STATUS IS INPUT-FILE-STATUS.  A
11 000110     SELECT MASTER-FILE ASSIGN TO DISK-MSTFILEB
12 000120         FILE STATUS IS MASTER-FILE-STATUS.  B
   000130
13 000140 DATA DIVISION.
14 000150 FILE SECTION.
15 000160 FD INPUT-FILE LABEL RECORDS STANDARD.
16 000170 01 INPUT-RECORD.
17 000180     05 INPUT-EMPLOYEE-NUMBER      PICTURE 9(6).
18 000190     05 INPUT-EMPLOYEE-NAME      PICTURE X(28).
19 000200     05 INPUT-EMPLOYEE-CODE      PICTURE 9.
20 000210     05 INPUT-EMPLOYEE-SALARY    PICTURE 9(6)V99.
21 000220 FD MASTER-FILE LABEL RECORDS STANDARD.
22 000230 01 MASTER-RECORD.
23 000240     05 MST-EMPLOYEE-NUMBER      PICTURE 9(6).
24 000250     05 MST-EMPLOYEE-NAME      PICTURE X(28).
25 000260     05 MST-EMPLOYEE-CODE      PICTURE 9.
26 000270     05 MST-EMPLOYEE-SALARY    PICTURE 9(6)V99.
27 000280 WORKING-STORAGE SECTION.
28 000290 77 INPUT-FILE-STATUS          PICTURE XX.
29 000300 77 MASTER-FILE-STATUS        PICTURE XX.
30 000310 01 INPUTEND                  PICTURE X VALUE SPACE.
31 000320     88 THE-END-OF-INPUT        VALUE "E".
32 000330 01 MASTEREND                  PICTURE X VALUE SPACE.
33 000340     88 THE-END-OF-MASTER      VALUE "E".
34 000350 01 ERROR-INFO.
35 000360     05 OP-NAME                  PICTURE X(12).
36 000370     05 FILLER                  PICTURE XX VALUE SPACE.
37 000380     05 FILE-NAME              PICTURE X(11).
38 000390     05 FILLER                  PICTURE XX VALUE SPACE.
39 000400     05 FILLER                  PICTURE X(14)
40 000410         VALUE "FILE STATUS IS".
41 000420     05 FILLER                  PICTURE XX VALUE SPACE.
42 000430     05 SK                      PICTURE XX.
43 000440 PROCEDURE DIVISION.
   000450 DECLARATIVES.
   000460 INPUT-FILE-ERROR SECTION.
   000470     USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.  C
   000480 INPUT-FILE-ERROR-PARA.
44 000490     MOVE INPUT-FILE-STATUS TO SK.
45 000500     MOVE "INPUT-FILE" TO FILE-NAME.
46 000510     DISPLAY "FILE PROCESSING ERROR".
47 000520     DISPLAY ERROR-INFO.
48 000530     DISPLAY "PROCESSING TERMINATED DUE TO I-O ERROR".
49 000540     STOP RUN.
   000550 I-O-FILE-ERROR SECTION.
   000560     USE AFTER STANDARD ERROR PROCEDURE ON MASTER-FILE.  D
   000570 MASTER-FILE-ERROR-PARA.
50 000580     MOVE MASTER-FILE-STATUS TO SK.
51 000590     MOVE "MASTER-FILE" TO FILE-NAME.
52 000600     DISPLAY "FILE PROCESSING ERROR".
53 000610     DISPLAY ERROR-INFO.
54 000620     DISPLAY "PROCESSING TERMINATED DUE TO I-O ERROR".
55 000630     STOP RUN.
   000640 END DECLARATIVES.
   000650 MAIN-PROGRAM SECTION.
   000660 OPEN-FILES.
56 000670     MOVE "OPEN" TO OP-NAME.
57 000680     OPEN INPUT INPUT-FILE
   000690         I-O MASTER-FILE.
   000700 PROCESSING-LOGIC.
58 000710     PERFORM READ-INPUT-FILE.
59 000720     PERFORM READ-MASTER-FILE.
60 000730     PERFORM PROCESS-FILES UNTIL THE-END-OF-INPUT.

```

Figure 113 (Part 1 of 2). Example of a Sequential File Update Program

```

5738CB1 V2R2M0 AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7..IDENTFCN S COPYNAME CHG DATE
000740 CLOSE-FILES.
61 000750 MOVE "CLOSE" TO OP-NAME.
62 000760 CLOSE MASTER-FILE
000770 INPUT-FILE.
63 000780 STOP RUN.
000790 READ-INPUT-FILE.
64 000800 MOVE "READ" TO OP-NAME.
65 000810 READ INPUT-FILE
66 000820 AT END SET THE-END-OF-INPUT TO TRUE.
000830 READ-MASTER-FILE.
67 000840 MOVE "READ" TO OP-NAME.
68 000850 READ MASTER-FILE
000860 AT END
69 000870 SET THE-END-OF-MASTER TO TRUE
70 000880 MOVE "AT END CLOSE" TO OP-NAME
71 000890 CLOSE MASTER-FILE
72 000900 MOVE "OPEN EXTEND" TO OP-NAME
73 000910 OPEN EXTEND MASTER-FILE.
000920 PROCESS-FILES.
74 000930 IF THE-END-OF-MASTER
75 000940 WRITE MASTER-RECORD FROM INPUT-RECORD
76 000950 PERFORM READ-INPUT-FILE
000960 ELSE
77 000970 IF MST-EMPLOYEE-NUMBER LESS THAN INPUT-EMPLOYEE-NUMBER
78 000980 PERFORM READ-MASTER-FILE
000990 ELSE
79 001000 IF MST-EMPLOYEE-NUMBER = INPUT-EMPLOYEE-NUMBER
80 001010 MOVE "REWRITE" TO OP-NAME
81 001020 REWRITE MASTER-RECORD FROM INPUT-RECORD
82 001030 PERFORM READ-INPUT-FILE
83 001040 PERFORM READ-MASTER-FILE
001050 ELSE
84 001060 DISPLAY "ERROR RECORD -> ", INPUT-EMPLOYEE-NUMBER
85 001070 PERFORM READ-INPUT-FILE.
***** END OF SOURCE *****

5738CB1 V2R2M0 AS/400 COBOL Messages
STMT
* 15 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000160
Message . . . . : Blocking/Deblocking for file 'INPUT-FILE'
will be performed by compiler-generated code.
***** END OF MESSAGES *****

Message Summary
Total Info(0-4) Warning(5-19) Error(20-29) Severe(30-39) Terminal(40-99)
1 1 0 0 0 0
Source records read . . . . . : 107
Copy records read . . . . . : 0
Copy members processed . . . . . : 0
Sequence errors . . . . . : 0
Highest severity message issued . . : 0
LBL0901 00 Program UPDTSEQ created in library XMPLIB.
***** END OF COMPILATION *****

```

Figure 113 (Part 2 of 2). Example of a Sequential File Update Program

The example in Figure 113 on page 350 includes:

- A** A FILE STATUS clause so that the program records the status of the most recent I/O request involving INPUT-FILE.
- B** A FILE STATUS clause so that the program records the status of the most recent I/O request involving MASTER-FILE.
- C** A USE procedure that is run when an I/O error occurs during the processing of INPUT-FILE.
- D** A USE procedure that is run when an I/O error occurs during the processing of MASTER-FILE.

File status values and USE procedures play important roles in *error handling*. For more information, see Chapter 6, "COBOL/400 Exception and Error Handling."

Indexed File Creation

An **indexed file** is a file that records the key and the position of each record in a separate part of the file called an index.

This program creates an indexed file of summary records for bank depositors. The key within each indexed file record is INDEX-KEY (the depositor's account number); the input records are ordered in ascending sequence upon this key. Records are read from the input file and transferred to the indexed file record area. The indexed file record is then written.

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S COPYNAME  CHG DATE
 1 000010 IDENTIFICATION DIVISION.
 2 000020 PROGRAM-ID. CRTIND.
   000030
 3 000040 ENVIRONMENT DIVISION.
 4 000050 CONFIGURATION SECTION.
 5 000060 SOURCE-COMPUTER. IBM-AS400.                                05/24/91
 6 000070 OBJECT-COMPUTER. IBM-AS400.                                05/24/91
 7 000080 INPUT-OUTPUT SECTION.
 8 000090 FILE-CONTROL.
 9 000100     SELECT INDEXED-FILE ASSIGN TO DISK-INDEXFILE
10 000110     ORGANIZATION IS INDEXED
11 000120     ACCESS IS SEQUENTIAL
12 000130     RECORD KEY IS INDEX-KEY
13 000140     FILE STATUS IS INDEXED-FILE-STATUS.
14 000150     SELECT INPUT-FILE ASSIGN TO DISK-FILEG
15 000160     FILE STATUS IS INPUT-FILE-STATUS.
16 000170 DATA DIVISION.
17 000180 FILE SECTION.
18 000190 FD INDEXED-FILE LABEL RECORDS STANDARD.
19 000200 01 INDEX-RECORD.
20 000210     05 INDEX-KEY                PICTURE X(10).
21 000220     05 INDEX-FLD1                PICTURE X(10).
22 000230     05 INDEX-NAME                PICTURE X(20).
23 000240     05 INDEX-BAL                PICTURE S9(5)V99.
24 000250 FD INPUT-FILE LABEL RECORDS STANDARD.
25 000260 01 INPUT-RECORD.
26 000270     05 INPUT-KEY                PICTURE X(10).
27 000280     05 INPUT-NAME                PICTURE X(20).
28 000290     05 INPUT-BAL                PICTURE S9(5)V99.
29 000300 WORKING-STORAGE SECTION.
30 000310 77 INDEXED-FILE-STATUS          PICTURE XX.
31 000320 77 INPUT-FILE-STATUS            PICTURE XX.
32 000330 77 OP-NAME                      PICTURE X(7).
33 000340 01 INPUTEND                     PICTURE X VALUE SPACES.
34 000350     88 THE-END-OF-INPUT          VALUE "E".
35 000360 01 ERRORFLAG                    PICTURE X VALUE SPACES.
36 000370     88 ERROR-OCCURRED            VALUE "1".
37 000380 PROCEDURE DIVISION.
   000390 DECLARATIVES.
   000400 INPUT-ERROR SECTION.
   000410     USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
   000420 INPUT-ERROR-PARA.
38 000430     DISPLAY "UNEXPECTED ERROR ON ", OP-NAME, " FOR INPUT-FILE ".
39 000440     DISPLAY "FILE STATUS IS ", INPUT-FILE-STATUS.
40 000450     SET ERROR-OCCURRED TO TRUE.
   000460 OUTPUT-ERROR SECTION.
   000470     USE AFTER STANDARD ERROR PROCEDURE ON OUTPUT.
   000480 OUTPUT-ERROR-PARA.
41 000490     DISPLAY "UNEXPECTED ERROR ON ", OP-NAME, " FOR INDEXED-FILE ".
42 000500     DISPLAY "FILE STATUS IS ", INDEXED-FILE-STATUS.
43 000510     SET ERROR-OCCURRED TO TRUE.
   000520 END DECLARATIVES.
   000530 MAIN-PROCESSING SECTION.
   000540 MAIN-PROCEDURE.
44 000550     MOVE "OPEN" TO OP-NAME.
45 000560     OPEN INPUT INPUT-FILE
   000570         OUTPUT INDEXED-FILE.
46 000580     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
48 000590     PERFORM READ-INPUT-FILE.
49 000600     PERFORM LOAD-INDEXED-FILE THRU READ-INPUT-FILE
   000610         UNTIL THE-END-OF-INPUT.

```

Figure 114 (Part 1 of 2). Example of an Indexed File Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7..IDENTFCN S COPYNAME  CHG DATE
50 000620      MOVE "CLOSE" TO OP-NAME.
51 000630      CLOSE INPUT-FILE
000640          INDEXED-FILE.
52 000650      IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
54 000660      STOP RUN.
000670 LOAD-INDEXED-FILE.
55 000680      MOVE INPUT-KEY TO INDEX-KEY.
56 000690      MOVE INPUT-NAME TO INDEX-NAME.
57 000700      MOVE INPUT-BAL TO INDEX-BAL.
58 000710      MOVE SPACES TO INDEX-FLD1.
59 000720      MOVE "WRITE" TO OP-NAME.
60 000730      WRITE INDEX-RECORD
000740          INVALID KEY
61 000750          DISPLAY "WRITE FAILED FOR KEY ", INDEX-KEY.
62 000760      IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
000770 READ-INPUT-FILE.
64 000780      MOVE "READ" TO OP-NAME.
65 000790      READ INPUT-FILE
66 000800          AT END SET THE-END-OF-INPUT TO TRUE.
67 000810      IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
000820 ERROR-TERMINATION.
69 000830      DISPLAY "I-O ERROR OCCURRED - PROCESS TERMINATING".
70 000840      STOP RUN.
          ***** END OF SOURCE *****
5738CB1 V2R2M0          AS/400 COBOL Messages
STMT
* 18 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000190
Message . . . . : Blocking/Deblocking for file 'INDEXED-FILE'
will be performed by compiler-generated code.
* 24 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000250
Message . . . . : Blocking/Deblocking for file 'INPUT-FILE'
will be performed by compiler-generated code.
          ***** END OF MESSAGES *****
          Message Summary
Total  Info(0-4)  Warning(5-19)  Error(20-29)  Severe(30-39)  Terminal(40-99)
2      2          0              0              0              0
Source records read . . . . . : 84
Copy records read . . . . . : 0
Copy members processed . . . . . : 0
Sequence errors . . . . . : 0
Highest severity message issued . . : 0
LBL0901 00 Program CRTIND created in library XMPLIB.
          ***** END OF COMPILATION *****

```

Figure 114 (Part 2 of 2). Example of an Indexed File Program

Indexed File Updating

This program updates the indexed file created in the CRTIND program, using dynamic access.

The input records contain the key for the record, the depositor name, and the amount of the transaction.

When the input record is read, the program tests for:

- If this is a transaction record (in which case, all fields of the record are filled)
- If this is a record requesting sequential retrieval of a specific generic class (in which case, only the INPUT-GEN-FLD field of the input record contains data).

Random access is used for the updating and printing of the transaction records. Sequential access is used for the retrieval and printing of all records within one generic class.

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B.+...2....3.....4.....5.....6.....7..IDENTFCN S COPYNAME  CHG DATE
 1 000010 IDENTIFICATION DIVISION.
 2 000020 PROGRAM-ID.  UPDTIND.
   000030
 3 000040 ENVIRONMENT DIVISION.
 4 000050 CONFIGURATION SECTION.
 5 000060 SOURCE-COMPUTER.  IBM-AS400.
 6 000070 OBJECT-COMPUTER.  IBM-AS400.
 7 000080 INPUT-OUTPUT SECTION.
 8 000090 FILE-CONTROL.
 9 000100     SELECT MASTER-FILE ASSIGN TO DISK-INDEXFILE
10 000110     ORGANIZATION IS INDEXED
11 000120     ACCESS IS DYNAMIC
12 000130     RECORD KEY IS MASTER-KEY
13 000140     FILE STATUS IS MASTER-FILE-STATUS.
14 000150     SELECT INPUT-FILE ASSIGN TO DISK-FILEH
15 000160     FILE STATUS IS INPUT-FILE-STATUS.
16 000170     SELECT PRINT-FILE ASSIGN TO PRINTER-QSYSPT
17 000180     FILE STATUS IS PRINT-FILE-STATUS.
18 000190 DATA DIVISION.
19 000200 FILE SECTION.
20 000210 FD  MASTER-FILE LABEL RECORDS STANDARD.
21 000220 01  MASTER-RECORD.
22 000230 05  MASTER-KEY.
23 000240 10  MASTER-GEN-FLD  PICTURE X(5).
24 000250 10  MASTER-DET-FLD  PICTURE X(5).
25 000260 05  MASTER-FLD1    PICTURE X(10).
26 000270 05  MASTER-NAME     PICTURE X(20).
27 000280 05  MASTER-BAL      PICTURE S9(5)V99.
28 000290 FD  INPUT-FILE LABEL RECORDS STANDARD.
29 000300 01  INPUT-REC.
30 000310 05  INPUT-KEY.
31 000320 10  INPUT-GEN-FLD   PICTURE X(5).
32 000330 10  INPUT-DET-FLD   PICTURE X(5).
33 000340 05  INPUT-NAME      PICTURE X(20).
34 000350 05  INPUT-AMT      PICTURE S9(5)V99.
35 000360 FD  PRINT-FILE LABEL RECORDS OMITTED
36 000370 LINAGE 12 LINES FOOTING AT 9.
37 000380 01  PRINT-RECORD-1.
38 000390 05  PRINT-KEY         PICTURE X(10).
39 000400 05  FILLER           PICTURE X(5).
40 000410 05  PRINT-NAME     PICTURE X(20).
41 000420 05  FILLER           PICTURE X(5).
42 000430 05  PRINT-BAL      PICTURE $$$,$$9.99-.
43 000440 05  FILLER           PICTURE X(7).
44 000450 05  PRINT-AMT      PICTURE $$$,$$9.99-.
45 000460 05  FILLER           PICTURE X(5).
46 000470 05  PRINT-NEW-BAL  PICTURE $$$,$$9.99-.
47 000480 01  PRINT-RECORD-2 PICTURE X(89).
48 000490 WORKING-STORAGE SECTION.
49 000500 77  MASTER-FILE-STATUS PICTURE XX.
50 000510 77  INPUT-FILE-STATUS  PICTURE XX.
51 000520 77  PRINT-FILE-STATUS  PICTURE XX.
52 000530 77  LINES-TO-FOOT      PICTURE 99.
53 000540 01  PAGE-HEAD.
54 000550 05  FILLER             PICTURE X(38) VALUE SPACES.
55 000560 05  FILLER             PICTURE X(13) VALUE "UPDATE REPORT".
56 000570 05  FILLER             PICTURE X(38) VALUE SPACES.
57 000580 01  COLUMN-HEAD.
58 000590 05  FILLER             PICTURE X(6) VALUE "KEY ID".
59 000600 05  FILLER             PICTURE X(9) VALUE SPACES.
60 000610 05  FILLER             PICTURE X(4) VALUE "NAME".
61 000620 05  FILLER             PICTURE X(21) VALUE SPACES.
62 000630 05  FILLER             PICTURE X(11) VALUE "CUR BALANCE".
63 000640 05  FILLER             PICTURE X(6) VALUE SPACES.
64 000650 05  FILLER             PICTURE X(13) VALUE "UPDATE AMOUNT".
65 000660 05  FILLER             PICTURE X(4) VALUE SPACES.
66 000670 05  FILLER             PICTURE X(11) VALUE "NEW BALANCE".
67 000680 05  FILLER             PICTURE X(4) VALUE SPACES.
68 000690 01  PAGE-FOOT.
69 000700 05  FILLER             PICTURE X(81) VALUE SPACES.
70 000710 05  FILLER             PICTURE A(6) VALUE "PAGE ".
71 000720 05  PG-NUMBER          PICTURE 99 VALUE 00.
   000730
72 000740 01  INPUTEND           PICTURE X VALUE SPACE.
73 000750 88  THE-END-OF-INPUT   VALUE "E".

```

Figure 115 (Part 1 of 4). Example of an Indexed File Update Program

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B..+...2....+...3....+...4....+...5....+...6....+...7..IDENTFCN S COPYNAME  CHG DATE
74 000760 01 ERRORFLAG          PICTURE X VALUE SPACE.
75 000770 88 ERROR-OCCURRED      VALUE "1".
76 000780 01 ERROR-DATA.
77 000790 05 FILLER              PICTURE X(21)
                                VALUE "STATEMENT FAILING IS ".
78 000800
79 000810 05 OP-NAME            PICTURE X(9).
80 000820 05 FILLER              PICTURE X(16)
81 000830                                VALUE "FILE STATUS IS".
82 000840 05 STATUS-VALUE        PICTURE XX.
83 000850 01 INPUT-MESSAGE.
84 000860 05 FILLER              PICTURE X(30)
85 000870                                VALUE "UNEXPECTED ERROR ON INPUT-FILE" .
86 000880 01 I-O-MESSAGE.
87 000890 05 FILLER              PICTURE X(31)
88 000900                                VALUE "UNEXPECTED ERROR ON MASTER-FILE" .
89 000910 01 OUTPUT-MESSAGE.
90 000920 05 FILLER              PICTURE X(30)
91 000930                                VALUE "UNEXPECTED ERROR ON PRINT-FILE" .
92 000940 PROCEDURE DIVISION.
    000950 DECLARATIVES.
    000960 INPUT-ERROR SECTION.
    000970     USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
    000980 INPUT-ERROR-PARA.
93 000990     DISPLAY INPUT-MESSAGE.
94 001000     MOVE INPUT-FILE-STATUS TO STATUS-VALUE.
95 001010     DISPLAY ERROR-DATA.
96 001020     SET ERROR-OCCURRED TO TRUE.
    001030 I-O-ERROR SECTION.
    001040     USE AFTER STANDARD ERROR PROCEDURE ON I-O.
    001050 I-O-ERROR-PARA.
97 001060     DISPLAY I-O-MESSAGE.
98 001070     MOVE MASTER-FILE-STATUS TO STATUS-VALUE.
99 001080     DISPLAY ERROR-DATA.
100 001090     SET ERROR-OCCURRED TO TRUE.
    001100 OUTPUT-ERROR SECTION.
    001110     USE AFTER STANDARD ERROR PROCEDURE ON OUTPUT.
    001120 OUTPUT-ERROR-PARA.
101 001130     DISPLAY OUTPUT-MESSAGE.
102 001140     MOVE PRINT-FILE-STATUS TO STATUS-VALUE.
103 001150     DISPLAY ERROR-DATA.
104 001160     SET ERROR-OCCURRED TO TRUE.
    001170 END DECLARATIVES.
    001180 MAIN-PROCESSING SECTION.
    001190 MAIN-PROCEDURE.
105 001200     MOVE "OPEN" TO OP-NAME.
106 001210     OPEN INPUT INPUT-FILE
    001220         I-O MASTER-FILE
    001230         OUTPUT PRINT-FILE.
107 001240     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
109 001250     PERFORM PAGE-START.
110 001260     PERFORM READ-INPUT-FILE.
111 001270     PERFORM PROCESS-DATA THRU READ-INPUT-FILE
    001280         UNTIL THE-END-OF-INPUT.
112 001290     PERFORM PAGE-END.
113 001300     MOVE "CLOSE" TO OP-NAME.
114 001310     CLOSE INPUT-FILE
    001320         MASTER-FILE
    001330         PRINT-FILE.
115 001340     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
117 001350     STOP RUN.
    001360
    001370 PROCESS-DATA.
118 001380     IF INPUT-DET-FLD EQUAL SPACES
119 001390         PERFORM INIT-SEQUENTIAL-PROCESS
    001400     ELSE
120 001410         PERFORM DYNAMIC-PROCESS.
    001420 READ-INPUT-FILE.
121 001430     MOVE "READ" TO OP-NAME.
122 001440     READ INPUT-FILE
123 001450         AT END SET THE-END-OF-INPUT TO TRUE.
124 001460     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
    001470
    001480 INIT-SEQUENTIAL-PROCESS.

```

Figure 115 (Part 2 of 4). Example of an Indexed File Update Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
126 001490  MOVE INPUT-GEN-FLD TO MASTER-GEN-FLD.
127 001500  MOVE "START" TO OP-NAME.
128 001510  START MASTER-FILE
      001520  KEY IS NOT LESS THAN MASTER-GEN-FLD
      001530  INVALID KEY
129 001540  DISPLAY "MASTER-FILE START FAILED: INVALID KEY ",
      001550  MASTER-GEN-FLD
130 001560  MOVE HIGH-VALUE TO MASTER-GEN-FLD.
131 001570  IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
133 001580  PERFORM SEQUENTIAL-PROCESS
      001590  UNTIL INPUT-GEN-FLD NOT EQUAL MASTER-GEN-FLD.
      001600
      001610 SEQUENTIAL-PROCESS.
134 001620  MOVE "READ NEXT" TO OP-NAME.
135 001630  READ MASTER-FILE NEXT RECORD
136 001640  AT END MOVE HIGH-VALUE TO MASTER-GEN-FLD.
137 001650  IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
139 001660  IF INPUT-GEN-FLD EQUAL MASTER-GEN-FLD
140 001670  MOVE MASTER-KEY TO PRINT-KEY
141 001680  MOVE MASTER-NAME TO PRINT-NAME
142 001690  MOVE MASTER-BAL TO PRINT-NEW-BAL
143 001700  PERFORM PRINT-DETAIL.
      001710
      001720 DYNAMIC-PROCESS.
144 001730  MOVE INPUT-KEY TO MASTER-KEY.
145 001740  MOVE "READ" TO OP-NAME.
146 001750  READ MASTER-FILE
      001760  INVALID KEY
147 001770  DISPLAY "MASTER-FILE READ FAILED: INVALID KEY ",
      001780  MASTER-KEY
148 001790  MOVE HIGH-VALUE TO MASTER-GEN-FLD.
149 001800  IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
151 001810  IF INPUT-GEN-FLD EQUAL MASTER-GEN-FLD
152 001820  MOVE MASTER-KEY TO PRINT-KEY
153 001830  MOVE MASTER-NAME TO PRINT-NAME
154 001840  MOVE MASTER-BAL TO PRINT-BAL
155 001850  MOVE INPUT-AMT TO PRINT-AMT
156 001860  ADD INPUT-AMT TO MASTER-BAL
157 001870  MOVE MASTER-BAL TO PRINT-NEW-BAL
158 001880  PERFORM PRINT-DETAIL
159 001890  MOVE "REWRITE" TO OP-NAME
160 001900  REWRITE MASTER-RECORD
      001910  INVALID KEY
161 001920  DISPLAY "MASTER-FILE REWRITE FAILED: INVALID KEY ",
      001930  MASTER-KEY
162 001940  MOVE HIGH-VALUE TO MASTER-GEN-FLD.
163 001950  IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
      001960 PRINT-DETAIL.
165 001970  MOVE "WRITE" TO OP-NAME.
166 001980  WRITE PRINT-RECORD-1
      001990  AT END-OF-PAGE
167 002000  PERFORM PAGE-END THROUGH PAGE-START.
168 002010  IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
170 002020  MOVE SPACES TO PRINT-RECORD-1.
      002030
      002040 PAGE-END.
171 002050  MOVE "WRITE" TO OP-NAME.
172 002060  ADD 1 TO PG-NUMBER.
173 002070  SUBTRACT LINAGE-COUNTER OF PRINT-FILE FROM 12
      002080  GIVING LINES-TO-FOOT.
174 002090  MOVE SPACES TO PRINT-RECORD-1.
175 002100  WRITE PRINT-RECORD-1
      002110  AFTER ADVANCING LINES-TO-FOOT.
176 002120  WRITE PRINT-RECORD-2 FROM PAGE-FOOT
      002130  BEFORE ADVANCING PAGE.
177 002140  IF ERROR-OCCURRED GO TO ERROR-TERMINATION.

```

Figure 115 (Part 3 of 4). Example of an Indexed File Update Program


```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B.....2.....3.....4.....5.....6.....7..IDENTFCN S COPYNAME  CHG DATE
002150 PAGE-START.
179 002160 WRITE PRINT-RECORD-2 FROM PAGE-HEAD
002170 AFTER ADVANCING 0 LINES.
180 002180 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
182 002190 MOVE SPACES TO PRINT-RECORD-2.
183 002200 WRITE PRINT-RECORD-2 FROM COLUMN-HEAD
002210 AFTER ADVANCING 1 LINE.
184 002220 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
186 002230 MOVE SPACES TO PRINT-RECORD-2.
002240 ERROR-TERMINATION.
187 002250 DISPLAY "PROCESS TERMINATING ABNORMALLY".
188 002260 STOP RUN.
          ***** END OF SOURCE *****
5738CB1 V2R2M0          AS/400 COBOL Messages
STMT
* 28 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000290
Message . . . . : Blocking/Deblocking for file 'INPUT-FILE'
will be performed by compiler-generated code.
          ***** END OF MESSAGES *****
          Message Summary
Total   Info(0-4)   Warning(5-19)   Error(20-29)   Severe(30-39)   Terminal(40-99)
1       1           0                0                0                0
Source records read . . . . . : 226
Copy records read . . . . . : 0
Copy members processed . . . . . : 0
Sequence errors . . . . . : 0
Highest severity message issued . . : 0
LBL0901 00 Program UPDTIND created in library XMPLIB.
          ***** END OF COMPILATION *****

```

Figure 115 (Part 4 of 4). Example of an Indexed File Update Program

Relative File Creation

This program creates a relative file of summary sales records using sequential access. Each record contains a five-year summary of unit and dollar sales for one week of the year; there are 52 records within the file, each representing one week.

Each input record represents the summary sales for one week of one year. The records for the first week of the last five years (in ascending order) are the first five input records. The records for the second week of the last five years are the next five input records, and so on. Thus, five input records fill one output record.

The RELATIVE KEY for the RELATIVE-FILE is not specified because it is not required for sequential access unless the START statement is used. (For updating, however, the key is INPUT-WEEK.)

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
 1 000010 IDENTIFICATION DIVISION.
 2 000020 PROGRAM-ID. CRTREL.
   000030
 3 000040 ENVIRONMENT DIVISION.
 4 000050 CONFIGURATION SECTION.
 5 000060 SOURCE-COMPUTER. IBM-AS400.
 6 000070 OBJECT-COMPUTER. IBM-AS400.
 7 000080 SPECIAL-NAMES. REQUESTOR IS REQUESTOR.
 8 000090 FILE-CONTROL.
 9 000100 SELECT RELATIVE-FILE ASSIGN TO DISK-FILED
10 000110 ORGANIZATION IS RELATIVE
11 000120 ACCESS IS SEQUENTIAL
12 000130 FILE STATUS RELATIVE-FILE-STATUS.
13 000140 SELECT INPUT-FILE ASSIGN TO DISK-FILEC
14 000150 FILE STATUS INPUT-FILE-STATUS.
   000160
15 000170 DATA DIVISION.
16 000180 FILE SECTION.
17 000190 FD RELATIVE-FILE LABEL RECORDS ARE STANDARD.
18 000200 01 RELATIVE-RECORD-01.
19 000210 05 RELATIVE-RECORD OCCURS 5 TIMES INDEXED BY REL-INDEX.
20 000220 10 RELATIVE-YEAR PICTURE 99.
21 000230 10 RELATIVE-WEEK PICTURE 99.
22 000240 10 RELATIVE-UNIT-SALES PICTURE S9(6).
23 000250 10 RELATIVE-DOLLAR-SALES PICTURE S9(9)V99.
24 000260 FD INPUT-FILE LABEL RECORDS STANDARD.
25 000270 01 INPUT-RECORD.
26 000280 05 INPUT-YEAR PICTURE 99.
27 000290 05 INPUT-WEEK PICTURE 99.
28 000300 05 INPUT-UNIT-SALES PICTURE S9(6).
29 000310 05 INPUT-DOLLAR-SALES PICTURE S9(9)V99.
30 000320 WORKING-STORAGE SECTION.
31 000330 77 INPUT-FILE-STATUS PICTURE XX.
32 000340 77 RELATIVE-FILE-STATUS PICTURE XX.
33 000350 01 WORK-RECORD.
34 000360 05 WORK-YEAR PICTURE 99 VALUE 00.
35 000370 05 WORK-WEEK PICTURE 99.
36 000380 05 WORK-UNIT-SALES PICTURE S9(6).
37 000390 05 WORK-DOLLAR-SALES PICTURE S9(9)V99.
38 000400 01 ERROR-INFO.
39 000410 05 OP-NAME PICTURE X(5).
40 000420 05 FILLER PICTURE X(10)
41 000430 VALUE " ERROR ON ".
42 000440 05 FILE-NAME PICTURE X(13).
43 000450 05 FILLER PICTURE X(16)
44 000460 VALUE " FILE STATUS IS ".
45 000470 05 STATUS-VALUE PICTURE XX.
46 000480 01 ERROR-FLAG PICTURE X VALUE SPACE.
47 000490 88 ERROR-OCCURRED VALUE "1".
48 000500 01 INPUTEND PICTURE X VALUE SPACE.
49 000510 88 THE-END-OF-INPUT VALUE "E".
   000520
50 000530 PROCEDURE DIVISION.
   000540 DECLARATIVES.
   000550
   000560 INP-FILE-ERROR SECTION.
   000570 USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.
   000580 INPUT-FILE-ERROR.
51 000590 MOVE "INPUT-FILE" TO FILE-NAME.
52 000600 MOVE INPUT-FILE-STATUS TO STATUS-VALUE.
53 000610 SET ERROR-OCCURRED TO TRUE.
   000620 REL-FILE-ERROR SECTION.
   000630 USE AFTER STANDARD ERROR PROCEDURE ON RELATIVE-FILE.
   000640 RELATIVE-FILE-ERROR.
54 000650 MOVE "RELATIVE-FILE" TO FILE-NAME.
55 000660 MOVE RELATIVE-FILE-STATUS TO STATUS-VALUE.
56 000670 SET ERROR-OCCURRED TO TRUE.
   000680 END DECLARATIVES.
   000690 BEGIN-PROCESSING SECTION.
   000700 PROCESSING-CONTROL.
57 000710 MOVE "OPEN" TO OP-NAME.
58 000720 OPEN INPUT INPUT-FILE
   000730 OUTPUT RELATIVE-FILE.
59 000740 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
61 000750 SET REL-INDEX TO 1.
62 000760 PERFORM READ-INPUT-FILE.

```

Figure 116 (Part 1 of 2). Example of a Relative File Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...+...2...+...3...+...4...+...5...+...6...+...7..IDENTFCN S COPYNAME  CHG DATE
63 000770 PERFORM PROCESS-DATA THRU READ-INPUT-FILE
    000780 UNTIL THE-END-OF-INPUT.
64 000790 CLOSE RELATIVE-FILE INPUT-FILE.
65 000800 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
67 000810 STOP RUN.
    000820 ERROR-TERMINATION.
68 000830 DISPLAY ERROR-INFO UPON REQUESTOR.
69 000840 DISPLAY "PROCESSING TERMINATED DUE TO I-O ERROR"
    000850 UPON REQUESTOR.
70 000860 STOP RUN.
    000870 PROCESS-DATA.
71 000880 MOVE INPUT-RECORD TO RELATIVE-RECORD (REL-INDEX).
72 000890 IF REL-INDEX NOT = 5
73 000900 SET REL-INDEX UP BY 1
    000910 ELSE
74 000920 SET REL-INDEX TO 1
75 000930 PERFORM RELATIVE-FILE-WRITE.
    000940 READ-INPUT-FILE.
76 000950 MOVE "READ" TO OP-NAME.
77 000960 READ INPUT-FILE
78 000970 AT END SET THE-END-OF-INPUT TO TRUE.
79 000980 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
    000990 RELATIVE-FILE-WRITE.
81 001000 MOVE "WRITE" TO OP-NAME.
82 001010 WRITE RELATIVE-RECORD-01.
83 001020 IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
          ***** END OF SOURCE *****
5738CB1 V2R2M0          AS/400 COBOL Messages
STMT
* MSGID: LBL0027 SEVERITY: 10 SEQNBR:
  Message . . . . : I-O SECTION not found. Assumed present
* 17 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000190
  Message . . . . : Blocking/Deblocking for file 'RELATIVE-FILE'
    will be performed by compiler-generated code.
* 24 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000260
  Message . . . . : Blocking/Deblocking for file 'INPUT-FILE'
    will be performed by compiler-generated code.
          ***** END OF MESSAGES *****
                          Message Summary
Total   Info(0-4)   Warning(5-19)   Error(20-29)   Severe(30-39)   Terminal(40-99)
   3         2           1             0               0                 0
Source records read . . . . . : 102
Copy records read . . . . . : 0
Copy members processed . . . . . : 0
Sequence errors . . . . . : 0
Highest severity message issued . . : 10
  LBL0901 00 Program CRTREL created in library XMPLIB.
          ***** END OF COMPILATION *****

```

Figure 116 (Part 2 of 2). Example of a Relative File Program

Relative File Updating

This program uses sequential access to update the file of summary sales records created in the CRTREL program. The updating program adds a record for the new year and deletes the oldest year's records from RELATIVE-FILE.

The input record represents the summary sales record for one week of the preceding year. The RELATIVE KEY for the RELATIVE-FILE is in the input record as INPUT-WEEK. The RELATIVE KEY is used to check that the record was correctly written.

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
 1 000010 IDENTIFICATION DIVISION.
 2 000020 PROGRAM-ID.  UPDTREL.
   000030
 3 000040 ENVIRONMENT DIVISION.
 4 000050 CONFIGURATION SECTION.
 5 000060 SOURCE-COMPUTER.  IBM-AS400.          05/24/91
 6 000070 OBJECT-COMPUTER.  IBM-AS400.          05/24/91
 7 000080 INPUT-OUTPUT SECTION.
 8 000090 FILE-CONTROL.
 9 000100     SELECT RELATIVE-FILE ASSIGN TO DISK-FILED
10 000110     ORGANIZATION IS RELATIVE
11 000120     ACCESS IS SEQUENTIAL
12 000130     RELATIVE KEY INPUT-WEEK
13 000140     FILE STATUS STATUS-VALUE.
14 000150     SELECT INPUT-FILE ASSIGN TO DISK-FILES2
15 000160     FILE STATUS STATUS-VALUE.
   000170
16 000180 DATA DIVISION.
17 000190 FILE SECTION.
18 000200 FD RELATIVE-FILE LABEL RECORDS STANDARD.
19 000210 01 RELATIVE-RECORD          PICTURE X(105).
20 000220 FD INPUT-FILE LABEL RECORDS STANDARD.
21 000230 01 INPUT-RECORD.
22 000240 05 INPUT-YEAR          PICTURE 99.
23 000250 05 INPUT-WEEK          PICTURE 99.
24 000260 05 INPUT-UNIT-SALES    PICTURE S9(6).
25 000270 05 INPUT-DOLLAR-SALES  PICTURE S9(9)V99.
26 000280 WORKING-STORAGE SECTION.
   000290
27 000300 01 INPUTEND          PICTURE X VALUE SPACE.
28 000310 88 THE-END-OF-INPUT    VALUE "E".
29 000320 01 WORK-RECORD.
30 000330 05 FILLER          PICTURE X(21).
31 000340 05 CURRENT-WORK-YEARS  PICTURE X(84).
32 000350 05 NEW-WORK-YEAR.
33 000360 10 WORK-YEAR          PICTURE 99.
34 000370 10 WORK-WEEK          PICTURE 99.
35 000380 10 WORK-UNIT-SALES    PICTURE S9(6).
36 000390 10 WORK-DOLLAR-SALES  PICTURE S9(9)V99.
37 000400 66 WORK-OUT-RECORD RENAMES
38 000410 CURRENT-WORK-YEARS THROUGH NEW-WORK-YEAR.
39 000420 01 ERROR-MESSAGE.
40 000430 05 OP-NAME          PICTURE X(7).
41 000440 05 FILLER          PICTURE X(10)
42 000450          VALUE " ERROR ON ".
43 000460 05 FILE-NAME        PICTURE X(13).
44 000470 05 FILLER          PICTURE X(16)
45 000480          VALUE " FILE STATUS IS ".
46 000490 05 STATUS-VALUE    PICTURE X(2).
   000500
47 000510 PROCEDURE DIVISION.
   000520 DECLARATIVES.
   000530 I-O-ERROR SECTION.
   000540     USE AFTER STANDARD ERROR PROCEDURE ON RELATIVE-FILE,
   000550     INPUT-FILE.
   000560 ERROR-PROCEDURE.
48 000570     DISPLAY ERROR-MESSAGE.
49 000580     DISPLAY "PROCESSING TERMINATING".
50 000590     STOP RUN.
   000600 END DECLARATIVES.
   000610 MAIN-PROCEDURE SECTION.
   000620 BEGIN-PROCESSING.
51 000630     MOVE "OPEN" TO OP-NAME.
52 000640     MOVE "INPUT-FILE" TO FILE-NAME.
53 000650     OPEN INPUT INPUT-FILE.
54 000660     MOVE "RELATIVE-FILE" TO FILE-NAME.
55 000670     OPEN I-O RELATIVE-FILE.
56 000680     PERFORM READ-FILES.
57 000690     PERFORM UPDATE-RELATIVE-FILE THRU READ-FILES
   000700     UNTIL THE-END-OF-INPUT.
58 000710     MOVE "CLOSE" TO OP-NAME.
59 000720     MOVE "INPUT-FILE" TO FILE-NAME.
60 000730     CLOSE INPUT-FILE.
61 000740     MOVE "RELATIVE-FILE" TO FILE-NAME.
62 000750     CLOSE RELATIVE-FILE.
63 000760     STOP RUN.

```

Figure 117 (Part 1 of 2). Example of a Relative File Update Program

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B.+...2....+...3....+...4....+...5....+...6....+...7..IDENTFCN S COPYNAME  CHG DATE
000770 UPDATE-RELATIVE-FILE.
64 000780 MOVE "REWRITE" TO OP-NAME.
65 000790 MOVE "RELATIVE-FILE" TO FILE-NAME.
66 000800 REWRITE RELATIVE-RECORD FROM WORK-OUT-RECORD.
000810 READ-FILES.
67 000820 MOVE "READ" TO OP-NAME.
68 000830 MOVE "RELATIVE-FILE" TO FILE-NAME.
69 000840 READ RELATIVE-FILE INTO WORK-RECORD
70 000850 AT END SET THE-END-OF-INPUT TO TRUE.
71 000860 MOVE "INPUT-FILE" TO FILE-NAME.
72 000870 READ INPUT-FILE INTO NEW-WORK-YEAR
73 000880 AT END SET THE-END-OF-INPUT TO TRUE.
          * * * * * E N D   O F   S O U R C E   * * * * *

5738CB1 V2R2M0          AS/400 COBOL Messages
STMT
* 20 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000220
    Message . . . . : Blocking/Deblocking for file 'INPUT-FILE'
    will be performed by compiler-generated code.
          * * * * * E N D   O F   M E S S A G E S   * * * * *

          Message Summary
Total   Info(0-4)   Warning(5-19)   Error(20-29)   Severe(30-39)   Terminal(40-99)
1       1           0                 0                 0                 0
Source records read . . . . . : 88
Copy records read . . . . . : 0
Copy members processed . . . . . : 0
Sequence errors . . . . . : 0
Highest severity message issued . . : 0
LBL0901 00 Program UPDTREL created in library XMPLIB.
          * * * * * E N D   O F   C O M P I L A T I O N   * * * * *

```

Figure 117 (Part 2 of 2). Example of a Relative File Update Program

Relative File Retrieval

This program retrieves the summary file created by the CRTREL program, using dynamic access.

The records of the INPUT-FILE contain one required field (INPUT-WEEK), which is the RELATIVE KEY for RELATIVE-FILE, and one optional field (END-WEEK). An input record containing data in INPUT-WEEK and spaces in END-WEEK requests a printout for that one specific RELATIVE-RECORD; the record is retrieved through random access. (**Random processing** is a method of processing in which records can be read from, written to, or removed from a file in an order requested by the program that is using them.) An input record containing data in both INPUT-WEEK and END-WEEK requests a printout of all the RELATIVE-FILE records within the RELATIVE KEY range of INPUT-WEEK through END-WEEK inclusive. These records are retrieved through sequential access.

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B. ....2....3....4....5....6....7..IDENTFCN S COPYNAME  CHG DATE
 1 000010 IDENTIFICATION DIVISION.
 2 000020 PROGRAM-ID. RTRVREL.
   000030
 3 000040 ENVIRONMENT DIVISION.
 4 000050 CONFIGURATION SECTION.
 5 000060 SOURCE-COMPUTER. IBM-AS400.                05/24/91
 6 000070 OBJECT-COMPUTER. IBM-AS400.                05/24/91
 7 000080 SPECIAL-NAMES. REQUESTOR IS REQUESTOR.
 8 000090 INPUT-OUTPUT SECTION.
 9 000100 FILE-CONTROL.
10 000110     SELECT RELATIVE-FILE ASSIGN TO DISK-FILED
11 000120         ORGANIZATION IS RELATIVE
12 000130         ACCESS IS DYNAMIC
13 000140         RELATIVE KEY INPUT-WEEK
14 000150         FILE STATUS IS RELATIVE-FILE-STATUS.
15 000160     SELECT INPUT-FILE ASSIGN TO DISK-FILEF
16 000170         FILE STATUS IS INPUT-FILE-STATUS.
17 000180     SELECT PRINT-FILE ASSIGN TO PRINTER-QSYSRPT
18 000190         FILE STATUS IS PRINT-FILE-STATUS.
   000200
19 000210 DATA DIVISION.
20 000220 FILE SECTION.
21 000230 FD RELATIVE-FILE LABEL RECORDS STANDARD.
22 000240 01 RELATIVE-RECORD-01.
23 000250     05 RELATIVE-RECORD OCCURS 5 TIMES INDEXED BY REL-INDEX.
24 000260         10 RELATIVE-YEAR          PICTURE 99.
25 000270         10 RELATIVE-WEEK          PICTURE 99.
26 000280         10 RELATIVE-UNIT-SALES     PICTURE S9(6).
27 000290         10 RELATIVE-DOLLAR-SALES PICTURE S9(9)V99.
28 000300 FD INPUT-FILE LABEL RECORDS STANDARD.
29 000310 01 INPUT-RECORD.
30 000320     05 INPUT-WEEK                PICTURE 99.
31 000330     05 END-WEEK                  PICTURE 99.
32 000340 FD PRINT-FILE LABEL RECORDS OMITTED.
33 000350 01 PRINT-RECORD.
34 000360     05 PRINT-WEEK                  PICTURE 99.
35 000370     05 FILLER                    PICTURE X(5).
36 000380     05 PRINT-YEAR                 PICTURE 99.
37 000390     05 FILLER                    PICTURE X(5).
38 000400     05 PRINT-UNIT-SALES          PICTURE ZZZ,ZZ9.
39 000410     05 FILLER                    PICTURE X(5).
40 000420     05 PRINT-DOLLAR-SALES        PICTURE $$$,$$$,$$$.$99.
41 000430 WORKING-STORAGE SECTION.
42 000440 77 RELATIVE-FILE-STATUS         PICTURE XX.
43 000450 77 INPUT-FILE-STATUS           PICTURE XX.
44 000460 77 PRINT-FILE-STATUS           PICTURE XX.
45 000470 77 HIGH-WEEK                    PICTURE 99 VALUE 53.
46 000480 77 OP-NAME                      PICTURE X(9).
47 000490 01 INPUTEND                     PICTURE X(9).
48 000500     88 THE-END-OF-INPUT         VALUE "E".
49 000510 PROCEDURE DIVISION.
   000520 DECLARATIVES.
   000530 RELATIVE-FILE-ERROR SECTION.
   000540     USE AFTER STANDARD ERROR PROCEDURE ON RELATIVE-FILE.
   000550 RELATIVE-ERROR-MSG.
50 000560     DISPLAY OP-NAME, " ERROR ON RELATIVE-FILE ".
51 000570     DISPLAY "FILE STATUS VALUE IS ", RELATIVE-FILE-STATUS.
52 000580     DISPLAY "PROCESSING TERMINATED ".
53 000590     STOP RUN.
   000600 INPUT-FILE-ERROR SECTION.
   000610     USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.
   000620 INPUT-ERROR-MSG.
54 000630     DISPLAY OP-NAME, " ERROR ON INPUT-FILE ".
55 000640     DISPLAY "FILE STATUS VALUE IS ", INPUT-FILE-STATUS.
56 000650     DISPLAY "PROCESSING TERMINATED ".
57 000660     STOP RUN.
   000670 PRINT-FILE-ERROR SECTION.
   000680     USE AFTER STANDARD ERROR PROCEDURE ON PRINT-FILE.
   000690 PRINT-ERROR-MSG.
58 000700     DISPLAY OP-NAME, " ERROR ON PRINT-FILE ".
59 000710     DISPLAY "FILE STATUS VALUE IS ", PRINT-FILE-STATUS.
60 000720     DISPLAY "PROCESSING TERMINATED ".
61 000730     STOP RUN.
   000740 END DECLARATIVES.

```

Figure 118 (Part 1 of 2). Example of a Relative File Retrieval Program

```

5738CB1 V2R2M0                AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
000750 MAIN-PROCEDURE SECTION.
000760 MAIN-PROCESSING.
62 000770 MOVE "OPEN" TO OP-NAME.
63 000780 OPEN INPUT INPUT-FILE RELATIVE-FILE
000790 OUTPUT PRINT-FILE.
64 000800 MOVE SPACES TO PRINT-RECORD.
65 000810 PERFORM READ-INPUT-FILE.
66 000820 PERFORM CONTROL-PROCESS THRU READ-INPUT-FILE
000830 UNTIL THE-END-OF-INPUT.
67 000840 MOVE "CLOSE" TO OP-NAME.
68 000850 CLOSE RELATIVE-FILE
000860 INPUT-FILE
000870 PRINT-FILE.
69 000880 STOP RUN.
000890 CONTROL-PROCESS.
70 000900 IF (END-WEEK = SPACES OR END-WEEK = 00)
71 000910 PERFORM RANDOM-PROCESS
000920 ELSE
72 000930 PERFORM SEQUENTIAL-PROCESS.
000940 READ-INPUT-FILE.
73 000950 MOVE "READ" TO OP-NAME.
74 000960 READ INPUT-FILE
75 000970 AT END SET THE-END-OF-INPUT TO TRUE.
000980 RANDOM-PROCESS.
76 000990 MOVE "READ" TO OP-NAME.
77 001000 READ RELATIVE-FILE
78 001010 INVALID KEY MOVE HIGH-WEEK TO RELATIVE-WEEK(1).
79 001020 IF RELATIVE-WEEK(1) NOT EQUAL HIGH-WEEK
80 001030 PERFORM PRINT-SUMMARY VARYING REL-INDEX FROM 1 BY 1
001040 UNTIL REL-INDEX > 5.
001050 SEQUENTIAL-PROCESS.
81 001060 MOVE "READ" TO OP-NAME.
82 001070 READ RELATIVE-FILE
83 001080 INVALID KEY MOVE HIGH-WEEK TO RELATIVE-WEEK(1).
84 001090 PERFORM READ-REL-SEQ
001100 UNTIL RELATIVE-WEEK(1) GREATER THAN END-WEEK.
001110
001120 READ-REL-SEQ.
85 001130 PERFORM PRINT-SUMMARY VARYING REL-INDEX FROM 1 BY 1
001140 UNTIL REL-INDEX > 5.
86 001150 MOVE "READ NEXT" TO OP-NAME.
87 001160 READ RELATIVE-FILE NEXT RECORD
88 001170 AT END MOVE HIGH-WEEK TO RELATIVE-WEEK(1).
001180 PRINT-SUMMARY.
89 001190 MOVE RELATIVE-YEAR (REL-INDEX) TO PRINT-YEAR.
90 001200 MOVE RELATIVE-WEEK (REL-INDEX) TO PRINT-WEEK.
91 001210 MOVE RELATIVE-UNIT-SALES (REL-INDEX) TO PRINT-UNIT-SALES.
92 001220 MOVE RELATIVE-DOLLAR-SALES(REL-INDEX) TO PRINT-DOLLAR-SALES.
93 001230 MOVE "WRITE" TO OP-NAME.
94 001240 WRITE PRINT-RECORD AFTER ADVANCING 2 LINES.
          ***** END OF SOURCE *****
5738CB1 V2R2M0                AS/400 COBOL Messages
STMT
* 28 MSGID: LBL0650 SEVERITY: 00 SEQNBR: 000300
Message . . . . : Blocking/Deblocking for file 'INPUT-FILE'
will be performed by compiler-generated code.
          ***** END OF MESSAGES *****
          Message Summary
Total   Info(0-4)   Warning(5-19)   Error(20-29)   Severe(30-39)   Terminal(40-99)
1       1           0                 0                 0                 0
Source records read . . . . . : 124
Copy records read . . . . . : 0
Copy members processed . . . . . : 0
Sequence errors . . . . . : 0
Highest severity message issued . . : 0
LBL0901 00 Program RTRVREL created in library XMPLIB.
          ***** END OF COMPILATION *****

```

Figure 118 (Part 2 of 2). Example of a Relative File Retrieval Program

Sorting and Merging Files

Figure 119 illustrates the creation of sorted files of current sales and year-to-date sales.

First, the SORT statement for current sales is executed. The input procedure for this sorting operation is SCREEN-DEPT. The records are sorted in ascending order of department, and within each department, in descending order of net sales. The output for this sort is then printed.

After the sorting operation is completed, the current sales records are merged with the year-to-date sales records. The records in this file are merged in ascending order of department number and, within each department, in ascending order of employee numbers, and, for each employee, in ascending order of months to create an updated year-to-date master file.

When the merging process finishes, the updated year-to-date master file is printed.

```
5738CB1 V2R2M0 910524          AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7..IDENTFCN S COPYNAME  CHG DATE
 1 000010 IDENTIFICATION DIVISION.
 2 000020 PROGRAM-ID. SORTMERGE.
 3 000030*****
 4 000040* THIS IS A SORT/MERGE EXAMPLE USING AN INPUT PROCEDURE *
 5 000050*****
 6 000060 ENVIRONMENT DIVISION.
 7 000070 CONFIGURATION SECTION.
 8 000080 SOURCE-COMPUTER. IBM-AS400.
 9 000090 OBJECT-COMPUTER. IBM-AS400.
10 000100 SPECIAL-NAMES.
11 000110 REQUESTOR IS CONSOLE.
12 000120 INPUT-OUTPUT SECTION.
13 000130 FILE-CONTROL.
14 000140 SELECT WORK-FILE ASSIGN TO DISK-WRK.
15 000150 SELECT CURRENT-SALES-FILE-IN ASSIGN TO DISK-CURRIN.
16 000160 SELECT CURRENT-SALES-FILE-OUT ASSIGN TO DISK-CURROUT.
17 000170 SELECT YTD-SALES-FILE-IN ASSIGN TO DISK-YTDIN.
18 000180 SELECT YTD-SALES-FILE-OUT ASSIGN TO DISK-YTDOUT.
19 000190 SELECT PRINTER-OUT ASSIGN TO PRINTER-QPRINT.
20 000200 DATA DIVISION.
21 000210 FILE SECTION.
22 000220 SD WORK-FILE
23 000230 DATA RECORD IS SALES-RECORD.
24 000240 01 SALES-RECORD.
25 000250 05 EMPL-NO PIC 9(6).
26 000260 05 DEPT PIC 9(2).
27 000270 05 SALES PIC 9(7)V99.
28 000280 05 NAME-ADDR PIC X(61).
29 000290 05 MONTH PIC X(2).
30 000300 FD CURRENT-SALES-FILE-IN
31 000310 LABEL RECORDS STANDARD
32 000320 DATA RECORD CURRENT-SALES-IN.
33 000330 01 CURRENT-SALES-IN.
34 000340 05 EMPL-NO PIC 9(6).
35 000350 05 DEPT PIC 9(2).
36 000360 88 ON-SITE-EMPLOYEE VALUES 0
37 000370 THRU 6, 8.
38 000380 05 SALES PIC 9(7)V99.
39 000390 05 NAME-ADDR PIC X(61).
40 000400 05 MONTH PIC X(2).
41 000410 FD CURRENT-SALES-FILE-OUT
42 000420 LABEL RECORDS STANDARD
43 000430 DATA RECORD CURRENT-SALES-OUT.
44 000440 01 CURRENT-SALES-OUT.
45 000450 05 EMPL-NO PIC 9(6).
46 000460 05 DEPT PIC 9(2).
47 000470 05 SALES PIC 9(7)V99.
48 000480 05 NAME-ADDR PIC X(61).
49 000490 05 MONTH PIC X(2).
```

Figure 119 (Part 1 of 3). Example of Use of SORT/MERGE


```

5738CB1 V2R2M0 910524 AS/400 COBOL Source
STMT SEQNBR -A 1 B...2...3...4...5...6...7...IDENTFCN S COPYNAME CHG DATE
47 000500 FD YTD-SALES-FILE-IN
48 000510 LABEL RECORDS STANDARD
49 000520 DATA RECORD YTD-SALES-IN.
50 000530 01 YTD-SALES-IN.
51 000540 05 EMPL-NO PIC 9(6).
52 000550 05 DEPT PIC 9(2).
53 000560 05 SALES PIC 9(7)V99.
54 000570 05 NAME-ADDR PIC X(61).
55 000580 05 MONTH PIC X(2).
56 000590 FD YTD-SALES-FILE-OUT
57 000600 LABEL RECORDS STANDARD
58 000610 DATA RECORD YTD-SALES-OUT.
59 000620 01 YTD-SALES-OUT.
60 000630 05 EMPL-NO PIC 9(6).
61 000640 05 DEPT PIC 9(2).
62 000650 05 SALES PIC 9(7)V99.
63 000660 05 NAME-ADDR PIC X(61).
64 000670 05 MONTH PIC X(2).
65 000680 FD PRINTER-OUT
66 000690 LABEL RECORDS OMITTED
67 000700 DATA RECORD PRINT-LINE.
68 000710 01 PRINT-LINE.
69 000720 05 RECORD-LABEL PIC X(25).
70 000730 05 DISK-RECORD-DISPLAY PIC X(80).
71 000740 WORKING-STORAGE SECTION.
72 000750 01 SALES-FILE-IN-EOF-STATUS PIC X VALUE "F".
73 000760 88 SALES-FILE-IN-END-OF-FILE VALUE "T".
74 000770 01 SALES-FILE-OUT-EOF-STATUS PIC X VALUE "F".
75 000780 88 SALES-FILE-OUT-END-OF-FILE VALUE "T".
76 000790 01 YTD-SALES-OUT-EOF-STATUS PIC X VALUE "F".
77 000800 88 YTD-SALES-OUT-END-OF-FILE VALUE "T".
78 000810 PROCEDURE DIVISION.
000820 OPEN-PRINTER-FILE SECTION.
000830 005-PRINTER-FILE.
79 000840 OPEN OUTPUT PRINTER-OUT.
000850 LIST-SORT-LIST-CURRENT-SALES SECTION.
000860 010-LIST-SORT-CURRENT-SALES.
80 000870 SORT WORK-FILE
000880 ON ASCENDING KEY DEPT OF SALES-RECORD
000890 ON DESCENDING KEY SALES OF SALES-RECORD
000900 INPUT PROCEDURE SCREEN-DEPT
000910 GIVING CURRENT-SALES-FILE-OUT.
000920 020-LIST-SORTED-SALES.
81 000930 OPEN INPUT CURRENT-SALES-FILE-OUT.
82 000940 PERFORM 100-PRINT-SALES-FILE-OUT
000950 THRU 110-END-PRINT-SALES-FILE-OUT
000960 UNTIL SALES-FILE-OUT-END-OF-FILE.
83 000970 CLOSE CURRENT-SALES-FILE-OUT.
000980 UPDATE-YEARLY-REPORT SECTION.
000990 040-MERGE-CURRENT-PREVIOUS.
84 001000 MERGE WORK-FILE
001010 ON ASCENDING KEY DEPT OF SALES-RECORD
001020 ON ASCENDING KEY EMPL-NO OF SALES-RECORD
001030 ON ASCENDING KEY MONTH OF SALES-RECORD
001040 USING YTD-SALES-FILE-IN
001050 CURRENT-SALES-FILE-IN
001060 GIVING YTD-SALES-FILE-OUT.
001070 040-PRINT-YTD-SALES-OUT.
85 001080 OPEN INPUT YTD-SALES-FILE-OUT.
86 001090 PERFORM 120-READ-PRINT-YTD-SALES-OUT
001100 UNTIL YTD-SALES-OUT-END-OF-FILE.
87 001110 CLOSE YTD-SALES-FILE-OUT
001120 PRINTER-OUT.
88 001130 STOP RUN.
001140 SCREEN-DEPT SECTION.
001150 060-S-D-1.
89 001160 OPEN INPUT CURRENT-SALES-FILE-IN
90 001170 PERFORM 070-READ-SELECT-DEPT THRU 080-END-READ-SELECT-DEPT
001180 UNTIL SALES-FILE-IN-END-OF-FILE.
91 001190 CLOSE CURRENT-SALES-FILE-IN.
92 001200 GO TO 090-END-S-D-1.
001210 070-READ-SELECT-DEPT.
93 001220 READ CURRENT-SALES-FILE-IN
94 001230 AT END MOVE "T" TO SALES-FILE-IN-EOF-STATUS
95 001240 GO TO 080-END-READ-SELECT-DEPT.

```

Figure 119 (Part 2 of 3). Example of Use of SORT/MERGE

```

5738CB1 V2R2M0 910524          AS/400 COBOL Source
STMT SEQNBR -A 1 B.+. . . . . 2. . . . . 3. . . . . 4. . . . . 5. . . . . 6. . . . . 7. IDENTFCN S COPYNAME  CHG DATE
 96 001250  MOVE "UNSORTED CURRENT SALES ",
    001260  TO RECORD-LABEL OF PRINT-LINE.
 97 001270  MOVE CURRENT-SALES-IN TO DISK-RECORD-DISPLAY.
 98 001280  WRITE PRINT-LINE.
 99 001290  IF ON-SITE-EMPLOYEE
100 001300  MOVE CURRENT-SALES-IN TO SALES-RECORD
101 001310  RELEASE SALES-RECORD.
    001320 080-END-READ-SELECT-DEPT.
    001330  EXIT.
102 001340 090-END-S-D-1.
    001350 END-SCREEN-DEPT SECTION.
    001360 100-PRINT-SALES-FILE-OUT.
103 001370  READ CURRENT-SALES-FILE-OUT
104 001380  AT END MOVE "T" TO SALES-FILE-OUT-EOF-STATUS
105 001390  GO TO 110-END-PRINT-SALES-FILE-OUT.
106 001400  MOVE "SORTED CURRENT SALES "
    001410  TO RECORD-LABEL OF PRINT-LINE.
107 001420  MOVE CURRENT-SALES-OUT TO DISK-RECORD-DISPLAY.
108 001430  WRITE PRINT-LINE.
    001440 110-END-PRINT-SALES-FILE-OUT.
    001450  EXIT.
109 001460 120-READ-PRINT-YTD-SALES-OUT.
110 001470  READ YTD-SALES-FILE-OUT
111 001480  AT END MOVE "T" TO YTD-SALES-OUT-EOF-STATUS
112 001490  GO TO 130-END-READ-PRT-YTD-SALES-OUT.
113 001500  MOVE "MERGED YTD SALES ",
    001510  TO RECORD-LABEL OF PRINT-LINE.
114 001520  MOVE YTD-SALES-OUT TO DISK-RECORD-DISPLAY.
115 001530  WRITE PRINT-LINE.
    001540 130-END-READ-PRT-YTD-SALES-OUT.
    001550  EXIT.
          * * * * *  E N D  O F  S O U R C E  * * * * *

```

Figure 119 (Part 3 of 3). Example of Use of SORT/MERGE

Appendix H. Example of a COBOL Formatted Dump

Figure 120 on page 368 shows an example of a COBOL formatted dump. A dump is usually available if something goes wrong when you try to run your program.

The following list describes the labeled areas of the figure:

- A** The exception for which the dump was requested and the location in the program where the exception occurred.
- B** The COBOL statement number of the last I-O operation that was run before the exception occurred. This information is produced only if at least one I-O operation has been processed.
- C** The current information for each file. This information is produced only if the program has files.
- D** Beginning of compiler-generated fields (included in the dump if you respond with an F option).
- E** I-O flags for the current file:

Bit	Meaning
1	File is open
2	File is locked
3	End of file
4	(Reserved)
5	Optional file
6	Check indexed file for duplicates at open
7	End of page
8	(Reserved).
- F** Previous status code.
- G** Beginning of Module Global Table (MGT).³
- H** Last exception code.
- I** Invocation number of current program.
- J** Qualified program name and library.
- K** Beginning of the Program Global Table (PGT).⁴
- L** Invocation number of the main COBOL program.
- M** Job date (YYMMDD).
- N** Beginning of user fields.
- O** Invalid zoned field printed in hexadecimal.

³ The Module Global Table (MGT) defines a common area for the module. The table is used to pass information to run-time subroutines.

⁴ The Program Global Table (PGT) is a communication area for the entire COBOL run unit. There is only one PGT for the run unit.

```

5738CB1 V2R2M0          AS/400 COBOL Source
STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S COPYNAME  CHG DATE
 1 000100 IDENTIFICATION DIVISION.                                03/07/89
 2 000200 PROGRAM-ID.      XMPLDUMP.                              03/22/89
 3 000300 AUTHOR.          PROGRAMMER NAME.                       03/07/89
 4 000400 INSTALLATION.    COBOL DEVELOPMENT CENTRE.             03/07/89
 5 000500 DATE-WRITTEN.    11/27/88.                             03/07/89
 6 000600 DATE-COMPILED.   05/24/91 12:21:54.                   03/07/89
 7 000700 ENVIRONMENT DIVISION.                                  03/07/89
 8 000800 CONFIGURATION SECTION.                                03/07/89
 9 000900 SOURCE-COMPUTER. IBM-AS400.                            03/07/89
10 001000 OBJECT-COMPUTER. IBM-AS400.                            03/07/89
11 001100 INPUT-OUTPUT SECTION.                                 03/07/89
12 001200 FILE-CONTROL.                                         03/07/89
13 001300 SELECT FILE-1 ASSIGN TO DISK-SALES.                   03/22/89
14 001400 DATA DIVISION.                                       03/07/89
15 001500 FILE SECTION.                                         03/07/89
16 001600 FD FILE-1                                             03/07/89
17 001700 LABEL RECORDS ARE STANDARD.                           02/17/89
18 001800 01 RECORD-1.                                          03/07/89
19 001900 05 R-TYPE          PIC X(1).                           02/17/89
20 002000 05 R-AREA-CODE    PIC 9(2).                            02/17/89
21 002100 88 R-NORTH-EAST VALUES 15 THROUGH 30.                02/17/89
22 002200 05 R-SALES-CAT-1  PIC S9(5)V9(2) COMP-3.              02/17/89
23 002300 05 R-SALES-CAT-2  PIC S9(5)V9(2) COMP-3.              02/17/89
24 002400 05 FILLER         PIC X(1).                            02/17/89
    002500
25 002600 WORKING-STORAGE SECTION.                               03/07/89
26 002700 01 W-SALES-VALUES.                                     02/17/89
27 002800 05 W-CAT-1        PIC S9(8)V9(2).                      02/17/89
28 002900 05 W-CAT-2        PIC S9(8)V9(2).                      02/17/89
29 003000 05 W-TOTAL        PIC S9(8)V9(2).                      02/17/89
    003100
30 003200 01 W-EDIT-VALUES.                                     02/17/89
31 003300 05 FILLER         PIC X(8) VALUE "TOTALS: ".          02/17/89
32 003400 05 W-EDIT-1       PIC Z(7)9.9(2)-.                    02/17/89
33 003500 05 FILLER         PIC X(3) VALUE SPACES.              02/17/89
34 003600 05 W-EDIT-2       PIC Z(7)9.9(2)-.                    02/17/89
35 003700 05 FILLER         PIC X(3) VALUE SPACES.              02/17/89
36 003800 05 W-EDIT-TOTAL   PIC Z(7)9.9(2)-.                    02/17/89
    003900
37 004000 01 END-FLAG       PIC X(1) VALUE SPACE.               02/17/89
38 004100 88 END-OF-INPUT VALUE "Y".                             02/17/89
    004200
39 004300 PROCEDURE DIVISION.                                    02/17/89
    004400*****
    004500* OPEN THE INPUT FILE, CLEAR TOTALS, CALL MAIN PROCESS THEN *
    004600* DISPLAY THE RESULTS AND END THE RUN.                  *
    004700*****
    004800 P-START.                                              02/17/89
40 004900 OPEN INPUT FILE-1.                                     02/17/89
41 005000 MOVE ZEROS TO W-SALES-VALUES.                          02/17/89
42 005100 PERFORM P-MAIN UNTIL END-OF-INPUT.                     02/17/89
    005200
43 005300 MOVE W-CAT-1 TO W-EDIT-1.                              02/17/89
44 005400 MOVE W-CAT-2 TO W-EDIT-2.                              02/17/89
45 005500 MOVE W-TOTAL TO W-EDIT-TOTAL.                          02/17/89
46 005600 DISPLAY W-EDIT-VALUES.                                 02/17/89
47 005700 STOP RUN.
    005800
    005900*****
    006000* READ THE INPUT FILE PROCESSING ONLY THOSE RECORDS FOR THE *
    006100* NORTH EAST AREA. WHEN END-OF-INPUT REACHED, SET THE FLAG *
    006200*****
    006300 P-MAIN.
48 006400 READ FILE-1 AT END SET END-OF-INPUT TO TRUE.
50 006500 IF R-NORTH-EAST AND NOT END-OF-INPUT
51 006600 ADD R-SALES-CAT-1 TO W-CAT-1, W-TOTAL
52 006700 ADD R-SALES-CAT-2 TO W-CAT-2, W-TOTAL.
          * * * * * E N D   O F   S O U R C E   * * * * *

```

```

MCH1202 exception in program XMPLDUMP in QTEMP at MI instruction number 005C COBOL statement number 51. A
Last I-O operation was at statement 48. B
LBE7903-Information pertaining to file FILE-1. C
LBE7905-File is open.
LBE7906-Last I-O operation completed for file was READ.
LBE7907-Last file status for file was 04.
LBE7910-Last extended file status for file was.

```

Figure 120 (Part 1 of 10). Example of a COBOL Formatted Dump

Appendix I. Glossary of Abbreviations

Abbreviation	Meaning	Explanation
Appl Dev Tools	Application Development Tools	Consisting of programs for the AS/400 system, such as the Screen Design Aid (SDA) and the Source Entry Utility (SEU).
ANSI	American National Standards Institute	An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.
ASCII	American National Standard Code for Information Interchange	The code developed by American National Standards Institute for information exchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 8-bit characters, consisting of 7-bit control characters and symbolic characters, plus one parity-check bit.
CICS	Customer Information Control Service	An IBM licensed program that enables transactions entered at remote work stations to be processed concurrently by user-written application programs. The licensed program includes functions for building, using, and maintaining databases, and for communicating with CICS on other operating systems.
CL	Control Language	The set of all commands with which a user requests system functions.
DBCS	Double-Byte Character Set	A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, displaying, and printing of DBCS characters requires hardware and programs that support DBCS. Four double-byte character sets are supported by the system: Japanese, Korean, Simplified Chinese, and Traditional Chinese. Contrast with single-byte character set.
DDM	Distributed Data Management	A function of the operating system that allows an application program or user on one system to use data files stored on remote systems. The systems must be connected by a communications network, and the remote systems must also be using DDM.
DDS	Data Description Specifications	A description of the user's database or device files that is entered into the system in a fixed form. The description is then used to create files.

Abbreviation	Meaning	Explanation
EBCDIC	Extended Binary-Coded Decimal Interchange Code.	A coded character set consisting of 256 eight-bit characters.
FIPS	Federal Information Processing Standard	An official standard to improve the utilization and management of computers and data processing in business.
ICF	Intersystem Communications Function	A function of the operating system that allows a program to communicate interactively with another program or system.
I/O	Input/Output	Data provided to the computer or data resulting from computer processing.
LVLCHK	Level Checking	A function that compares the record format-level identifiers of a file to be opened with the file description that is part of a compiled program to determine if the record format for the file changed since the program was compiled.
ODT	Object Definition Table	A table built at compile time by the system to keep track of objects declared in the program. The program objects in the table include variables, constants, labels, operand lists and exception descriptions. The table resides in the compiled program object.
OS/400	Operating System/400	The AS/400 operating system.
SDA	Screen Design Aid	A function of the AS/400 Application Development Tools licensed program that helps the user design, create, and maintain displays and menus.
SEU	Source Entry Utility	A function of the AS/400 Application Development Tools licensed program that is used to create and change source members.
SQL/400	Structured Query Language/400	An IBM licensed program supporting the relational database that is used to put information into a database and to get and organize selected information from a database.
UPSI	User Program Status Indicator switch	An external program switch that performs the functions of a hardware switch. Eight switches are provided: UPSI 0 - 7.

Note: The abbreviations for OS/400 commands do not appear here. Refer to the *CL Reference* for OS/400 commands and their usage.

Bibliography

For additional information about topics related to COBOL/400 programming on the AS/400 system, refer to the following IBM AS/400 publications:

- *Communications: Management Guide*, SC41-0024
Short title: *Communications Management Guide*
- *Device Configuration Guide*, SC41-8106
Short title: *Device Configuration Guide*
- *Licensed Programs and New Release Installation Guide*, SC41-9878
Short title: *Licensed Programs and New Release Installation Guide*
- *Languages: Systems Application Architecture* AD/Cycle* COBOL/400* Reference Summary*, SX09-1209
Short title: *COBOL/400* Reference Summary*
- *Data Description Specifications Reference*, SC41-9620
Short title: *DDS Reference*
- *Data Description Specifications Coding Form*, SX41-9891
Short title: *DDS Coding Form*
- *Communications: Intersystem Communications Function Programmer's Guide*, SC41-9590
Short title: *ICF Programmer's Guide*
- *System Operator's Guide*, SC41-8082
Short title: *Operator's Guide*
- *Basic Security Guide*, SC41-0047 and *Security Reference*, SC41-8083
Short titles: *Basic Security Guide* and *Security Reference*
- *Distributed Data Management Guide*, SC41-9600

Short title: *DDM Guide*

- *Database Guide*, SC41-9659
Short title: *Database Guide*
- *Utilities: Interactive Data Definition Utility User's Guide*, SC41-9657
Short title: *IDDU User's Guide*
- *System Programmer's Interface Reference*, SC41-8223
Short title: *System Programmer's Interface Reference*
- *CICS/400 Application Programming Guide*, SC33-0822
Short title: *CICS/400 Application Programming Guide*
- *Version 2 Release 1.1 Changes*, GC41-0029
Short title: *Version 2 Release 1.1 Changes*
- *Advanced Backup and Recovery Guide*, SC41-8079
Short title: *Advanced Backup and Recovery Guide*
- *IBM SAA AD/Cycle CoOperative Development Environment: Debug Tool User's Reference*, SC26-4664
Short title: *IBM SAA AD/Cycle CoOperative Development Environment: Debug Tool User's Reference*

For information about Systems Application Architecture (SAA) Common Programming Interface (CPI) COBOL, refer to the following publication:

- *Systems Application Architecture Common Programming Interface COBOL Reference*, SC26-4354.

Index

Special Characters

- / (slash) 12, 36
 - maximum number in a program 87
- * (asterisk) 12

Numerics

- 8-byte binary items, and performance 264

A

- abnormal program termination 50
- about this manual xi
- ACCEPT statement 101, 173, 339
- access mode 169, 237, 244
 - DYNAMIC 242
 - RANDOM 242
- access path
 - description 125
 - example for indexed files 243
 - file processing 246
 - specifications 103
- *ACCUPDALL option 26
- *ACCUPDNE option 26
- ACQUIRE statement 174
- ADDMSGD (Add Message Description)
 - command 327
- ADDRESS OF special register 276, 283
 - description 283
 - difference from calculated ADDRESS OF 283
- addresses
 - incrementing using pointers 301
 - passing between programs 299
- ADM/400 2
- ADVANCING phrase 229
 - for FORMATFILES 230
- ADVANCING PAGE phrase 332
- ALCOBJ (Allocate Object) command 91
- ALIAS keyword 109
- alias name 111
- alias, definition 109
- *ALL option 27
- Allocate Object (ALCOBJ) command 91
- alphabet-name, definition 331
- alphabetic character, definition 332
- ALTER statement 308
- American National Standards Institute (ANSI) xiii, 1, 319, 327, 379

American National Standards Institute (ANSI)

(continued)

- ANSI 74 COBOL versus ANSI 85 COBOL 331
 - conforming to standards
 - with indexed files 237
 - with relative files 245
 - with sequential files 245
 - standard xiii, 1, 327
- APIs (Application Programming Interfaces)
 - error-handling 51, 68
 - using with pointers 287
- *APOST option 20
- Application Development Manager/400 2
- Application Development Tools, messages 323
- Application Programming Interfaces (APIs)
 - error-handling 51, 68
 - using with pointers 287
- arguments, describing in the calling program 276
- arithmetic operators 3
- arrival sequence 125, 245, 246
- arrows, shown in syntax 4
- ASSIGN clause 87, 139, 168
 - device name 87
- assignment name 87, 138, 139, 168, 337
- * (asterisk) 12
- AT END condition 78, 183, 186
- *ATR option 21
- ATTRIBUTE DATA 174
- attributes
 - of data items 44
 - of files 44
 - of table items 45
- ATTRIBUTES field 44
- AUT parameter for CRTCLPGM command 27
- authorization-list-name option 27

B

- batch compiles 35
- batch jobs, representation of DBCS data in 343
- binary items, and performance 264
- *BLANK option 19
- BLANK WHEN ZERO
 - defining with LIKE clause 254
- *BLK option 23
- block, description 100

- blocking output records 100
- Boolean data types 20, 140, 172
- Boolean literal, definition 20
- boundary
 - definition 93
 - record 23
 - violation 71, 247
- breakpoints
 - as an OS/400 function 53
 - considerations for using 61
 - description 55
 - displaying table elements 58
 - displaying variables 58
 - example 55
 - traces, differences between 62
 - use of 55, 60
- browsing a compiler listing
 - See source entry utility (SEU)
- BY CONTENT, definition 275
- BY REFERENCE, definition 275

C

- calculation operations
 - on fixed-length fields 128
- call by identifier 278
- CALL statement
 - BY CONTENT identifier 276
 - BY CONTENT LENGTH OF identifier 276
 - BY CONTENT literal 276
 - BY CONTENT, implicit MOVE 285
 - by identifier 278
 - BY REFERENCE ADDRESS OF
 - record-name 276
 - BY REFERENCE identifier 275
 - recursive, description 269
 - to QCMDEXC 251
 - using pointers 285
 - within a segmented program 308
- called program
 - definition 269
- calling programs
 - BY CONTENT 275
 - BY REFERENCE 275
 - definition 269
 - to begin at another entry point
 - using pointers 285
 - within a segmented program 308
- calling the COBOL compiler 15
- CANCEL statement 278, 303, 332
 - with non-COBOL programs 275

- *CBL statement 37
- CCSIDs (Coded Character Set Identifiers) 133
- CDRA (Character Data Representation Architecture) 133
- Change Debug (CHGDBG) command 53
- *CHANGE option 27
- Change Program Variable (CHGPGMVAR)
 - command 61
 - change/date (CHGDATE) field 42
 - changes from ANSI 74 COBOL 331–332
 - changing the value of variables 61
- Character Data Representation Architecture (CDRA) 133
- characters, double-byte 333
- characters, replaced in field name 111
- checking DBCS literals 335
- checking work station validity 136
- CHGDBG (Change Debug) command 53
- CHGPGMVAR (Change Program Variable)
 - command 61
- choices, shown in syntax 4
- CICS (Customer Information Control System)
 - statements 13
- CICSCBL member type 13
- CICSSQLCBL member type 13
- CL (control language) commands
 - for running programs 7
 - for testing programs 53
 - issuing using QCMDEXC in a program 251
- clauses
 - ACCESS MODE 169
 - ASSIGN 139, 168
 - CONTROL-AREA 170
 - CURRENCY clause 12
 - DECIMAL-POINT clause 12
 - FILE STATUS 101, 169
 - INDICATOR 141
 - JUSTIFIED 338
 - LIKE 141
 - LINAGE 229
 - OCCURS 141, 337
 - ORGANIZATION 168
 - ORGANIZATION IS INDEXED 237
 - PICTURE 141, 263, 338
 - RECORD KEY 126
 - REDEFINES 331, 337
 - RELATIVE KEY 169
 - RENAMES 338
 - REPLACING identifier-1 BY identifier-2
 - clause 12
 - SAME AREA 331

clauses (*continued*)

- SAME RECORD AREA 331
- SEGMENT-LIMIT 306
- SORT-MERGE AREA 331
- syntax, notation for 3
- USAGE 141
- VALUE 141, 338
- WITH DEBUGGING MODE 309

CLOSE operation 23

CLOSE statement 175, 332

closing files with the CANCEL statement 332

code optimizing 22

CODE/400 2, 20

Coded Character Set Identifiers (CCSIDs) 133

coding errors 54

coding form 6, 11

coding formats provided by SEU 11

coding tables 260

command summary listing 39

command syntax, using 3

commands

- Add Message Description (ADDMSGD) 327
- Allocate Object (ALCOBJ) 91
- Change Debug (CHGDBG) 53
- Change High Level Language Pointer (CHGHLTPTR) 61
- Change Pointer (CHGPTR) 61
- Change Program Variable (CHGPGMVAR) 61
- Create Authorization List (CRTAUTL) 27
- Create COBOL Program (CRTCBLPGM)
 - See Create COBOL Program command
- Display Program Variable (DSPPGMVAR) 61
- Display Trace Data (DSPTRCDTA) 63
- End COBOL Debug (ENDCBLDBG) 311, 312
- Grant Object Authority (GRTOBJAUT) 27
- Monitor Message (MONMSG) 16
- Override Message File (OVRMSGF) 327
- Override to Diskette File (OVRDKTF) 88
- Start COBOL Debug (STRCBLDBG) 310, 312
- Start Debug (STRDBG) 53
- Start Source Entry Utility (STRSEU)
 - See source entry utility

comments with DBCS characters 336

COMMIT statement 92, 94

commitment boundary, definition 93

commitment control 80, 92, 96

- example 95
- locking level 93

common keys 125

Common Programming Interface (CPI) support 321

communication module 321

communications, interactive

- interprogram considerations 269, 345
- recovery 80
- with other programs 135
- with remote systems 135
- with work station users 135

COMP-3 items, and performance 264

compile-time errors 54

compiler failure 16

compiler options

- See *also* PROCESS statement
- also parameters, CRTCBLPGM command
- *ACCUPDALL 26
- *ACCUPDNE 26
- *ALL 27
- and syntax checking with SEU 12
- *APOST 20
- as specified in PROCESS statement 31
- *ATR 21
- authorization-list-name option 27
- batch compiling 35
- *BLANK 19
- *BLK 23
- *CHANGE 27
- check for sequence errors 19
- check subscript ranges at run time 21
- compiler options listing 36, 40
- count verb usage 19
- create cross-reference listing 19, 21, 46
- create Data Division map 20
- create object code 19
- create source listing 19, 40
- *CRTF 22
- *CURLIB 18, 24
- *CURRENT 26, 30
- *DATETIME 24
- *DDSFILLER 22
- *DEB1 25
- *DEB2 25
- delimiter for nonnumeric and Boolean literals 20
- *DFRWRT 25
- *DUMP 21
- *DUPKEYCHK 22
- *EXCLUDE 27
- *EXTACCDSP 23
- file-name option 24
- *FLAG 25, 36
- *FS21DUPKY 23
- *GEN 19

compiler options (*continued*)

- *GRAPHIC option 24
- *HIGH 25
- include attributes for the IRP 21
- *INTERMEDIATE 25
- *INZDLT 23
- *LIBCRTAUT 27
- *LIBL 18, 24
- library-name option 18, 25
- *LINENUMBER 20
- *LIST 21
- list compiler options in effect 36, 40
- *LSTDBG 21
- *MAP 20, 36
- maximum-severity-level option 24
- message-limit option
 - *MINIMUM 25
 - *NOATR 21
 - *NOBLK 23
 - *NOCRTF 22
 - *NODATETIME option 24
 - *NODDSFILLER 22
 - *NODEB 25
 - *NODFRWRT 25
 - *NODUMP 21
 - *NODUPKEYCHK 22
 - *NOEXTACCDSP 23
 - *NOFIPS 25
 - *NOFLAG 25
 - *NOFS21DUPKY 23
 - *NOGEN 19
 - *NOINZDLT 23
 - *NOLIST 21
 - *NOLSTDBG 21
 - *NOMAP 20
 - *NOMAX 24
 - *NONUMBER 19
 - *NOOBSOLETE 25
 - *NOOPTIMIZE 22
 - *NOOPTIONS 20
 - *NOPATCH 21
 - *NOPRINT 21
 - *NOPRTCORR 20
 - *NORANGE 22
 - *NOSECLVL 20
 - *NOSEG 25
 - *NOSEQUENCE 19
 - *NOSOURCE 19
 - *NOSRCDBG 20
 - *NOSTDERR 22
 - *NOSTDINZ 23

compiler options (*continued*)

- *NOSYNC 22
- *NOUNDSPCHR 25
- *NOUNREF 22
- *NOVARCHAR 23
- *NOVBSUM 19
- *NOXREF 19, 21
- *NUMBER 20
- *OBSOLETE 25
- *OPTIMIZE 22
- optimizing source code 22
- *OPTIONS 20, 36
- overview 6
- *OWNER 26
- parameters of the CRTCLPGM
 - command 18—31
- *PATCH 21
- *PGM 18
- *PGMID 18
- *PRINT 21
- PROCESS statement, using to specify 31
- program listings, DBCS characters in 345
- program-name 18
- *PRTCORR 20
- *PRV 26, 30
- QLBLSRC (default source file) 18
- QSYSVRT (default printer file) option 24
- *QUOTE 20
- *RANGE 21
- release-level option 26, 30
- *SECLVL 20
- *SEG1 25
- *SEG2 25
- *SEQUENCE 19
- severity-level option 19, 26
- *SOURCE 18, 19, 36
- source-file-member-name option 19
- source-file-name option 18
- *SRCDBG 21
- *SRCMBRTXT 19
- *STDERR 22
- *STDINZ 23
- suppressing second-level messages 20
- suppressing source listing 40
- *SYNC 22
- text-description 19
- *UNDSPCHR 25
- *UNREF 22
- *USE 27
- use compiler-generated sequence numbers 20

compiler options (*continued*)
 use user-supplied sequence numbers 20
 *USER 26
 V2R1M0 value for release-level option 31
 V2R1M1 value for release-level option 31
 V2R2M0 value for release-level option 31
 *VARCHAR 23
 *VBSUM 19, 36
 *XREF 19, 21, 36
 compiler output
 See *also* messages
 browsing 37
 See *also* source entry utility
 command summary listing 39
 compiler output 35, 36
 cross-reference listing 46
 CRTCBLPGM options 36
 Data Division map 43
 description 36
 examples 36
 FIPS messages listing 45
 listing descriptions 36
 listing options 40
 messages 325
 options listing 39, 40
 program listings, DBCS characters in 345
 suppressing source listing 40
 compiling COBOL programs
 abnormal compiler termination 16
 compiler-detected errors 54
 example listing 39
 exceeding internal size limits 15
 failed attempts 16
 for extended ACCEPT/DISPLAY 23
 for the previous release 30
 invoking the compiler 15
 messages 325
 multiple programs 35
 output 36
 redirecting files 88
 TGTRLS, using 30
 CONCAT keyword 120
 Configuration Section, description 10, 336
 conforming to ANSI standards 327
 constant, NULL figurative 282
 contents of DEBUG-ITEM special register 315
 contiguous items, definition 238
 contiguous key fields, multiple 238
 control
 returning 270
 transferring 269
 CONTROL-AREA clause 170–171
 control language commands
 See CL commands
 control of segmentation 306
 *CONTROL statement 37
 COPY DDS, use with indicators 116
 control, returning from a called program 270
 control, transferring to another program 269
 CoOperative Development Environment/400 2, 20
 copies of ANSI standard available xiii
 COPY statement
 and DBCS characters 344
 and externally described data 110, 111
 and floating-point 124
 changes from ANSI 74 COBOL 331
 data field structures 114
 DD, DDR, DDS, or DDSR 110
 DDS results 108, 116
 description 110
 example of data structures generated by 200
 examples of key generation 119
 format-1 COPY statement 35
 in File Section 112
 key fields 238
 listing source statements 37
 outside File Section 112
 PROCESS statement containing COPY statement 35
 suppressing source statements 37
 use with PROCESS statement 35
 use with TRANSACTION files 135
 with ALL-FORMATS 112
 COPYNAME field 42
 corresponding options, PROCESS and CRTCBLPGM 32
 CORRESPONDING phrase 252
 counting verbs in a source program 19, 42, 48
 CPI (Common Programming Interface)
 support 321
 Create Authorization List (CRTAUTL)
 command 27
 Create COBOL Program (CRTCBLPGM)
 command
 AUT parameter 27
 CVTOPT parameter 23, 33
 description of 6
 DUMP parameter 27
 entering from CL program 28
 entering from command line 28
 EXTDSPOPT parameter 34

Create COBOL Program (CRTCLPGM)
 command (*continued*)
 FLAG parameter 26, 34
 FLAGSTD parameter 25, 34, 36
 GENLVL parameter 19, 32
 GENOPT parameter 21, 33
 ITDUMP (n) parameter 27
 MSGLMT parameter 24
 OPTION parameter 19, 32, 36
 parameters, description of 18–31
 PGM parameter 18
 prompt displays, using 16
 PRTRFILE parameter 24
 REPLACE parameter 26
 SAAFLAG parameter 25, 34, 36
 SRCFILE parameter 18
 SRCMBR parameter 18
 syntax of 29
 TEXT parameter 19
 TGTRLS parameter 26
 USRPRF parameter 26

creating files
 indexed files 347, 352
 relative files 347, 357
 sequential files 347

cross-reference listing
 and breakpoints 55
 CRTCLPGM options 19, 21
 description of listing 46
 example 46
 testing, using in 59

CRTAUTL (Create Authorization List)
 command 27

CRTCLPGM command
 See Create COBOL Program command

*CRTF option 22
 *CURLIB option 18, 24
 *CURRENT option 26, 30

Customer Information Control System (CICS)
 statements 13

CVTOPT parameter 23, 33

D

data area
 description 301
 local 302
 PIP 302

data class type (TYPE) field 44

data communications file 135, 168

data description entry for Boolean data 140

data description specifications (DDS)
 command attention (CA) keys 136
 CONCAT keyword 120
 Create File commands 102
 date fields 129
 DD option, description 111
 DDR option, description 111
 DDS option, description 111
 DDSR option, description 111
 definition 136
 description 104
 display management 136
 examples
 CONCAT keyword 120
 for a display device file 137
 for field reference file 105
 for subfile record format 156, 157
 formats, data structures generated by 200
 key generation 119
 keyed access path for an indexed file 243
 RENAME keyword 122
 specifications for a database file 108
 specifying a record format 107
 SST keyword 123
 work station programs 196, 227

externally described files 102, 238

FORMATFILE files 230

function keys 136
 function of 136

graphic data fields 129

incorporate description in program 106

key fields 238

multiple device files 158

program-described files 102

RENAME keyword 122

SAA fields 129

SST keyword 123

subfiles 153

suffixes 121

time fields 129

timestamp fields 129

TRANSACTION files 135

use of keywords 104

variable-length fields 127

work station validity checking 136

Data Division
 arguments for calling program 276
 Boolean data facilities 172
 DBCS characters 337
 description 10

Data Division (*continued*)

- map of, compiler option 20, 43
- transaction files 169, 171
- data errors with de-editing, handling 263
- data field 10
- data field structures 114
- data item
 - attributes of 44
 - defining as a pointer 279
 - in subprogram linkage 277
 - passing, with its length 276
 - substring reference 258
- data items, unreferenced 22
- data types 127
 - date 129
 - graphic 129
 - restrictions for SAA data types 129
 - time 129
 - timestamp 129
- data, passing
 - BY CONTENT and BY REFERENCE 276
 - in groups 277
- database files
 - See also disk files
 - DATABASE file considerations 237
 - DATABASE versus DISK 237
 - DISK file considerations 237
 - processing methods 237
- date data type 129
- date-last-modified area 10
- *DATETIME option 24
- DBCS support
 - See double-byte character set support
- DBCS-graphic data type 129
- DD name 110
- DDR name 110
- DDS
 - See data description specifications
- DDS name 110
- *DDSFILLER option 22
- DDSR name 110
- de-editing 262
 - definition 262
 - examples 262
- *DEB1 option 25
- *DEB2 option 25
- debugging programs 53, 309
 - breakpoints
 - considerations for using 61
 - description 55
 - changing variable contents 61

debugging programs (*continued*)

- compile-time switch 309
- Data Division map and IRP listing, using 59
- debug module 321
- DEBUG-CONTENTS 315
- DEBUG-ITEM special register 315
- DEBUGGING MODE as compile-time switch 309
- declaratives, running of 312
- description 6, 53, 265
- displaying table elements 58
- displaying variables 58
- ENDCBLDBG (End COBOL Debug)
 - command 311, 312
- features available 309
- file status 101
- formatted dump 65
- functions for 53
- line, definition 317
- lines of a source program 317
- OS/400 functions for 53
- overview 6
- run-time switch 65, 310
- STRCBLDBG (Start COBOL Debug)
 - command 310, 312
- traces
 - considerations for using 64
 - description 62
 - USE FOR DEBUGGING procedures 312
- default source file (QLBLSRC) 10, 18
- default values, indication of 16
- DEFINED field 47
- delays, reducing length of on initialization 247
- deleted records, initializing files with 23, 247
- delimiting SQL statements 12
- descending file considerations 249
- descending key sequence, definition 249
- description and reference numbers flagged
 - field 45
- designing your program 9
- destination of compiler output 35
- device control information 138
- device dependence 87
 - examples 88
- device files
 - and I/O 87
 - DATABASE file considerations 237
 - DISK file considerations 237
 - multiple 158
 - single 158

- device independence 87
- *DFRWRT option 25
- diagnostic levels 326
- diagnostic messages 47
- diagrams, syntax 29
- direct files
 - See relative files
- disclaimers
 - availability of products ix
 - examples ix
 - patents ix
 - sending information to IBM ii
 - small programs ix
 - technical inaccuracies ix
 - typographical errors ix
 - US government users ii
- disk files 237
 - processing methods 248
- displacement (DISP) field 44
- display device
 - DDS for 136
 - record format 136, 137
- display device file 136
- display format data, definition 136
- DISPLAY statement 340
- Display Trace Data (DSPTRCDTA) command 63
- displaying a compiler listing 37
- displays
 - CRTCBLPGM prompt display 17
 - data description specifications (DDS) for 136
 - display program messages 324
 - ENDCBLDBG prompt display 311
 - for sample programs
 - order inquiry 212, 213
 - payment update 225, 226
 - transaction inquiry 201
 - SEU display messages 323
 - STRCBLDBG prompt display 310
 - subfiles 154
- distributed data management (DDM) 379
- DIVIDE statement 332
- divisions of programs
 - Data Division 20, 169, 171, 337
 - Environment Division 167, 306, 307, 336
 - Identification Division 10
 - optional 10
 - Procedure Division 172, 306, 307, 338–344
 - required 10
- do while structure, testing for end of chained list 301
- double spacing 37
- double-byte character set (DBCS)
 - support 333–346
 - ACCEPT statement 339
 - and alphanumeric data 342
 - checking 335
 - comments with DBCS characters 336
 - communications between programs 345
 - definition 379
 - enabling in COBOL programs 333
 - graphic 344
 - in the Data Division 337
 - in the Environment Division 336
 - in the Identification Division 336
 - in the Procedure Division 338–344
 - open 344
 - PROCESS statement 333, 341
 - representation of DBCS data in batch jobs 343
 - searching for in a table 344
 - sorting 344
 - specifying DBCS literals 333, 334
- DROP statement 175
- DSPTRCDTA (Display Trace Data) command 63
- *DUMP option 21
- DUMP parameter for CRTCBLPGM command 27
- dump, formatted 367
- *DUPKEYCHK option 22, 265
- duplication errors 254
- dynamic access mode 155, 169, 244, 248
- dynamic file creation 22
- dynamic processing, definition 169

E

- EBCDIC character, definition 380
- editing source programs 9
 - See also source entry utility (SEU)
- efficiency considerations 264
- efficiency, increased 22
- eight-byte binary items, and performance 264
- EJECT statement 36
- elementary pointer data items 283
- embedded SQL 12
- End COBOL Debug (ENDCBLDBG) command 311, 312
- end of chained list, testing for 301
- END-OF-PAGE phrase 332
- END-READ phrase 183, 186

- END-REWRITE phrase 188
- END-WRITE phrase 195
- ending a called program 270
- entering CRTCLPGM from CL program 28
- entering CRTCLPGM from command line 28
- entering source programs 9, 11
- entering your program
 - See source entry utility (SEU)
- Environment Division
 - and DBCS characters 336
 - and transaction files 167
 - SEGMENT-LIMIT clause 306, 307
- error checking for de-editing, run time 263
- error handling 67
 - APIs 51, 68
 - nonstandard 75
 - overview 67
 - standard 74
- error recovery, example 80
- errors
 - ADVANCING phrase with FORMATFILE files 230
 - duplication 254
- errors to avoid 54
- errors, in syntax
 - See syntax errors
- examples
 - access path for indexed file 243
 - breakpoint 55
 - COBOL and files 103
 - commitment control 92, 96
 - compiler options listing 20, 36
 - COPY DDS results 108, 117
 - COPY statement in PROCESS statement 35
 - cross-reference listing 46
 - Data Division map 43
 - DDS
 - CONCAT keyword 120
 - for a display device file 136, 137
 - for a record format 107
 - for a record format with ALIAS keyword 109
 - for field reference file 105
 - for multiple device files 158
 - for subfiles 156, 157
 - key generation 119
 - RENAME keyword 122
 - SST keyword 123
 - diagnostic messages listing 47
 - END-OF-PAGE condition 231
 - entering CRTCLPGM from command line 28
 - examples (*continued*)
 - error recovery 80
 - externally described printer files 233
 - file processing
 - indexed files 352, 353
 - relative files 357, 359
 - sequential files 347, 349
 - FIPS messages listing 45
 - FORMATFILE file 230
 - formatted dump 367
 - generic START 238, 239
 - indicators 143
 - LENGTH OF special register with pointers 282
 - length of variable-length field 128
 - MOVE with pointers 284
 - multiple device files 161
 - pointers
 - aligning 280
 - and LENGTH OF special register 282
 - and REDEFINES clause 280
 - and results of MOVE 284
 - initializing with NULL 282
 - passing items containing 285
 - processing chained list 298
 - program structure 9
 - record format specifications 105, 108
 - ROLLING phrase 191
 - run units
 - multiples, running concurrently 274
 - multiples, running consecutively 272
 - single unit 270
 - with shared program 273
 - SEU display messages 323
 - source listing 40
 - trace 62
 - using pointers in chained list 298
 - variable-length graphic data 131
 - verb usage by count listing 42
 - work station application programs
 - order inquiry 202
 - payment update 213
 - transaction inquiry 196
- exceptions 16, 50, 69, 79
- *EXCLUDE option 27
- exclusive-allow-read lock state 91
- EXIT PROGRAM statement 270, 303
- expressions 261, 339
- *EXTACCDSP option 23
- EXTDSPOPT parameter of the CRTCLPGM command 34

- EXTEND mode, definition 91
- extended ACCEPT and DISPLAY statements 23
- extensions, IBM
 - double-byte character set (DBCS)
 - support 333–346
 - flagging 25, 327
 - format, indication in syntax 5
 - GOBACK
 - overview 1
 - reading 5
 - transaction files 135–227
- extensions, list of 1
- external description
 - adding functions to 126
 - overriding functions to 126
- external file status 68
- externally described files 110, 232
 - adding functions 126
 - advantages of using for printer files 230
 - and COPY statement, DD, DDR, DDS, DDSR
 - format 117
 - considerations for using 103
 - DDS for 106
 - description 102
 - level checking 126
 - overriding functions 126
 - printer files, specifying with
 - FORMATFILE 230
- EXTERNALLY-DESCRIBED-KEY 238
- externally described TRANSACTION
 - files 135–138

F

- failed I/O and record locking 92
- failure of compiler 16
- Federal Information Processing Standard (FIPS)
 - 1986 COBOL standard 327
 - description 327
 - flagging deviations from 25, 327, 345
 - FLAGSTD parameter 25, 45
 - messages 45, 325, 327
 - options 25
 - standard modules 327
 - standards to which the compiler adheres xiii
 - with DBCS characters 345
- FIB (file information block) 69
- field names
 - DDS added to 121, 122
 - additional notes 124
 - construction of 114

- fields
 - attributes
 - BLANK WHEN ZERO 256
 - defaults 256
 - SIGN IS TRAILING 256
 - USAGE IS DISPLAY 256
 - BLANK WHEN ZERO attribute 256
 - date 129
 - fixed length 128
 - floating-point 124
 - null-capable 129
 - time 129
 - time separator 37
 - timestamp 129
 - variable-length 127
 - character 127, 128
 - graphic 128, 130
 - length of, example 128
 - maximum length 127
 - restrictions 128
- figurative constant QUOTE 20
- figurative constant, NULL 282
- file and record locking 91, 93
- file boundaries 247
- file considerations 87, 237, 302
- file control entry 87
 - of Environment Division 167
 - TRANSACTION file processing entry 167
- file descriptions 106, 171
- file information block (FIB) 69
- file locking 91
- file-name option 24
- file operations for printer file 229
- file organization 246
- file processing
 - See files
- file redirection 88, 90
- file status
 - OQ 247
 - 9N 81
 - 9Q 247
 - after I/O 81
 - coded examples 349
 - from message monitors 71
 - how it is set 70
 - internal and external 68
 - obtaining 101
 - statements that affect 303
- FILE STATUS clause 101
- files
 - See also disk files, externally described files, program-described files, source files

files (*continued*)

- access paths 246
- attributes of 44
- closing 332
- creation of
 - indexed 347, 352
 - relative 347, 357
 - sequential 347
- DATABASE 237
- DATABASE versus DISK 237
- description 347
- DISK 237
- examples
 - indexed files 352, 353
 - relative files 357, 359
 - sequential files 347, 349
- external description 103
- FORMATFILE 230
- indexed organization 237
- keys 125
- logical 242
- methods of 246
- on AS/400 systems 87, 347
- preserving sequence of records 246
- PRINTER 229
- processing methods 237
- redirecting access to 88
- relative 244
- relative organization 244
- retrieval of, relative 347, 361
- sample programs 347–363
- sequential 245
- sequential organization 245
- specific 229
- techniques for processing 347–363
- TRANSACTION 135

files unavailable at OPEN time 22

FILLER

- floating-point fields 124

FIPS flagging

- See Federal Information Processing Standard

FIPS-ID field 45

FIPS violations flagged, total 46

fixed length graphic fields 130

*FLAG option 25, 36

FLAG parameter for CRTCLPGM

- command 26, 34

FLAGSTD parameter for CRTCLPGM

- command 25, 34, 45

float test, de-editing 263

- floating-point fields 124
- FOOTING phrase 332
- format (record) level structures 113
- format-1 COPY statement 35
- format-2 COPY statement 16
- format names, additional notes 124
- FORMAT phrase 177, 182, 185, 187
- FORMATFILE files
 - description 230
 - sample program 230
- formats, using SEU
 - See source entry utility
- formatted dump 53, 65, 367
- *FS21DUPKY option 23
- function keys
 - and CONTROL-AREA clause 171
 - specifying with DDS
 - See transaction files
- functional processing modules 319

G

- *GEN option 19
- general-use programming interfaces
 - description x
 - error-handling 68
 - QCMDEXC 28, 251
- generation of I/O formats 116
- generation of keys 119
- generation of message monitors 71
- generic START statement 238
- GENLVL parameter for CRTCLPGM
 - command 19, 32
- GENOPT parameter for CRTCLPGM
 - command 21, 33
- GIVING phrase 331, 332
- GO TO statement 308
- GOBACK statement 303
- Grant Object Authority (GRTOBJAUT)
 - command 27
- graphic data types 129
 - restrictions 130
- *GRAPHIC option 24
- group level names 114
- group structures, aligning pointers within 280
- GRTOBJAUT (Grant Object Authority)
 - command 27

H

- handling data errors, de-editing 263

- *HIGH option 25
- highlights 2, 347
- hyphen, produced when copying ALIAS names 111

I

- I/O formats 116
- IBM extensions
 - double-byte character set (DBCS) support 333–346
 - flagging 25, 327
 - format, indication in syntax 5
 - GOBACK
 - overview 1
 - reading 5
 - transaction files 135–227
- ICF
 - See intersystem communications function
- Identification Division
 - and DBCS characters 336
 - description 10
- identifier
 - call by 278
 - defining in Working-Storage section 254
 - unreferenced 22
- increasing efficiency 22
- INDARA keyword 116
- independence, device 87
- independent segments 305
- indexed files
 - creation 347, 352
 - description 237
 - key fields 237
 - processing methods for types DISK and DATABASE 237
 - updating 347, 353
- indexed files, definition 352
- indexed I-O module 320
- indicator structures 115
- indicators
 - and ASSIGN clause 139
 - and Boolean data items 140
 - and COPY statement 112, 116
 - associated with command keys 136
 - data description entries 140
 - description 113, 138
 - example, using in programs 143
 - in a separate indicator area 138, 139, 142, 265
 - in the record area 139, 142
 - INDARA DDS keyword 138

- indicators (*continued*)
 - INDICATOR clause 141
 - INDICATORS phrase 141
 - performance considerations 265
 - sample programs 143
 - special considerations for 141
 - structures 114
 - TRANSACTION file processing 138
 - using 140
- industry standards xiii
- initialization of storage 275
- initializing files with deleted records 247
- initializing pointers 282
 - with NULL figurative constant 282
- input field 136, 185
- input records 100
- input spool 89, 90
- input verbs, processing of
 - since Version 1, Release 3 78
- Input-Output formats 116
- input-output verbs, processing of
 - since Version 1, Release 3 78
- INSPECT statement 341
- inter-program module 320
- *INTERMEDIATE option 25
- intermediate representation of program (IRP)
 - cross-reference listing 21
 - how to list 21
 - how to list attributes 21
 - how to use 55
 - sample listing 60
- internal file status 68
- internal name (I-NAME) field 44
- internal size limits 15
- International Standards Organization (ISO) xiii
- interprogram calls using pointers 285
- interprogram communication
 - considerations 269
- intersystem communications function (ICF)
 - ACCESS MODE clause 169
 - ASSIGN clause 168
 - communications 153
 - CONTROL-AREA clause 170
 - FILE STATUS clause 169
 - multiple and single device files 158
 - ORGANIZATION clause 168
 - RELATIVE KEY clause 169
 - using to specify subfiles 153
- INTO phrase 182, 331
- introduction to COBOL/400 1

- invalid characters 111
 - DDR and DDSR options 111
- INVALID KEY phrase 186, 188, 195
 - role since Version 1, Release 3 78
- *INZDLT option 23, 247
- I-O feedback 53, 101, 339
- I-O flags 367
- I-O operation 367
- I/O verbs, processing of
 - since Version 1, Release 3 78
- IRP (intermediate representation of program)
 - See intermediate representation of program (IRP)
- ITDUMP parameter for CRTCLPGM
 - command 27
- items grouped by level 46

J

- job failure, recovery 80
- JUSTIFIED clause 338

K

- key fields
 - contiguous, multiple 238
 - descending keys 249
 - for indexed files 237
 - name, -DDS added to 121, 122
 - partial keys 238
 - program-defined 242
- keyed read 89
- keyed sequence 125, 237, 246, 249
- keys
 - common 125
 - floating-point 124
 - generation of 119
 - record 125
 - validity 238
- keywords
 - DDS 109, 120, 122, 123
 - in syntax diagrams 3
 - INDARA 116

L

- language elements
 - See program structure
- last-used state, description 270
- LDA (local data area) 302
- length (LENGTH) field 44

- length of records in source file 10
- LENGTH OF special register 276, 282
- length of statement, maximum 11, 12
- level checking 126
- level of data item (LVL) field 43
- level of language support 319, 320, 327
- *LIBCRTAUT option 27
- *LIBL option 18, 24
- libraries, test 53
- library-name option 18, 25
- LIKE clause
 - description 254
 - format of 255
 - PICTURE portion 256
- limitations 87
 - TGTRLS parameter 31
- limits, internal, size 15
- LINAGE clause 229
- *LINENUMBER option 20
- linkage items, setting the address of 283
- Linkage Section
 - describing data to be received 277
 - parameters for a called program 276
- *LIST option 21
- listings
 - browsing
 - See source entry utility
 - command summary 39
 - compiler options in effect 20
 - cross-reference 46
 - Data Division map 43, 59
 - DBCS characters in 345
 - default output file 24
 - example, source listing 19, 40, 42
 - examples of 40
 - FIPS messages 45
 - messages
 - description 47
 - example 47
 - from COBOL/400 compiler 325
 - minimum record length 24
 - options 40
 - scanning for syntax errors 38
 - specifying output file for 24
 - verb usage by count 19, 42
- literals, DBCS 334–336, 343, 344
- literals, delimiting 20
- local data area (LDA), definition 302
- lock level
 - high, under commitment control 93
 - low, under commitment control 93

- lock state 91
- locking, file and record 91
- logic of segmentation 306
- logical file considerations 242
- logical operators 3
- looking at a compiler listing
 - See source entry utility (SEU)
- loops in a program 266, 267
- *LSTDBG option 21

M

- main program, description 269
- major/minor return codes 73
- *MAP option 20, 36
- maximum record length, dynamically created files 22
- maximum-severity-level option 24
- maximum source statement length 11, 12
- member type
 - See source member type
- members 90
- memory management
 - See segmentation
- MERGE statement 308, 331, 344, 364
- message files 327
- message-limit option 24
- message monitor generation 71
- messages
 - Application Development Tools 323
 - compilation 325
 - compile-time 323
 - diagnostic 47
 - field on diagnostic messages listing 48
 - FIPS 325
 - interactive 323
 - responding to in an interactive environment 325
 - run-time 324
 - and standard error handling 67
 - SAA, flagged 46
 - severity levels 19, 24, 326
 - statistics 48
 - types 323
- methodology for entering programs 9
- migrating
 - ANSI 74 COBOL programs 331
 - to ANSI 85 COBOL 331
 - to COBOL/400 language 331
- *MINIMUM option 25

- mismatched records, reducing occurrence 277
- module global table (MGT), definition 367
- Monitor Message (MONMSG) command 16
- monitoring exceptions 16
- monitoring operations
- monitors, message 71
- MONMSG (Monitor Message) command 16
- MOVE statement 315, 342
 - CORRESPONDING phrase 252
 - using pointers 283
- MSGID and severity level field 47
- MSGLMT parameter 24
- multiple contiguous key fields 238
- multiple device files 158—166, 174, 180, 186
- multiple members 90

N

- name, assignment 87, 138, 139, 168, 337
- names defined when GENOPT(*NOUNREF) specified 15
- NAMES field 47
- NEXT MODIFIED phrase 185
- NO DATA phrase 183
 - role since Version 1, Release 3 78
- NO LOCK phrase, and performance 91, 266
- NO REWIND phrase 332
- *NOATR option 21
- *NOBLK option 23
- *NOCRTF option 22
- *NODATETIME option 24
- *NODDSFILLER option 22
- *NODEB option 25
- *NODFRWRT option 25
- *NODUMP option 21
- *NODUPKEYCHK option 22
- *NOEXTACCDSP option 23
- *NOFIPS option 25
- *NOFLAG option 25
- *NOFS21DUPKY option 23
- *NOGEN option 19
- *NOGRAPHIC option 24
- *NOINZDLT option 23
- *NOLIST option 21
- *NOLSTDBG option 21
- *NOMAP option 20
- *NOMAX option 24
- nonnumeric literals 20
- nonstandard error handling 75
- nonstandard language extensions
 - See IBM extensions

- *NONUMBER option 19
- *NOBSOLTE option 25
- *NOOPTIMIZE option 22
- *NOOPTIONS option 20
- *NOPATCH option 21
- *NOPRINT option 21
- *NOPRTCORR option 20
- *NORANGE option 22, 265
- *NOSECLVL option 20
- *NOSEG option 25
- *NOSEQUENCE option 19
- *NOSOURCE option 19
- *NOSRCDBG option 20
- *NOSTDERR option 22
- *NOSTDINZ option 23
- *NOSYNC option 22
- NOT AT END phrase 183, 186
 - role since Version 1, Release 3 78
- NOT INVALID KEY phrase 186, 188, 195
 - role since Version 1, Release 3 78
- notation, syntax 3
- notices
 - description ix
 - patents ix
 - products announced, unavailable ix
 - revision bars ix
- *NOUNDSPCHR option 25
- *NOUNREF option 22
- *NOVARCHAR option 23
- *NOVBSUM option 19
- *NOXREF option 19, 21
- nucleus module 320
- null-capable fields 129
- NULL figurative constant 282
- null values 129, 300
- *NUMBER option 20
- numeric items
 - and performance 264
 - moving with de-editing 262
- numeric usage clauses 114

O

- Object Definition Table (ODT) 22
- object names, OS/400 16
- object program
 - compiler options, specifying 21
 - optimizing, specifying at compile-time 22
 - output from compiler 15
 - specifying authority to 27
 - subscript range checking 21

- object program, definition 6
- *OBSOLETE option 25
- OCCURS clause 337
- ODP (open data path) 92
- ODT (Object Definition Table) 22
- offset, relative to 16-byte boundary 285
- open data path (ODP) 92
- OPEN operation, increasing speed of 92
- OPEN statement 176
- OPEN type 91
- OPEN-FEEDBACK 339
- operation, more efficient 22
- operators, arithmetic and logical 3
- *OPTIMIZE option 22
- optimizing code 22
- OPTION parameter for CRTCLPDM
 - command 19, 36
- optional clauses 3
- optional divisions 10
- optional items, syntax 3
- optional processing modules 320
- options
 - data formats, *NORANGE option 265
 - for the PROCESS statement 35
 - listing 40
 - of CRTCLPDM command
 - parameters 18–31
 - segmentation 307
- OPTIONS listing 40
- *OPTIONS option 20, 32, 36
- order of clauses 3
- ORGANIZATION clause 168
- ORGANIZATION IS INDEXED clause 237
- OS/400 operating system
 - and messages 327
 - breakpoint commands 55
 - device control information 138
 - device independence and device dependence 87
 - functions for debugging 53
 - input/output 138
 - internal size limits 15
 - object names 16
 - security, maintaining while testing 53
 - testing, functions for 53
- output
 - compiler 36
 - compiler, displaying 37
 - output field 136
 - output file, definition 347

- output spool 89
- output verbs, processing of
 - since Version 1, Release 3 78
- Override Message File (OVRMSGF)
 - command 327
- Override to Diskette File (OVRDKTF)
 - command 88
- overriding messages 327
- overriding program specifications 90
- overview 6
- OVRDKTF command 88
- OVRMSGF command 327
- *OWNER option 26

P

- packed decimal items 264
- paper positioning 229
- parameters of CRTCLPGM command 18
 - See also Create COBOL Program (CRTCLPGM) command
- parameters, describing in the called program 276
- partial key, referring to 238
- parts of a COBOL program
 - See program structure
- parts of a program 9
- passing addresses between programs 299
- passing data 275
 - in groups 277
- passing data item and its length 276
- *PATCH option 21
- PERFORM statement 308, 332
- performance considerations 264
 - I/O operations 100
- permanent segment 305
- *PGM option 18
- PGM parameter for CRTCLPGM command 18
- *PGMID option 18
- phrases
 - ADVANCING 229
 - ADVANCING PAGE 332
 - AT END 183, 186
 - CORRESPONDING 20
 - END-OF-PAGE 332
 - END-REWRITE 188
 - END-WRITE 195
 - FOOTING 332
 - FORMAT 177, 182, 185, 187
 - GIVING 331, 332
 - INDICATORS 141, 178
 - INTO 182, 331

- phrases (*continued*)
 - INVALID KEY 186, 188, 195
 - NEXT MODIFIED 185
 - NO DATA 183
 - NO REWIND 332
 - NOT AT END 183, 186
 - NOT INVALID KEY 186, 188, 195
 - REEL/UNIT 332
 - RELATIVE KEY 331
 - REMAINDER 332
 - ROLLING 191
 - STARTING 190
 - SUBFILE 178
 - TERMINAL 178, 183, 186, 188, 189, 194
 - USING 331
- PICTURE clause 263, 338
 - and performance 264
 - defining with LIKE clause 254
- PICTURE definitions 114
- PIP (program initialization parameters) data area 302
 - description 302
- pointer alignment, definition 279
- pointer data items
 - definition 278
 - elementary items 283
- pointers
 - aligning on boundaries
 - 01-level items 280
 - 77-level items 280
 - automatically using FILLER 280
 - with blocking in effect 280
 - and REDEFINES clause 280
 - assigning null value 300
 - defining 279
 - defining alignment 279
 - definition 278
 - description 278
 - examples
 - accessing user space 287
 - processing chained list 298
 - in CALL statement 285
 - in File Section 280
 - in Linkage Section 277
 - in MOVE statement 283
 - restrictions 283
 - in records 281
 - in tables 280
 - in Working-Storage 280
 - initializing 282
 - length of 278

- pointers (*continued*)
 - manipulating data items 279
 - moving between group items 285
 - null value 300
 - processing a chained list 298
 - reading 281
 - writing 281
- portability considerations
 - See segmentation
- position of PROCESS statement 31
- preface xi
- prestart job 302
- previous release, compiling for 30
- *PRINT option 21
- printer file, default 24
- printing
 - based on indicators 230
 - editing field values 230
 - in overflow area 230
 - maintaining print formats 230
 - multiple lines 230
 - output from job with WRITE statement 229
 - paging 229
 - paper positioning 229
 - spacing 229
- procedure branching statements 343
- Procedure Division
 - and DBCS characters 338
 - and transaction files 172
 - changes from ANSI 74 COBOL 331
 - description 10
 - segmentation 307
 - specifying debugging in 312
 - subdivisions in 305
 - USE FOR DEBUGGING declarative 312
 - using SET statement to specify address 283
- PROCESS statement 333
 - allowable options for 32
 - compiler options specified in 31
 - compiler output 36
 - considerations
 - blocking output records 100
 - commitment control considerations 92
 - DATABASE files 237
 - device dependencies 87
 - DISK files 237
 - file and record locking 91
 - overriding program specifications 90
 - overview 251
 - processing methods for types DISK and DATABASE 237
 - program-described and externally described files 102
 - PROCESS statement (*continued*)
 - considerations (*continued*)
 - spooling 89
 - unblocking input records 100
 - COPY statement, using with 35
 - description 31
 - format of 31
 - options 35
 - position of statement 31
 - rules for 31
 - scope of options with CRTCLPLGM command 35
 - specifying compiler options 40
 - techniques
 - file processing 347
 - indexed file creation 352
 - indexed file updating 353
 - relative file creation 357
 - relative file retrieval 361
 - relative file updating 359
 - sequential file creation 347
 - sequential file updating and extension 349
 - using to specify compiler options 31
 - processing methods for DATABASE files 237
 - processing methods for DISK files 237
 - processing of I/O verbs
 - since Version 1, Release 3 78
 - program control
 - returning 270
 - transferring 269
 - program-described files
 - considerations for using 103
 - description 102
 - externally described by DDS with Create File commands 102
 - TRANSACTION files 135
 - program global table (PGT), definition 367
 - program initialization parameters (PIP) data area
 - See PIP data area
 - program listings, DBCS characters in 345
 - program loops 266
 - program-name 18
 - program object
 - compiler options, specifying 21
 - optimizing, specifying at compile-time 22
 - output from compiler 15
 - specifying authority to 27
 - subscript range checking 21
 - program parts 9
 - program patch area 21

- program segments 305
- program size 22
- program stack, definition 269
- program structure
 - Data Division 171
 - Data Division map 43
 - data field 114
 - Environment Division 167
 - example 9, 10
 - format (record) level 113
 - Identification Division 10
 - indicator 114, 115
 - level of language support 321
 - Procedure Division 172
 - required and optional divisions 10
 - skeleton program 9
- program syntax, debugging line 317
- program template 21
- program termination
 - abnormal 50
 - and the CALL statement 308
 - file considerations 269
 - initialization 275
 - returning control 270
 - STOP RUN statement 270
 - with the CANCEL statement 332
- program variables
 - changing 61
 - pointers 61
- program-defined key fields 242
- programming considerations 251
- programming notes
 - number of entries in Object Definition Table (ODT) 15, 22
- prompts, using SEU
 - See source entry utility
- *PRTCORR option 20
 - example listing 252
- PRTFILE parameter for CRTCLPGM
 - command 24
- *PRV option 26, 30
- punctuation 3
- purpose of this manual xi

Q

- QCMDEXC, using in a program 28, 251
- QLBLMSG compile-time message file 327
- QLBLMSGGE run-time message file 327
- QLBLSRC (default source file) 10, 18

- QLRCHGCM API 68
- QLRRTVCE API 68
- QLRSETCE API 51, 68
- QSYSPRT (default printer file) option 24
- quadruple spacing 37
- *QUOTE option 20
- QUOTE, figurative constant, value of 20

R

- random processing, definition 361
- *RANGE option 21
 - reference modification 258
- READ statement 340
 - changes in the use of ANSI 74 COBOL 331
 - description 179
 - format, nonsubfile 182—183
 - format, subfile 185—186
 - indicators 142
 - processing facilities 178
 - FORMAT phrase 177
- READ WITH NO LOCK 91, 93
- record boundary 23
- record format
 - composition for display device 136
 - DDS for subfiles 156, 157
 - example, record format specification 103, 105, 108
 - fields 136
 - indicators 138
 - specification, use of DDS keywords in 104
 - subfiles 154
- RECORD KEY clause 126
 - EXTERNALLY-DESCRIBED-KEY 126
- record keys 125
- RECORD KEYS, valid 238
- record length of source file 10
- records
 - blocking output 100
 - containing pointers 281
 - locking
 - and failed I/O 92
 - and performance 266
 - by COBOL 91
 - updating database records 91
 - preserving sequence of 246
 - reducing mismatches 277
 - unblocking input 100
- recovery 80
 - example 81
 - procedure in program 81
 - with multiple acquired devices 81
 - with one acquired device 81

- recovery (*continued*)
 - transaction files 80
 - with commitment control 80
- recursive call, definition 269
- REDEFINES Clause 337
 - for ALL-FORMATS or I/O phrases 117
 - pointer data item as subject or object 280
- redefinition of formats 117
- redefinition, group level name 114
- redirecting files 88, 90
- REEL/UNIT phrase 332
- reference modification
 - and *RANGE option 21
 - and INSPECT statement 260
 - calculating offset 285
 - description 258
 - left-justification 260
 - retrieving time value 259
- reference numbers 19, 42, 47
- REFERENCES field 47
- references to other manuals xi
- referring to a partial key 238
- register 315
- reinitialization, avoiding 117
- related printed information 381
- relative files
 - and performance 265
 - creating 347, 357
 - definition 244
 - for OPEN OUTPUT 265
 - in COBOL 244
 - initializing for output 246
 - retrieval of 347, 361
 - sequential access 23
 - updating 347, 359
- relative I-O module 320
- RELATIVE KEY clause 169
- RELATIVE KEY phrase 331
- relative key, definition 155
- release-level option 26, 30
- RELEASE statement 344
- releasing a record read for update 91
- REMAINDER phrase 332
- remote systems, communications between 135, 302
- RENAME keyword 122
- RENAMES clause 338
- REPLACE parameter for CRTCLPGM
 - command 26
- REPLACING, in format 2 COPY 124
- reply modes 50
- report writer module 321
- required
 - clauses 3
 - divisions 10
 - items, in syntax 3
- reserved word list, and extended
 - ACCEPT/DISPLAY 23
- reserved word, -DDS added to 121, 124
- responding to messages in an interactive environment 325
- restrictions 87
- return codes 73
- return of control from called program 270
- RETURN statement 344
- REUSEDLT option
 - See reusing deleted records
- reusing deleted records
 - indexed files 238
 - relative files 245
 - sequential files 246
- Revoke Object Authority (RVKOBJAUT)
 - command 27
- REWRITE statement
 - and DBCS 340
 - description 187
 - for program-described transaction files 187
 - for TRANSACTION file 187
 - format 187, 188
 - indicators 142
 - processing facilities 177, 178
- ROLLBACK statement 92
 - boundary 93
- ROLLING phrase 191
- run time
 - common errors 54
 - debugging 65, 310
 - debugging switch 309
 - error handling, de-edit 263
 - messages 324
 - and standard error handling 67
 - monitoring exceptions 16
 - program termination 50
 - redirecting files 88
 - subscript range checks, specifying 21
 - switch 65, 310, 311
- run unit
 - definition 35, 269
 - examples
 - multiples, running consecutively 272
 - single run unit 270
 - with a shared program 273, 274

running COBOL/400 programs
description 49
system reply list and reply modes 50
RVKOBJAUT (Revoke Object Authority)
command 27

S

S in PICTURE clause 264
SAA Common Programming Interface (CPI)
support 321
SAA CPI (Common Programming Interface)
support 321
SAA data types
SAA flagging 46, 329
SAAFLAG parameter for CRTCLPGM
command 25, 34
screens
See displays
SEARCH statement 344
searching DBCS characters in a table 344
*SECLVL option 20
SECTION field 43
security
maintaining while testing 53
specifying authority to object program 27
*SEG1 option 25
*SEG2 option 25
SEGMENT-LIMIT clause 306
segment-numbers 305–307
segmentation 265, 305–308, 321, 364
segmented program 305
SELECT statement,
EXTERNALLY-DESCRIBED-KEY 119
separate indicator area (SI) attribute 138
sequence
combining numbers 20
errors, checking for 19
number 10
of records, preserving 246
sequence error indicator (S) 42
*SEQUENCE option 19
sequential access mode 23, 169, 183, 185, 245,
246
sequential files
creation 245, 347
definition 245
in COBOL 245
updating and extension 347, 349
sequential I-O module 320

service marks x
SET statement 342
SEU (source entry utility)
browsing a compiler listing 37
editing source programs 6, 9, 11
entering source programs 6, 9, 11
errors
coding errors 54
common errors 54
detected by compiler 54
listing 47
messages at run time 324
formats, using 11
prompts and formats 11
Start Source Entry Utility (STRSEU)
command 9
syntax-checking 11–13, 323
TYPE parameter 9
severity-level 19, 26
severity level of messages 19, 24, 326
shared files 91
shared-for-read 91
shared-for-read lock state 91
shared-for-update 91
shared-no-update 91
shared ODP (open data path) 92
shared records 91
shift-in character, definition 334
shift-out character, definition 334
SIGN clause, defining with LIKE clause 254
sign in PICTURE clause, and performance 264
sign test, de-editing 263
single device files 158
size limits, internal 15
skeleton program 9
SKIP statement 37
SKIP1 statement 37
SKIP2 statement 37
SKIP3 statement 37
slash (/) 12, 36
SORT statement 308, 344, 364
sort-merge module 320
SORT/MERGE statement 344
Source Entry Utility
See SEU
source file
default 10
fields 10
logical structure, specifying 168
program, suppressing listing 40
record length 10

- source file format
 - description 10
 - program segments 305
 - record length 10
- source-file-member-name 19
- source-file-name 18
- source language debugging 309
- source listing, example 40
- source member type
 - CICSCBL 13
 - CICSSQLCBL 13
 - compiling 16
 - specifying 9, 11
 - SQLCBL 13
 - syntax-checking 11, 13
- SOURCE NAME field 43
- *SOURCE option 19, 36
- source program
 - compiling 15, 309
 - counting verbs used 19
 - DEBUG-ITEM special register 315
 - debugging lines 317
 - definition 2
 - editing source programs
 - See SEU (source entry utility)
 - entering source programs
 - See SEU (source entry utility)
 - listing 19, 40
 - WITH DEBUGGING MODE switch and compilation 309
- source text manipulation module 320
- space pointer, definition 278
- spacing 37
- special registers
 - ADDRESS OF 276
 - LENGTH OF 276
 - implicit definition 282
 - in Procedure Division 282
- SPECIAL-NAMES paragraph 12, 229, 331
- specific file processing 229
- spooling 89, 90
- SQL (Structured Query Language)
 - statements 12
- SQLCBL member type 13
- *SRCDBG option 21
- SRCFILE parameter for CRTCLPGM
 - command 18
- SRCMBR parameter for CRTCLPGM
 - command 18
- *SRCMBRTXT option 19
- SST keyword 123
- stacks, shown in syntax 4
- standard error handling 67, 74
- standard, ANSI X3.23-1985 xiii
- Start COBOL Debug (STRCBLDBG)
 - command 310, 312
- Start Debug (STRDBG) command 53
- Start Source Entry Utility (STRSEU) command 9
- START statement 238, 340
- STARTING phrase 190
- starting the compiler 15
- statement length, maximum 11, 12
- statement number (STMT) field 43, 47
- statement number, compiler-generated (STMT) 42
- statements
 - ACCEPT 101, 173, 339
 - ACQUIRE 174
 - ALTER 308
 - arithmetic, in DBCS processing 341
 - breakpoints 55
 - CALL 308
 - CANCEL 332
 - CLOSE 175, 332
 - COMMIT 92
 - compiler output 36
 - COPY 102, 112, 331, 344
 - DISPLAY 340
 - DIVIDE 332
 - DROP 175
 - EJECT 36
 - in syntax diagrams 3
 - INSPECT 341
 - MERGE 308, 331, 344, 364
 - MOVE 315, 342
 - OPEN 176
 - PERFORM 308, 332
 - PROCESS 31, 333
 - READ 179, 331, 340
 - RELEASE 344
 - RETURN 331, 344
 - REWRITE 187, 340
 - ROLLBACK 92
 - SEARCH 344
 - SET 342
 - SKIP 37
 - SORT 308, 364
 - START 340
 - START, generic 238
 - STOP 343
 - STRING 342

- statements (*continued*)
 - UNSTRING 342
 - USE 195
 - WRITE 189, 332, 340
- *STDERR option 22
- *STDINZ option 23
- STOP RUN statement 270, 303
- STOP statement 343
- storage optimization
 - See segmentation
- storage, initialization of 275
- storage, using less 22
- STRCBLDBG (Start COBOL Debug)
 - command 310, 312
- STRDBG (Start Debug) command 53
- STRING statement 342
- STRSEU (Start Source Entry Utility) command 9
- structure, program
 - See program structure
- Structured Query Language (SQL)
 - statements 12
- subfield contents, DEBUG-ITEM special register 315
- subfiles 153–155, 178
- subprogram 35, 269
 - linkage 277
- subscript range checking, specifying 21
- subscript ranges 21
- subscripting 261
- substitution character (X'3F') in data 133
- suffix -DDS
 - added to key field name 121, 122
 - added to reserved word 121, 124
- summary of changes xv
 - changes made in Version 2 Release 1.1 xv
 - changes made in Version 2 Release 2 xvi
- support for ANSI X3.23-1985 standard 319
- suppressing source listing 40
- suppression of messages 326
- switch, run-time 65, 310, 311
- symbols used in syntax 3
- *SYNC option 22
- syntax
 - arrows 4
 - checking, in SEU 11, 12, 38
 - checking, unit of 11
 - debugging lines 317
 - diagrams, using 3
 - keywords in 3
 - notation 3
 - of CRTCLPDM command 29

- syntax (*continued*)
 - optional items 3
 - punctuation 3
 - required and optional clauses 3
 - required items 3
 - stacks 4
 - symbols 3
- system override considerations 90
- system reply list 50

T

- table items, attributes of 45
- table, reference modification 260
- target release 26, 30
- template, program 21
- TERMINAL phrase 178, 183, 186, 188, 189, 194
- termination, program 50, 332
- testing COBOL/400 programs
 - and debugging 53
 - breakpoints 55, 61
 - changing variable contents 61
 - Data Division map and IRP listing, using 59
 - displaying table elements 53, 58
 - displaying variables 58
 - file status 101
 - formatted dump 65
 - OS/400 functions for 53
 - overview 6
 - security, maintaining 53
 - test libraries 53
 - traces 62, 64
- text-description 19
- TEXT parameter for CRTCLPDM command 19
- TGTRLS parameter for CRTCLPDM
 - command 26, 30
- time data type 129
- time value, retrieving 259
- time-separation characters 37
- timestamp data type 129
- TITLE statement 37
- tools for entering source programs 9
- traces
 - as an OS/400 function 53
 - considerations 64
 - description 62
 - example 62
 - using 62
- tracing a loop 266
- trademarks x

- transaction files
 - ACCESS MODE clause 169
 - and subfiles 155
 - ASSIGN clause 168
 - Boolean data facilities 172
 - CONTROL-AREA clause 170
 - data description specifications (DDS) for 135, 136
 - Data Division considerations 169, 171
 - defining 135
 - description 135
 - display management 136
 - Environment Division considerations 167
 - externally described 135
 - file control entry and Environment Division 167
 - file description entry and Environment Division 171
 - FILE STATUS clause 169
 - file status, setting of 69
 - major return code 69
 - minor return code 69
 - ORGANIZATION clause 168
 - organization of 168
 - Procedure Division considerations 172
 - processing externally described 138
 - program-described
 - RELATIVE KEY clause 169
 - return codes 69
 - sample programs, work station 196
 - transferring control to another program 269
 - transferring program control 269
 - triple spacing 37

U

- UFCB (user file control block) 69
- unattended mode, running the program 327
- unblocking input records 100
- underscores, removed from end of field name 111
- underscores, translated to hyphens 111
- *UNDSPCHR option 25
- unit of syntax checking 11
- *UNREF option 22
- unreferenced data items 22
- unreferenced identifiers 15
- UNSTRING statement 342
- updating
 - and extension of sequential files 347, 349
 - indexed files 347, 353
 - relative files 347, 359

- updating (*continued*)
 - sequential files 349
- UPSI (user program status indicator) switch 380
- USAGE clause
 - defining with LIKE clause 254
 - numeric 114
 - USAGE IS POINTER 278
 - with transaction files 141
- USE FOR DEBUGGING declarative 312, 313
 - in the Procedure Division 312
 - using the procedures 313
- *USE option 27
- USE procedure
 - role since Version 1, Release 3 78
- USE statement
 - coded examples 349, 350
 - description 195
 - EXCEPTION/ERROR for TRANSACTION file 195
 - format 195
- user file control block (UFCB) 69
- *USER option 26
- user profile 26
- user program status indicator (UPSI) switch 380
- user spaces
 - accessing using APIs 287
 - using a subfile for display 153—155
 - using double-byte characters 333
 - using less storage 22
- USING phrase 331
- using REPLACING in format 2 COPY statement 124
- using the COBOL/400 language
 - See COBOL/400 language
- USRPRF parameter for CRTCLPGM command 26

V

- V2R1M0 option 31
- V2R1M1 option 31
- V2R2M0 option 31
- valid RECORD KEYS 238
- validity checking 136
- VALUE clause 338
- VALUE IS NULL 300
- value of figurative constant QUOTE 20
- *VARCHAR option 23
- variable-length fields 127
 - defining 127
 - example of 127, 130, 131
 - length of, example of 128

- variable-length fields (*continued*)
 - maximum length of 127
 - restrictions 128
- variables, changing values while testing 61
- *VBSUM option 19, 36
- verbs usage by count listing 42

W

- where DBCS characters can be used 336
- WITH DEBUGGING MODE switch and compilation 309
- work stations
 - communications between 135
 - sample programs
 - order inquiry 202
 - payment update 213
 - transaction inquiry 196
 - validity checking 136
- Working-Storage section
 - defining identifiers 254
- WRITE statement
 - and DBCS 340
 - changes from ANSI 74 COBOL 332
 - description 189
 - for program-described transaction files 189
 - for TRANSACTION file 189
 - format, nonsubfile 189—191
 - format, subfile 194—195
 - indicators 142
 - processing facilities 177—178

X

- X'3F' (substitution character) in data 133
- *XREF option 19, 21, 36

Readers' Comments

Application System/400

Languages:

Systems Application Architecture AD/Cycle

COBOL/400 User's Guide

Version 2

Publication No. SC09-1383-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, ON Canada M3C 1V7

Fold and Tape

Please do not staple

Fold and Tape

Readers' Comments

Application System/400

Languages:

Systems Application Architecture AD/Cycle

COBOL/400 User's Guide

Version 2

Publication No. SC09-1383-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, ON Canada M3C 1V7

Fold and Tape

Please do not staple

Fold and Tape

Readers' Comments

Application System/400

Languages:

Systems Application Architecture AD/Cycle

COBOL/400 User's Guide

Version 2

Publication No. SC09-1383-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, ON Canada M3C 1V7

Fold and Tape

Please do not staple

Fold and Tape

Readers' Comments

Application System/400

Languages:

Systems Application Architecture AD/Cycle

COBOL/400 User's Guide

Version 2

Publication No. SC09-1383-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, ON Canada M3C 1V7

Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5738-CB1

Printed in Ireland by Printech International plc

SC09-1383-01

