Interactive COBOL

Language Reference

&

Developer's Guide

ICOBOL Revision 3.60

No. 011-00100-13

01/11/2008

Much of the material in this manual is extracted from the ANSI X.3-1985 COBOL Standard, generally referred to as the ANSI COBOL 85 Standard. Accordingly, the following acknowledgment is made as required in that document.

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

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

The authors and copyright holders of the copyrighted materials used herein are:

> FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC (R) I and II, Data Automation System copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

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

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Chairman of the CODASYL COBOL Committee, P.O. 1808, Washington, DC 20013.

LICENSE AGREEMENT

Carefully read the following terms and conditions. **Use of this product constitutes your acceptance of these terms and conditions and your agreement to abide by them.**

You, the purchaser, are granted a non-exclusive license to use this software under the terms stated in this agreement. The program and its documentation are copyrighted and may not be copied or reproduced in any part, in any form, for any purpose, except according to the terms stated in this agreement.

You may:

1. use the software for up to the number of active users for which the software was purchased.

2. use the software provided a valid license is installed for the required number of active users to be supported at any one time.

3. copy the software into any machine readable form for backup purposes.

4. transfer the software from one computer to another.

5. assign or transfer the software and license to another party if the other party agrees to all the terms and conditions of this agreement. Once the transfer is complete you must destroy any copies of the software not transferred.

6. rent, sublicense, or lease the software and license if the user agrees to all the terms and conditions of this agreement.

7. not alter, modify, or adapt the software itself, including, but not limited to, translating, decompiling, or disassembling.

8. copy or reproduce the documentation for purposes of using a valid license.

This license and your right to use the software automatically terminate if you fail to comply with any provision of this License Agreement. You agree upon such termination to destroy the software and license.

LIMITED WARRANTY

Envyr Corporation warrants that (a) the software will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of receipt; and (b) any hardware accompanying the software will be free from defects in materials and workmanship under normal use and service for a period of one (1) year from the date of receipt. Any implied warranties on the software and hardware are limited to ninety (90) days and one (1) year respectively. Some states do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you.

Envyr Corporation's entire liability and your exclusive remedy shall be, at Envyr Corporation's option, either (a) return the license fee or (b) repair or replacement of the software or hardware that does not meet the above Limited Warranty and which is returned to the original vendor with a copy of the receipt. This Limited Warranty is void if failure of the software or hardware has resulted from accident, abuse, or misapplication.

In no event shall Envyr Corporation or its suppliers be liable for any damages whatsoever, including, but without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss, arising out of the use of or inability to use this software or hardware, even if Envyr Corporation has been advised of the possibility of such damages.

Restricted Rights Legend: Use, duplication, or disclosure by the U. S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at [DFARS] 252.227-7013 (October 1988).

Envyr Corporation
4904 Waters Edge Drive, Suite 160
Raleigh, N.C. 27606

NOTICE

This manual has been prepared for use only with the Interactive COBOL product by prospective customers or valid licensees. The information in this manual is preliminary and subject to change without prior notice.

In no event shall the seller be liable for any incidental, indirect, special or consequential damages whatsoever (including but not limited to lost profits) arising out of or related to this document or the information contained in it, even if the writers have been advised, knew or should have known of the possibility of such damage.

Program and Manual Copyright © 1995-96, 1998-2004, 2007 by Envyr Corporation, Raleigh, N.C. All rights reserved.

Revision History:

Release 2.20	- September 1996
Release 2.40	- June 1998
Release 2.60	- October 1999
Release 3.00	- August 2000
Release 3.01	- September 2000
Release 3.02	- October 2000
Release 3.03	- December 2000
Release 3.10	- April 2001
Release 3.20	- April 2002
Release 3.30	- February 2003
Release 3.40	- March 2004
Release 3.60	- January 2008

Effective with:
Interactive COBOL Revision 3.60

TRADEMARKS

ICHOST is a trademark of Envyr Corporation

AIX, PC, PC/XT, PC/AT, PS/2, and RISC System/6000 are trademarks of International Business Machines Corporation.
AOS/VS and DG/UX are trademarks of Data General Corporation.
AViiON and INFOS are registered trademarks of Data General Corporation.
IBM, PC-DOS, and RT are registered trademarks of International Business Machines Corporation.
Intel is a registered trademark of Intel Corporation.
Microsoft and MS-DOS are registered trademarks of Microsoft Corporation.
SunOS and Solaris are trademarks of Sun Microsystem, Inc.
Transoft is a registered trademark of Transoft, Ltd.
U/FOS is a trademark of Transoft, Ltd.
UNIX is a trademark of UNIX Systems Laboratories, Inc. (USL)

All other product names mentioned herein are trademarks of their respective owners.

TABLE OF CONTENTS

TABLE OF CONTENTS . . . 7

LIST OF EXAMPLES . . . 20

LIST OF FIGURES . . . 21

LIST OF SCREENS . . . 22

LIST OF TABLES . . . 22

PREFACE . . . 25

ENHANCEMENTS . . . 27

PART ONE - LANGUAGE REFERENCE . . . 31

I. CONVENTIONS USED IN THIS MANUAL . . . 33
A. Definition of a General Format . . . 33
1. Elements . . . 33
2. Words . . . 33
3. Level-Numbers . . . 33
4. Brackets and Braces . . . 33
5. Ellipsis . . . 33
6. Format Punctuation . . . 34
7. Use of Special Character Words in Formats . . . 34
8. Documentation Only . . . 34
B. Rules . . . 34
1. Syntax Rules . . . 34
2. General Rules . . . 34
C. ICOBOL Dialects and Feature-Sets . . . 35
1. Description of **ICOBOL** Dialects . . . 35
2. Notation of Dialect Differences . . . 35
3. Description of Feature-sets . . . 36
4. Notation of Feature-set Differences . . . 37

II. COBOL SOURCE PROGRAM . . . 39
A. General Description . . . 39
B. Concepts . . . 39
1. Character Set . . . 39
2. Language Structure . . . 39
2.1 Separators . . . 39
2.2 Character-Strings . . . 40
2.2.1 COBOL Words . . . 40
2.2.2 Literals . . . 43
.1 Nonnumeric Literals . . . 44
.2 Nonnumeric Hexadecimal Literals . . . 45
.3 Numeric Literals . . . 45
.4 Numeric Hexadecimal Literals . . . 46
.5 Figurative Constant Values . . . 46
.6 Date Literals (***ISQL***) . . . 48
.7 Time Literals (***ISQL***) . . . 48
.8 Timestamp Literals (***ISQL***) . . . 49
.9 Interval Literals (***ISQL***) . . . 49
.9.1 Year-Month Interval Literals (***ISQL***) . . . 50

.9.2 Day-Time Interval Literals (***ISQL***). . . . 51
2.2.3 LINAGE-COUNTER. . . . 52
2.2.4 PICTURE Character-Strings. . . . 52
2.2.5 Comment-Entries. . . . 52
3. Program and Run Unit Organization and Communication. . . . 52
3.1 Program and Run Unit Organization. . . . 53
3.2 Accessing Data and Files. . . . 53
3.2.1 Names. . . . 53
3.2.2 Objects. . . . 53
3.3 Inter-program Communication. . . . 55
3.3.1 Transfer of Control. . . . 55
3.3.2 Passing Parameters to Programs. . . . 56
3.4 Intra-program Communication. . . . 57
3.4.1 Transfer of Control. . . . 57
3.4.2 Shared Data. . . . 57
C. Organization. . . . 57
D. Structure. . . . 58
E. Divisions. . . . 58
F. Reference Format (Source). . . . 59
1. General Description. . . . 59
2. ANSI Card Format. . . . 59
3. Free-Form Format (CRT). . . . 60
4. Sequence Numbers (ANSI Card Format). . . . 61
5. Continuation of Lines. . . . 61
6. Blank Lines. . . . 61
7. Comments. . . . 61
8. Debugging Lines. . . . 62
9. Division, Section, and Paragraph Formats. . . . 62
9.1 Division Header. . . . 62
9.2 Section Header. . . . 62
9.3 Paragraph Header, Paragraph-Name, and Paragraph. . . . 62
10. DATA DIVISION Entries. . . . 63
11. DECLARATIVES. . . . 63
G. COPY Statement. . . . 64

III. IDENTIFICATION DIVISION. . . . 67
A. General Description. . . . 67
B. Organization. . . . 67
C. PROGRAM-ID Paragraph. . . . 69
D. DATE-COMPILED Paragraph. . . . 69

IV. ENVIRONMENT DIVISION. . . . 71
A. General Description. . . . 71
B. Concepts. . . . 71
C. Organization. . . . 71
D. CONFIGURATION SECTION. . . . 73
1. SOURCE-COMPUTER Paragraph. . . . 73
2. OBJECT-COMPUTER Paragraph. . . . 74
3. SPECIAL-NAMES Paragraph. . . . 74
E. INPUT-OUTPUT SECTION. . . . 83
1. FILE-CONTROL Paragraph. . . . 83
2. File Control Entry. . . . 84
3. ACCESS MODE Clause. . . . 90
4. ALLOW SUB-INDEX and LEVELS Clauses (***VXCOBOL***). . . . 92
5. ALTERNATE RECORD KEY Clause (***ANSI 74*** and ***ANSI 85***). . . . 93
6. ALTERNATE RECORD KEY Clause (***VXCOBOL***). . . . 96
7. ASSIGN Clause. . . . 98
8. COMPRESSION Clauses (***VXCOBOL***). . . . 101

9. DELETE LOGICAL/PHYSICAL Clause (***ANSI 74*** and ***ANSI 85***). . . . 102
10. FILE STATUS Clause. . . . 103
11. INDEX SIZE, DATA SIZE Clauses. . . . 104
12. INFOS STATUS Clause (***VXCOBOL***). . . . 105
13. ORGANIZATION Clause. . . . 106
14. QUEUE Clause. . . . 107
15. RECORD DELIMITER Clause (***ANSI 74*** and ***ANSI 85***). . . . 108
16. RECORD KEY Clause. . . . 110
17. RESERVE Clause (***VXCOBOL***). . . . 113
18. I-O-CONTROL Paragraph. . . . 114
19. SAME Clause. . . . 115

V. DATA DIVISION. . . . 117
A. General Description. . . . 117
B. Concepts. . . . 117
1. Logical Record Concept. . . . 117
1.1 Physical Aspects of a File. . . . 117
1.2 Conceptual Characteristics of a File. . . . 117
1.3 Record Concepts. . . . 117
2. Concept of Levels. . . . 117
3. Concept of Class and Category of Data. . . . 118
4. Selection of Character Representation and Radix. . . . 119
5. Algebraic Signs. . . . 119
6. Standard Alignment Rules. . . . 119
7. Item Alignment for Increased Object-Code Efficiency. . . . 120
8. Table Handling. . . . 120
8.1 Table Definition. . . . 121
8.2 Initial Values of Tables. . . . 122
8.3 References to Table Items. . . . 122
8.4 Subscripting. . . . 123
9. Uniqueness of Reference. . . . 124
9.1 Qualification. . . . 124
9.2 Subscripting. . . . 126
9.3 Identifiers. . . . 127
9.3.1 Identifier. . . . 127
9.3.2 Function-identifier. . . . 129
9.3.3 Reference-modifier. . . . 130
9.3.4 Predefined-address. . . . 131
9.3.5 Data-address-identifier. . . . 132
9.3.6 Length-identifier. . . . 132
9.3.7 LINAGE-COUNTER. . . . 133
9.3.8 SQLSTATE (***ISQL***). . . . 133
9.4. Condition-Name. . . . 136
C. Organization. . . . 137
D. FILE SECTION. . . . 138
1. File Description Entry/Sort-Merge Description Entry. . . . 138
2. Record Description Structure. . . . 143
3. Initial Values. . . . 143
4. BLOCK CONTAINS Clause. . . . 144
5. CODE-SET Clause. . . . 145
6. DATA BLOCK and INDEX BLOCK Clauses (***VXCOBOL***). . . . 147
7. DATA RECORDS Clause. . . . 148
8. EXTERNAL Clause. . . . 149
9. FEEDBACK Clause (***VXCOBOL***). . . . 150
10. INDEX NODE Clause (***VXCOBOL***). . . . 151
11. LABEL RECORD Clause. . . . 152
12. LINAGE Clause. . . . 153
13. MERIT Clause (***VXCOBOL***). . . . 156

14. PARTIAL RECORD Clause (***VXCOBOL***). . . . 157
15. RECORD Clause (***ANSI 74*** and ***ANSI 85***). . . . 158
16. RECORDING MODE Clause (***ANSI 74*** and ***ANSI 85***). . . . 162
17. RECORDING MODE Clause (***VXCOBOL***). . . . 163
E. WORKING-STORAGE SECTION. . . . 165
1. Noncontiguous Working Storage. . . . 165
2. Working Storage Records. . . . 165
3. Record Description Structure. . . . 165
4. Initial Values. . . . 165
5. Data Description Entry. . . . 166
6. BLANK WHEN ZERO Clause. . . . 169
7. Data-Name or FILLER Clause. . . . 170
8. EXTERNAL Clause. . . . 171
9. JUSTIFIED Clause. . . . 172
10. Level-Number. . . . 173
11. OCCURS Clause. . . . 174
12. PICTURE Clause. . . . 176
13. REDEFINES Clause. . . . 183
14. RENAMES Clause. . . . 185
15. SIGN Clause. . . . 186
16. SYNCHRONIZED Clause. . . . 188
17. USAGE Clause. . . . 189
18. USAGE Clause (***ISQL***). . . . 192
19. VALUE Clause. . . . 195
F. VIRTUAL-STORAGE SECTION **(*VXCOBOL*)**. . . . 198
G. LINKAGE SECTION. . . . 199
1. Noncontiguous Linkage Storage. . . . 199
2. Linkage Records. . . . 199
3. Initial Values. . . . 199
H. SCREEN SECTION. . . . 200
1. Screen Description. . . . 200
2. Screen Description Entry. . . . 200
3. AUTO, FULL, REQUIRED Clauses. . . . 210
4. BACKGROUND-COLOR, FOREGROUND-COLOR Clauses (***ANSI 74*** and ***ANSI 85***). . . . 211
5. BELL Clause. . . . 212
6. BLANK Clause. . . . 213
7. BLINK, BOLD/BRIGHT/HIGHLIGHT/DIM/LOWLIGHT, REVERSE/REVERSED/REVERSED-VIDEO, UNDERLINE/UNDERLINED Clauses. . . . 214
8. CONVERTING Clause. . . . 216
9. ERASE Clause. . . . 217
10. FROM, TO, USING Clauses. . . . 218
11. LINE and COLUMN Clauses. . . . 219
12. OCCURS Clause. . . . 222
13. PICTURE Clause. . . . 223
14. SECURE Clause. . . . 224
15. SIGN Clause. . . . 225
16. USAGE Clause (***ISQL***). . . . 226
17 VALUE Clause. . . . 228

VI. PROCEDURE DIVISION. . . . 229
A. General Description. . . . 229
1. DECLARATIVES. . . . 229
2. Procedures. . . . 229
3. Execution. . . . 229
B. Concepts. . . . 230
1. Arithmetic Expressions. . . . 230
1.1 Definition of an Arithmetic Expression. . . . 230
1.2 Arithmetic Operators. . . . 230

1.3 Formation and Evaluation Rules. . . . 231
2. Conditional Expressions. . . . 232
2.1 Simple Conditions. . . . 233
2.2 Complex Conditions. . . . 239
2.3 Abbreviated Combined Relation Conditions. . . . 240
2.4 Order of Evaluation of Conditions. . . . 241
3. Common Options and Rules for Statements. . . . 245
3.1 ROUNDED Phrase. . . . 245
3.2 ON SIZE ERROR Phrase. . . . 246
3.3 CORRESPONDING Phrase. . . . 246
3.4 Arithmetic Statements. . . . 249
3.5 Overlapping Operands. . . . 249
3.6 Multiple Results in Arithmetic Statements. . . . 249
3.7 Incompatible Data. . . . 250
4. Statements and Sentences. . . . 250
4.1 Conditional Statements and Sentences. . . . 250
4.2 Compiler Directing Statements and Sentences. . . . 252
4.3 Imperative Statements and Sentences. . . . 252
5. Scope of Statements. . . . 253
6. Explicit and Implicit Specifications. . . . 253
6.1 Explicit and Implicit Procedure Division References. . . . 253
6.2 Explicit and Implicit Transfers of Control. . . . 253
6.3 Explicit and Implicit Attributes. . . . 254
6.4 Scope Terminators. . . . 255
6.5 Explicit Scope Terminators. . . . 255
6.6 Implicit Scope Terminators. . . . 255
C. File Concepts. . . . 256
1. File Attributes. . . . 256
1.1 Sequential Organization. . . . 256
1.2 Relative Organization. . . . 256
1.3 Indexed Organization. . . . 257
1.4 INFOS Organization (***VXCOBOL***). . . . 257
2. Logical Records. . . . 257
2.1 Fixed Length Records. . . . 257
2.2 Variable Length Records (***ANSI 74*** and ***ANSI 85***). . . . 258
2.3 Variable Length Records (***VXCOBOL***). . . . 258
3. File Processing. . . . 258
4. Record Operations. . . . 258
4.1 Sequential Access Mode. . . . 258
4.2 Random Access Mode. . . . 259
4.3 Dynamic Access Mode. . . . 259
4.4 Open Mode. . . . 259
4.5 Current Volume Pointer. . . . 260
4.6 File Position Indicator. . . . 260
5. File Operations. . . . 260
6. Exception Handling. . . . 260
6.1 I-O Status (FILE STATUS). . . . 260
6.2 I-O Status (***ANSI 74***). . . . 261
6.3 I-O Status (***ANSI 85***). . . . 264
6.4 I-O Status (***VXCOBOL***). . . . 267
6.5 INFOS Status (***VXCOBOL***). . . . 271
6.6 The At End Condition. . . . 271
6.7 The Invalid Key Condition. . . . 271
6.8 The File Attribute Conflict Condition. . . . 272
6.9 Exception Declaratives. . . . 272
6.10 Optional Phrases. . . . 272
6.11 ACCEPT FROM EXCEPTION STATUS. . . . 273
7. Shared Record Area. . . . 273

8. INFOS File I-O Common Phrases (***VXCOBOL***). . . . 273
8.1 The POSITION Phrase. . . . 273
8.2 The Relative Motion Phrase. . . . 274
8.3 The KEY Series Phrase. . . . 275
8.4 The SUPPRESS Phrase. . . . 275
8.5 The LOCK/UNLOCK Phrase. . . . 276
D. Header. . . . 277
E. Statements. . . . 279
1. ACCEPT (keyboard). . . . 279
2. ACCEPT (system). . . . 290
3. ADD. . . . 297
4. CALL. . . . 299
5. CALL PROGRAM. . . . 303
6. CANCEL. . . . 307
7. CLOSE. . . . 309
8. COMMIT (***ISQL***). . . . 311
9. COMPUTE. . . . 313
10. CONNECT (***ISQL***). . . . 315
11. CONTINUE. . . . 318
12. DEALLOCATE (***ISQL***). . . . 319
13. DEFINE SUB-INDEX (***VXCOBOL***). . . . 321
14. DELETE. . . . 325
15. DELETE FILE. . . . 331
16. DISCONNECT (***ISQL***). . . . 333
17. DISPLAY. . . . 335
18. DIVIDE. . . . 343
19. EVALUATE (***ANSI 74*** and ***ANSI 85***). . . . 347
20. EXECUTE (***ISQL***). . . . 351
21. EXECUTE IMMEDIATE (***ISQL***). . . . 353
22. EXIT. . . . 355
23. EXIT PROGRAM. . . . 357
24. EXPUNGE (***VXCOBOL***). . . . 359
25. EXPUNGE SUB-INDEX (***VXCOBOL***). . . . 361
26. FETCH (***ISQL***). . . . 363
27. GET DIAGNOSTICS (***ISQL***). . . . 365
28. GO TO. . . . 369
29. GOBACK. . . . 371
30. IF. . . . 373
31. INITIALIZE (***ANSI 74*** and ***ANSI 85***). . . . 375
32. INSPECT. . . . 379
33. LINK SUB-INDEX (***VXCOBOL***). . . . 387
34. MERGE. . . . 389
35. MOVE. . . . 393
36. MULTIPLY. . . . 397
37. OPEN. . . . 399
38. PERFORM. . . . 404
39. PREPARE (***ISQL***). . . . 412
40. READ (***ANSI 74*** and ***ANSI 85***). . . . 414
41. READ (***VXCOBOL***). . . . 420
42. RELEASE. . . . 427
43. RETRIEVE (***VXCOBOL***). . . . 429
44. RETURN. . . . 431
45. REWRITE. . . . 433
46. ROLLBACK (***ISQL***). . . . 437
47. SEARCH. . . . 439
48. SET (***ANSI 74*** and ***ANSI 85***). . . . 443
49. SET (***VXCOBOL***). . . . 447
51. SORT. . . . 451

52. START. . . . 457
53. STOP. . . . 463
54. STRING. . . . 465
55. SUBTRACT. . . . 467
56. UNDELETE (***ANSI 74*** and ***ANSI 85***). . . . 469
57. UNDELETE (***VXCOBOL***). . . . 471
58. UNLOCK. . . . 473
59. UNSTRING. . . . 475
60. USE. . . . 479
61. WRITE. . . . 483

VII. BUILTINS. . . . 493
A. Introduction. . . . 493
1. Overview. . . . 493
B. Builtins. . . . 494
1. ?CBADDR. . . . 494
2. ?CBBADDR. . . . 495
3. ?CBSYS. . . . 496
4. CLI. . . . 497
5. IC_ABORT_TERM. . . . 498
6. IC_CHANGE_DIR. . . . 499
7. IC_CHANGE_PRIV. . . . 500
8. IC_CHECK_DATA. . . . 502
9. IC_COMPRESS_OFF. . . . 503
10. IC_COMPRESS_ON. . . . 504
11. IC_CREATE_DIR. . . . 505
12. IC_CURRENT_DIR. . . . 506
13. IC_DECODE_URL. . . . 507
14. IC_DELAY. . . . 508
15. IC_DETACH_PROGRAM. . . . 509
16. IC_DIR_LIST. . . . 511
17. IC_DISABLE_HOTKEY. . . . 512
18. IC_DISABLE_INTS. . . . 513
19. IC_ENABLE_HOTKEY. . . . 514
20. IC_ENABLE_INTS. . . . 515
21. IC_ENCODE_URL. . . . 516
22. IC_EXTRACT_STRING. . . . 517
23. IC_FULL_DATE. . . . 518
24. IC_GET_DISK_SPACE. . . . 519
25. IC_GET_ENV. . . . 520
26. IC_GET_FILE_IND. . . . 521
27. IC_GET_KEY. . . . 522
28. IC_HANGUP. . . . 524
29. IC_HEX_TO_NUM. . . . 525
30. IC_INFOS_STATUS_TEXT (***VXCOBOL***). . . . 526
31. IC_INSERT_STRING. . . . 527
32. IC_KILL_TERM. . . . 528
33. IC_LOGON. . . . 529
34. IC_LOWER. . . . 530
35. IC_MOVE_FILE_DATA. . . . 531
36. IC_MOVE_STRING. . . . 532
37. IC_MSG_TEXT. . . . 533
38. IC_NUM_TO_HEX. . . . 534
39. IC_PID_EXISTS. . . . 535
40. IC_PRINT_STAT. . . . 536
41. IC_QUEUE_LIST. . . . 541
42. IC_QUEUE_OPERATION. . . . 545
43. IC_QUEUE_STATUS. . . . 549
44. IC_REMOVE_DIR. . . . 550

45. IC_RENAME. . . . 551
46. IC_RESOLVE_FILE. . . . 552
47. IC_SEND_MAIL. . . . 554
48. IC_SEND_MSG. . . . 556
49. IC_SERIAL_NUMBER. . . . 557
50. IC_SET_ENV. . . . 558
51. IC_SET_TIMEOUT. . . . 559
52. IC_SET_USERNAME. . . . 560
53. IC_SHUTDOWN. . . . 561
54. IC_SYS_INFO. . . . 562
55. IC_TERM_CTRL. . . . 564
56. IC_TERM_STAT. . . . 565
57. IC_TRIM. . . . 567
58. IC_UPPER. . . . 568
59. IC_VERSION. . . . 569
60. IC_WINDOW_TITLE. . . . 570
61. IC_WINDOWS_MSG_BOX. . . . 572
62. IC_WINDOWS_SETFONT. . . . 575
63. IC_WINDOWS_SHELLEXECUTE (Windows only). . . . 576
64. IC_WINDOWS_SHOW_CONSOLE. . . . 577

VIII. INTRINSIC FUNCTIONS. . . . 579
A. General Description. . . . 579
1. Types of Functions. . . . 579
2. Arguments. . . . 579
3. Returned values. . . . 580
4. Date conversion functions. . . . 580
5. Summary of functions. . . . 581
B. Intrinsic Functions. . . . 584
1. ABS. . . . 584
2. ACOS. . . . 585
3. ANNUITY. . . . 586
4. ASIN. . . . 588
5. ATAN. . . . 589
6. BYTE-LENGTH. . . . 590
7. CHAR. . . . 592
8. COS. . . . 593
9. CURRENT-DATE. . . . 594
10. DATE-OF-INTEGER. . . . 596
11. DATE-TO-YYYYMMDD. . . . 597
12. DAY-OF-INTEGER. . . . 599
13. DAY-TO-YYYYDDD. . . . 600
14. E. . . . 602
15. EXP. . . . 603
16. EXP10. . . . 604
17. FACTORIAL. . . . 605
18. FRACTION-PART. . . . 606
19. HIGHEST-ALGEBRAIC. . . . 607
20. INTEGER. . . . 608
21. INTEGER-OF-DATE. . . . 609
22. INTEGER-OF-DAY. . . . 610
23. INTEGER-PART. . . . 611
24. LENGTH. . . . 612
25. LOG. . . . 614
26. LOG10. . . . 615
27. LOWER-CASE. . . . 616
28. LOWEST-ALGEBRAIC. . . . 617
29. MAX. . . . 618
30. MEAN. . . . 620

31. MEDIAN. . . . 622
32. MIDRANGE. . . . 624
33. MIN. . . . 626
34. MOD. . . . 628
35. NUMVAL. . . . 630
36. NUMVAL-C. . . . 632
37. NUMVAL-F. . . . 634
38. ORD. . . . 635
39. ORD-MAX. . . . 636
40. ORD-MIN. . . . 637
41. PI. . . . 638
42. PRESENT-VALUE. . . . 639
43. RANDOM. . . . 641
44. RANGE. . . . 642
45. REM. . . . 643
46. REVERSE. . . . 644
47. SIGN. . . . 645
48. SIN. . . . 646
49. SQRT. . . . 647
50. STANDARD-DEVIATION. . . . 648
51. SUM. . . . 649
52. TAN. . . . 650
53. TEST-DATE-YYYYMMDD. . . . 651
54. TEST-DAY-YYYYDDD. . . . 653
55. TEST-NUMVAL. . . . 655
56. TEST-NUMVAL-C. . . . 657
57. TEST-NUMVAL-F. . . . 659
58. UPPER-CASE. . . . 661
59. VARIANCE. . . . 662
60. WHEN-COMPILED. . . . 664
61. YEAR-TO-YYYY. . . . 666

IX. SCREEN HANDLER. . . . 669
A. General Description. . . . 669
1. Enabling the SCREEN HANDLER. . . . 669
2. Summary of Calls. . . . 670
3. Error Handling. . . . 671
B. Calls. . . . 672
1. SD_DRAW_BOX. . . . 672
2. SD_DRAW_HLINE and SD_DRAW_VLINE. . . . 673
3. SD_GET_IMAGE. . . . 674
4. SD_GET_POS. . . . 675
5. SD_MESSAGE, SD_ERROR_MESSAGE, SD_MESSAGE_ONLY. . . . 676
6. SD_NEW_WINDOW. . . . 677
7. SD_POP_UP_MENU. . . . 678
8. SD_POP_UP_MENU2. . . . 679
9. SD_READ_CHAR. . . . 680
10. SD_REDRAW. . . . 682
11. SD_REMOVE_WINDOW. . . . 683
12. SD_RETURN_INPUT. . . . 684
13. SD_SET_ACCEPT_TIMEOUT. . . . 685
14. SD_SYS_ERROR_MESSAGE. . . . 686

PART TWO - DEVELOPER'S GUIDE. . . . 687

X. INTRODUCTION TO THE DEVELOPER'S GUIDE. . . . 689
A. Overview. . . . 689

B. Operating Environment. . . . 689
1. General Concepts. . . . 689
1.1 Communication with the Operating System. . . . 689
1.2 I-O Redirection. . . . 689
1.3 Environment Variables. . . . 689
2. Directory Structure. . . . 690
3. ICEXEC Control Program. . . . 691
4. ICPERMIT License Program. . . . 691
C. Command-line Conventions. . . . 692
1. Switches. . . . 692
2. Conventions for Defining Syntax. . . . 692
3. Filename Case (upper or lower). . . . 692
D. Common Switches. . . . 692
1. Overall. . . . 692
2. Audit Switch. . . . 693
3. Quiet Switch. . . . 693
4. Help Switch. . . . 694
E. Filename Extensions. . . . 694
F. Exit Codes. . . . 696
G. Common Environment Variables. . . . 696
1. Overall. . . . 696
2. ICROOT. . . . 696
3. ICTMPDIR. . . . 697
4. Executable-Name Environment Variable. . . . 697
5. TZ (Windows only). . . . 697

XI. COMPILER (ICOBOL). . . . 699
A. Overview. . . . 699
B. Syntax. . . . 699
1. Rules. . . . 701
2. Environment Variables. . . . 702
C. Switches. . . . 702
1. Overview. . . . 702
2. Byte Alignment Switch (-B 1|2|4). . . . 703
3. COPY Sourcedir Switch (-c). . . . 703
4. COPY Path Switch (-C *copydir*). . . . 703
5. Dialect Switch (-D ic|vx|85). . . . 703
6. Error File Switch (-e | -E *erdir*). . . . 703
7. Format Switch (-F c|f). . . . 704
8. General Switch (-G {6|a|b|d|e|g|h|i|k|n|p|q|s}...). . . . 704
9. Hard Error Limit Switch (-H *cnt*). . . . 705
10. Information Switch (-i). . . . 705
11. Include listing options Switch (-I {g|m|p|x}...). . . . 705
12. Listing File Switch (-l | -L *lsdir*). . . . 705
13. Make ICODBC Data Definition Files Switch (-M *dddir*). . . . 706
14. No Switch (-N {h|p|s|u}...). . . . 706
15. OEM Version Switch (-o | -O *rev*). . . . 706
16. Program Output File Switch (-P *cxdir*). . . . 707
17. Revision Switch (-R 1|2|3|4). . . . 707
18. Statistics Switch (-s). . . . 707
19. Warnings Switch (-w). . . . 707
20. ICODBC Options Switch (-X "*string*"). . . . 707
21. Debug Switch (-Z *sydir*). . . . 709
D. Messages. . . . 709
1. Overview. . . . 709
1.1 Format. . . . 709
1.2 Examples. . . . 711
2. Error Messages. . . . 711

3. Warning Messages.. 712
4. Information Messages. 713
E. Example Output.. 715
F. Cross Reference Output. 716
G. ICODBC Support.. 717

XII. DEBUGGING. 721
A. **Introduction**.. 721
B. **Invocation**. 721
C. **Usage**. 722
D. **Commands**.. 726
1. Overview.. 726
2. AUDIT.. 727
3. BREAK. 727
4. COMMAND.. 730
5. DUMP.. 731
6. ERROR RESET. 731
7. EXECUTE.. 732
8. FIND.. 732
9. GO. 732
10. HELP.. 733
11. INFO.. 733
12. LIST. 734
13. MOVE.. 735
14. QUIT.. 735
15. RERUN.. 735
16. RUN. 736
17. STEP.. 736
18. TYPE.. 737
19. VIEW.. 737
20. ZOOM.. 738
E. **Performance Considerations**.. 738
F. **Quick Reference**. 739

XIII. ICREVSET. 741
A. Introduction. 741
B. Syntax. 741
C. General Rules. 741

XIV. ICDUMP. 743
A. Introduction. 743
B. Syntax. 743
C. Rules. 743
D. Example. 743

XV. RUNTIME (ICRUN).. 747
A. Introduction. 747
B. Printer Control Utility. 747
C. Program Termination.. 748
1. Two Types of Termination.. 748
1.1 Return to LOGON as Inactive.. 748
1.2 Return to Parent Process. 748
D. Device Support. 748
1. Overview.. 748
2. General Rules. 749
3. Parallel Printer Ports 750
4. Serial Ports.. 750
E. Filenaming Conventions. 751

1. Internal Filenames. . . . 751
2. External Filenames. . . . 751
2.1 Rules. . . . 752
2.2 Program names. . . . 753
2.3 Sequential and ICISAM Filenames. . . . 755
F. Extended OPEN options. . . . 756
1. Overview. . . . 756
2. Extended Sequential Open. . . . 757
3. Extended Relative Open (***ANSI 74*** and ***ANSI 85***). . . . 759
4. Extended Indexed Open. . . . 760
G. ICISAM Information. . . . 761
1. Overview. . . . 761
2. ICISAM Versions. . . . 761
3. ICISAM Reliability. . . . 762
4. ICISAM Key Ordering. . . . 762
H. Notes and Warnings. . . . 763
I. UNIX Pipe Opens. . . . 764
J. BTRIEVE Support. . . . 765
1. Overview. . . . 765
2. Runtime. . . . 765
3. ICREORG. . . . 766
4. ICNETD. . . . 766
K. C-ISAM Support. . . . 767
1. Overview. . . . 767
2. Runtime. . . . 767
3. ICREORG. . . . 767
4. ICNETD. . . . 767
L. HOT KEYS. . . . 767
1. Introduction. . . . 767
2. Construction. . . . 768
3. Restrictions. . . . 768
3. Example. . . . 769

XVI. ICODBC Driver. . . . 771
A. Introduction. . . . 771
B. General Information. . . . 771
C. Using the Driver. . . . 771
D. Creating .XDB and XDT Files. . . . 772
E. Managing Data Sources (On Windows). . . . 779
F. Managing Data Sources (On UNIX). . . . 780
G. Data Types Supported. . . . 784
H. Driver Limitations. . . . 786
I. SQL Grammar Supported. . . . 786
J. Usage Notes. . . . 789
K. Debugging. . . . 791
L. SYWARE. . . . 791

XVII. ICIDE. . . . 793
A. Introduction. . . . 793
B. Use. . . . 793
C. Changing .CO or .SR file associations. . . . 794

XVIII. GLOSSARY. . . . 797
A. Introduction. . . . 797
B. Definitions. . . . 797

APPENDICES. . . . 815
A. IMPLEMENTATION LIMITS. . . . 817

B. ESCAPE KEY TABLE. 819
C. ANSI 74 FILE STATUS CODES. 821
D. ANSI 85 FILE STATUS CODES. 823
E. *VXCOBOL* FILE STATUS CODES. 825
F. ANSI 74 and ANSI 85 EXCEPTION STATUS CODES. 827
G. *VXCOBOL* EXCEPTION STATUS CODES. 835
H. UNIX Errno. 843
I. ASCII CODES. 844
J. EBCDIC CODES. 846
K. COBOL RESERVED WORDS. 847
L. SYSTEM CALLS. 851

INDEX. 865

Support Information Request (SIR). 879

Enhancement/Suggestion Request. 881

CUSTOMER DOCUMENTATION COMMENT FORM. 883

LIST OF EXAMPLES

EXAMPLE 1. Identifying parameters passed by a calling program. . . . 56
EXAMPLE 2. Using a Program Switch. . . . 80
EXAMPLE 3. Modifying the collating sequence for a program. . . . 80
EXAMPLE 4. Changing 1 character in the collating sequence 80
EXAMPLE 5. Making multiple characters the same in the collating sequence. . . . 81
EXAMPLE 6. Reversing collating sequence for digits, uppercase alphabet. . . . 81
EXAMPLE 7. Definition for a one-dimensional table. . . . 121
EXAMPLE 8. Another one-dimensional table. . . . 121
EXAMPLE 9. Three one-dimensional tables without group names. . . . 121
EXAMPLE 10. Definition for a two-dimensional table. . . . 122
EXAMPLE 11. Referencing single- and multi-dimensional table elements. . . . 122
EXAMPLE 12. Referencing elements in 1-, 2-, and 3-dimensional tables. . . . 124
EXAMPLE 13. Referencing an intrinsic function with and without arguments. . . . 129
EXAMPLE 14. Abbreviated combined and negated combined relation conditions. . . . 241
EXAMPLE 15. MOVE CORRESPONDING and ADD CORRESPONDING. . . . 248
EXAMPLE 16. MOVE CORRESPONDING. . . . 249
EXAMPLE 17. CALL the Bourne shell from a COBOL program (UNIX). . . . 302
EXAMPLE 18. CALL the shell, have it execute "ls" and return (UNIX). . . . 302
EXAMPLE 19. CALL the "ls" command directly and return (UNIX). . . . 302
EXAMPLE 20. CALL the command processor (Windows). . . . 302
EXAMPLE 21. CALL the command processor and execute the DIR command (Windows). . . . 302
EXAMPLE 22. EVALUATE. . . . 350
EXAMPLE 23. INSPECT TALLYING, REPLACING. . . . 384
EXAMPLE 24. INSPECT TALLYING, REPLACING. . . . 385
EXAMPLE 25. INSPECT TALLYING, REPLACING. . . . 385
EXAMPLE 26. INSPECT TALLYING, REPLACING. . . . 386
EXAMPLE 27. INSPECT CONVERTING. . . . 386
EXAMPLE 28. Using Declaratives. . . . 481
EXAMPLE 29. ABS function. . . . 584
EXAMPLE 30. ACOS function. . . . 585
EXAMPLE 31. ANNUITY function. . . . 587
EXAMPLE 32. ASIN function. . . . 588
EXAMPLE 33. ATAN function. . . . 589
EXAMPLE 34. BYTE-LENGTH function. . . . 591
EXAMPLE 35. CHAR function. . . . 592
EXAMPLE 36. COS function. . . . 593
EXAMPLE 37. CURRENT-DATE function. . . . 595
EXAMPLE 38. DATE-OF-INTEGER function. . . . 596
EXAMPLE 39. DATE-TO-YYYYMMDD function. . . . 598
EXAMPLE 40. DAY-OF-INTEGER function. . . . 599
EXAMPLE 41. DAY-TO-YYYYDDD function. . . . 601
EXAMPLE 42. E function. . . . 602
EXAMPLE 43. EXP function. . . . 603
EXAMPLE 44. EXP10 function. . . . 604
EXAMPLE 45. FACTORIAL function. . . . 605
EXAMPLE 46. FRACTION-PART function. . . . 606
EXAMPLE 47. HIGHEST-ALGEBRAIC function. . . . 607
EXAMPLE 48. INTEGER function. . . . 608
EXAMPLE 49. INTEGER-OF-DATE function. . . . 609
EXAMPLE 50. INTEGER-OF-DAY function. . . . 610
EXAMPLE 51. INTEGER-PART function. . . . 611
EXAMPLE 52. LENGTH function. . . . 613
EXAMPLE 53. LOG function. . . . 614
EXAMPLE 54. LOG10 function. . . . 615
EXAMPLE 55. LOWER-CASE function. . . . 616

EXAMPLE 56. LOWEST-ALGEBRAIC function. . . . 617
EXAMPLE 57. MAX function.. . . . 619
EXAMPLE 58. MEAN function. . . . 621
EXAMPLE 59. MEDIAN function. . . . 623
EXAMPLE 60. MIDRANGE function. . . . 625
EXAMPLE 61. MIN function. . . . 627
EXAMPLE 62. MOD function.. . . . 629
EXAMPLE 63. NUMVAL function. . . . 631
EXAMPLE 64. NUMVAL-C function. . . . 633
EXAMPLE 65. NUMVAL-F function. . . . 634
EXAMPLE 66. ORD function. . . . 635
EXAMPLE 67. ORD-MAX function. . . . 636
EXAMPLE 68. ORD-MIN function. . . . 637
EXAMPLE 69. PI function. . . . 638
EXAMPLE 70. PRESENT-VALUE function. . . . 640
EXAMPLE 71. RANDOM function. . . . 641
EXAMPLE 72. RANGE function. . . . 642
EXAMPLE 73. REM function. . . . 643
EXAMPLE 74. REVERSE function. . . . 644
EXAMPLE 75. SIGN function.. . . . 645
EXAMPLE 76. SIN function. . . . 646
EXAMPLE 77. SQRT function. . . . 647
EXAMPLE 78. STANDARD-DEVIATION function. . . . 648
EXAMPLE 79. SUM function. . . . 649
EXAMPLE 80. TAN function. . . . 650
EXAMPLE 81. TEST-DATE-YYYYMMDD function. . . . 652
EXAMPLE 82. TEST-DAY-YYYYDDD function. . . . 654
EXAMPLE 83. TEST-NUMVAL function.. . . . 656
EXAMPLE 84. TEST-NUMVAL-C function.. . . . 658
EXAMPLE 85. TEST-NUMVAL-F function. . . . 660
EXAMPLE 86. UPPER-CASE function. . . . 661
EXAMPLE 87. VARIANCE function 663
EXAMPLE 88. WHEN-COMPILED function. . . . 665
EXAMPLE 89. YEAR-TO-YYYY function. . . . 667
EXAMPLE 90. ICDUMP of the Header (default). . . . 744
EXAMPLE 91. ICDUMP of the Program Code (using the -c switch). . . . 744
EXAMPLE 92. ICDUMP of the Reference Table (using the -r switch).. . . . 745
EXAMPLE 93. ICDUMP of the Data (using the -d switch). . . . 745

LIST OF FIGURES

FIGURE 1. Evaluation of *condition-1* AND *condition-2* AND ... *condition-n*.. . . . 242
FIGURE 2. Evaluation of *condition-1* OR *condition-2* OR ... *condition-n*. . . . 243
FIGURE 3. Evaluation of *condition-1* OR *condition-2* AND *condition-3*.. . . . 244
FIGURE 4. Evaluation of (*condition-1* OR NOT *condition-2*) AND *condition-3* AND *condition-4*. . . . 245
FIGURE 5. PERFORM [TEST BEFORE] VARYING with one condition. . . . 409
FIGURE 6. PERFORM [TEST BEFORE] VARYING with two conditions. . . . 410
FIGURE 7. Valid PERFORM constructs. . . . 411
FIGURE 8. Format 1 SEARCH statement having two WHEN phrases. . . . 442
FIGURE 9. **ICOBOL** Directory Structure (UNIX). . . . 690
FIGURE 10. **ICOBOL** Directory Structure (Windows). . . . 691

LIST OF SCREENS

SCREEN 1. Default Debugging SCREEN. . . . 722
SCREEN 2. Debugging SCREEN (all views enabled). . . . 722
SCREEN 3. Debugging SCREEN (no symbol file). . . . 723
SCREEN 4. Debugging SCREEN (symbols but no source). . . . 723

LIST OF TABLES

TABLE 1. Default External Filenames for Sequential Files. . . . 100
TABLE 2. Relationship of the Class and Categories of Data Items. . . . 118
TABLE 3. File Description Clauses by **ICOBOL** dialect and file type. . . . 143
TABLE 4. PICTURE Editing. . . . 179
TABLE 5. Sign Control in Fixed PICTURE Editing. . . . 180
TABLE 6. Sign Control in Floating PICTURE Editing. . . . 181
TABLE 7. PICTURE Precedence Rules. . . . 182
TABLE 8. SIGN Overpunch Characters. . . . 187
TABLE 9. BINARY & COMPUTATIONAL Storage Allocation. . . . 190
TABLE 10. COMPUTATIONAL-5 Storage Allocation. . . . 191
TABLE 11. INTERVAL Field Maximum Precision (***ISQL***). . . . 194
TABLE 12. BACKGROUND-COLOR and FOREGROUND-COLOR. . . . 211
TABLE 13. LINE and COLUMN relationship. . . . 221
TABLE 14. INTERVAL Field Maximum Precision (***ISQL***). . . . 227
TABLE 15. Combination of Symbols in Arithmetic Expressions. . . . 231
TABLE 16. Relational Operators. . . . 234
TABLE 17. Combinations of Conditions, Logical Operators, and Parentheses. . . . 240
TABLE 18. Variable Origin for DISPLAY and ACCEPT. . . . 284
TABLE 19. Function Key Escape Codes. . . . 294
TABLE 20. Common Error Conditions for a CALL Statement. . . . 301
TABLE 21. Common Error Conditions for a CALL PROGRAM Statement. . . . 304
TABLE 22. How Program Switches are evaluated. . . . 305
TABLE 23. CALL and CALL PROGRAM Compared. . . . 306
TABLE 24. Combination of operands in the EVALUATE statement. . . . 348
TABLE 25. Legality of Types of MOVE Statements. . . . 395
TABLE 26. Availability of a File (***ANSI 74***). . . . 400
TABLE 27. Availability of a File (***ANSI 85***). . . . 400
TABLE 28. Availability of a File (***VXCOBOL***). . . . 401
TABLE 29. Permissible Statements. . . . 401
TABLE 30. Validity of Operand Combinations in Format 1 SET Statements. . . . 445
TABLE 31. ***ANSI 74*** and ***ANSI 85*** ADVANCING Definitions. . . . 487
TABLE 32. ***VXCOBOL*** ADVANCING Definitions. . . . 488
TABLE 33. ***VXCOBOL*** CHANNEL ADVANCING Definitions. . . . 488
TABLE 34. List of BUILTINS. . . . 493
TABLE 35. IC_GET_KEY values returned. . . . 523
TABLE 36. Summary of Intrinsic Functions. . . . 583
TABLE 37. Summary of Screen Handler Calls. . . . 670
TABLE 38. Common Command-line Syntax Conventions. . . . 692
TABLE 39. Common Filename Extensions used by **ICOBOL**. . . . 695
TABLE 40. Cross Reference Symbol Types. . . . 717
TABLE 41. **ICOBOL** Data Types to ODBC Data Types. . . . 719
TABLE 42. Device Mappings. . . . 749
TABLE 43. Legal characters in a filename. . . . 751
TABLE 44. Illegal Characters in a Filename. . . . 752
TABLE 45. Characters Allowed in a Filename, in Certain Contexts. . . . 752
TABLE 46. Three Categories of Extended Open for Sequential Files. . . . 757
TABLE 47. BTRIEVE-specific Exception Status Codes. . . . 766

TABLE 48. ICODBC Data Types to ODBC SQL Data Types. 785

PREFACE

Scope

This manual defines the COBOL language supported by Interactive COBOL. This COBOL language is based on the ANSI COBOL standard X3.23-1985. The manual is intended for programmers already familiar with the COBOL language in general.

The complete documentation for Interactive COBOL includes the following manuals:

Installing and Configuring Interactive COBOL on UNIX (011-00402)
Installing and Configuring Interactive COBOL on Windows (011-00403)
Each manual provides the appropriate sections necessary to properly install and configure Interactive COBOL in the given environment.

Interactive COBOL Utilities Manual (011-00300)
Provides a simple guide to all the user visible utilities.

Interactive COBOL Language Reference & Developer's Guide (011-00100)
Contains two parts:

A) Interactive COBOL Language Reference: The complete COBOL syntax supported by all dialects of **ICOBOL.** Included are **ICOBOL** builtins, intrinsic functions, and screen calls.

B) Interactive COBOL Developer's Guide: Explains how to use the development tools including the compiler, debugger, ICREVSET, and ICDUMP. It also explains how the **ICOBOL** runtime works including how to program across the multiple environments supported by **ICOBOL**.

COBOL sp2 User Interface Development Manual
How to use the ICSP2 Panel Editor to define GUI screens.

COBOL FormPrint
How to use the ICQPRW FormPrint Editor to setup printers.

TERMS

This document uses several terms as generic names to describe the following products.

ANSI 74, ANSI 85, and ***VXCOBOL*** are the three dialects supported by Interactive COBOL and are used to describe differences.

AOS/VS refers to both AOS/VS II and AOS/VS (Classic) unless specifically stated..

ICOBOL refers to all dialects of the Interactive COBOL product unless otherwise stated.

INFOS refers to either AOS/VS INFOS II or U/FOS. INFOS II or U/FOS are explicitly used when needed.

VXCOBOL refers to all models of the ***VXCOBOL*** products unless otherwise stated.

UNIX refers to all supported dialects of UNIX (AIX™, HP-UX, etc.) including Linux unless specifically stated.

Windows refers to all supported Microsoft Windows products (Windows 2000, Windows XP, Windows Server 2003, Windows Vista, etc.) unless otherwise stated.

ENHANCEMENTS (Language area)

Interactive COBOL 3.60 added support for the following:

- Enhanced builtins: IC_WINDOWS_MSG_BOX, IC_WINDOWS_SHOW_CONSOLE, IC_WINDOW_TITLE

Interactive COBOL 3.57 added support for the following:

- Enhanced builtin: IC_SEND_MAIL

Interactive COBOL 3.56 added support for the following:

- Enhanced builtin: IC_WINDOWS_SETFONT

- Filenames can now contain "(" and ")"

Interactive COBOL 3.50 added support for the following:

- New builtin: IC_SEND_MAIL

Interactive COBOL 3.40 added support for the following:

- Integrated SQL (***ISQL***) added that provides a simple way of using popular relational databases directly from within your COBOL programs. ***ISQL*** provides many of the embedded SQL features but in an integrated fashion without the added complexity of pre-processors or call-level interface. Most of the SQL data types have been added to the base language set. At runtime, ***ISQL*** makes use of standard ODBC calls to access any data manager available to ODBC.

  New literal types include: DATE, TIME, TIMESTAMP, and INTERVALS.

  New data types include: CHARACTER, CHARACTER VARYING, DATE, INDICATOR, INTEGER, INTERVAL, NUMERIC, SMALLINT, TIME, and TIMESTAMP.

  New statements include: COMMIT, CONNECT, DEALLOCATE, DISCONNECT, EXECUTE, EXECUTE IMMEDIATE, FETCH, GET DIAGNOSTICS, PREPARE, ROLLBACK, and SET CONNECTION. ***(These statements require an additional ICSQL runtime license)***.

  New identifier: SQLSTATE

  Enhancements to other statements to support the new literal and data types.

  These features are made available with the new General switch (-G q) on the compiler.

  Debugger support for the above.

- Special Register LENGTH OF

- **VXCOBOL** dialect will allow CONTINUE, GOBACK, reference modification, and intrinsic functions

- Use of reference modification in the SCREEN SECTION

Interactive COBOL 3.35 added support for the following:

- Enhanced builtin: IC_SYS_INFO

Interactive COBOL 3.34 added support for the following:

- Enhanced builtin: IC_WINDOWS_SETFONT

- New builtin: IC_TRIM.

Interactive COBOL 3.30 added support for the following:

- ACCEPT FROM ENVIRONMENT updated to give the minimum and maximum screen column sizes and the computer name.

- New builtins: IC_COMPRESS_ON and IC_COMPRESS_OFF to support compress mode on screens.

- GOBACK statement added

- Inline comment (*>) added

Interactive COBOL 3.22 added support for the following:

- Enhanced builtin: IC_SYS_INFO to accept an optional argument.

- New builtin: IC_GET_FILE_IND returns header information about a particular ICISAM indexed file.

Interactive COBOL 3.20 added support for the following:

- Enhancements to the Screen Section, including OCCURS, LINE PLUS/MINUS variable, relative positioning after absolute positioning, identifier for FOREGROUND-COLOR and BACKGROUND-COLOR, CONVERTING UP/DOWN, and compatibility enhancements for the ERASE, BLANK, attribute control clauses.

- Introduction of screen control clauses such as line and column positioning and attribute control, etc. for non-screen ACCEPT and DISPLAY statements

Interactive COBOL 3.13 added support for the following:

- New builtins: IC_SET_ENV and IC_WINDOWS_SETFONT.

- New compiler switch (-c).

- ICIDE enhancements.

- Runtime support to write to the audit file. (DISPLAY UPON)

Interactive COBOL 3.12 added support for the following:

- Enhanced builtin: IC_SEND_MSG.

Interactive COBOL 3.11 added support for the following:

- New builtin: IC_WINDOWS_SHELLEXECUTE.

- On Windows, ICRUNW can now set its font and size at startup.

Interactive COBOL 3.10 added support for the following:

- New reserved words for the **ANSI 74** and **ANSI 85** dialects: CONVERT, CURSOR, HIGH, LOW, PROMPT, and TAB.

- Removed the debugger (ICDEB) as a separate executable and made an integral part of the runtime.

- On Windows, added an integrated development environment (ICIDE) allowing projects to be defined, edited, and compiled in one place.

Interactive COBOL 3.03 added support for the following:

- New reserved words for the **ANSI 74** and **ANSI 85** dialects: BACKGROUND, BEEP, FOREGROUND, and MINUS.

Interactive COBOL 3.01 added support for the following:

- New builtins: IC_QUEUE_LIST and IC_QUEUE_OPERATION.

Interactive COBOL 3.00 was a major release which added support for the following:

- Code and data space increased to 16MB each
- Multicharacter switches
- Nested COPY files
- Expressions in subscripts
- Reference modification (**ANSI 74/85** only)
- ACCEPT FROM DATE YYYYMMDD
- ACCEPT FROM DAY YYYYDDD
- ACCEPT FROM EXCEPTION STATUS WITH ERROR IN xx
- CALL by CONTENT
- CODE-SET
- COPY REPLACING
- OCCURS DEPENDING ON
- EVALUATE statement (**ANSI 74/85** only)
- EXTERNAL data and files
- INITIALIZE statement (**ANSI 74/85** only)
- Enhanced INSPECT (multiple TALLYING, CONVERTING clause)
- LINAGE support
- SECURE NO ECHO
- QUEUE IS added to SELECT
- RECORD DELIMITER added to SELECT
- START is available for sequential files
- STOP RUN literal
- Varying length records for all file types
- IS INITIAL PROGRAM
- 61 INTRINSIC FUNCTIONS added (ABS, ACOS, ANNUITY, ...) (**ANSI 74/85** only)

- New Builtins: IC_HANGUP, IC_LOGON, IC_QUEUE_STATUS, IC_SHUTDOWN, IC_INFOS_STATUS_TEXT, IC_PID_EXISTS, IC_HEX_TO_NUM, IC_NUM_TO_HEX, CLI, ?CBSYS, ?CBADDR, and ?CBBADDR
- Supports a ***VXCOBOL*** dialect (Data General AOS/VS COBOL compatible)

PART ONE - LANGUAGE REFERENCE

I. CONVENTIONS USED IN THIS MANUAL

A. Definition of a General Format

A *general format* is the specific arrangement of the elements of a clause or a statement.

A *clause* or a *statement* consists of elements as defined below. Throughout this document a format is shown adjacent to information defining the clause or statement. When more than one specific arrangement is permitted, the general format is separated into numbered or named formats. Clauses must be written in the sequence given in the general formats. (If they are used, optional clauses must appear in the sequence shown.) In certain cases, stated explicitly in the rules associated with a given format, clauses may appear in sequences other than that shown. Applications, requirements, or restrictions concerning a format, are shown as *rules*.

A.1. Elements

Elements that make up a clause or a statement consist of uppercase words, lowercase words, level-numbers, brackets, braces, connectives, and special characters.

A.2. Words

UNDERLINED UPPERCASE WORDS represent *keywords* and are required whenever the functions of which they are a part are used. An error will be reported by the compiler if a *keyword* is absent or incorrectly spelled.

UPPERCASE WORDS that are not underlined are optional; they are used only for readability.

Lowercase words, in a general format, are generic terms used to represent COBOL words, literals, PICTURE character-strings, comment-entries, or a complete syntactical entry that must be supplied by the user. Where generic terms are repeated in a general format, a number or letter appended to the term serves to identify that term for explanation or discussion.

A.3. Level-Numbers

When specific level-numbers appear in data description entry formats, those specific level-numbers are required when such entries are used in a COBOL program. In this document, the form 01, 02, ... , 09 is used to indicate level-numbers 1 through 9.

A.4. Brackets and Braces

Brackets, **[ ]**, enclose optional items.

Braces, **{ }**, enclose a set of alternatives, one of which is required; it must be selected explicitly or implicitly. If one of the options contains only reserved words which are not keywords, that option is the default if no option is explicitly specified.

Options are indicated in a general format or a portion of a general format by vertically stacking the set of alternatives, by a series of brackets or braces or by a combination of both. An option is selected by specifying one of the alternatives or by specifying a unique combination of possibilities from a series of brackets or braces.

A.5. Ellipsis (...)

In text, other than general formats, the ellipsis shows omission of a word or words when such omission does not impair comprehension. This is the conventional meaning of the ellipsis, and the use becomes apparent in context.

In the general format, the ellipsis represents indefinite repetition of the last item. The portion of the format that may be repeated is determined as follows:

> Given **...** (the ellipsis) in a format, scanning right to left, determine the **]** (right bracket) or **}** (right brace) delimiter immediately to the left of the **...** (ellipsis); continue scanning right to left and determine the logically matching **[** (left bracket) or **{** (left brace) delimiter; the **...** (ellipsis) applies to the portion of the format between the determined pair of delimiters. Thus a **[ ]...** indicates there can be zero or more occurrences of this item while a **{ }...** indicates there can be one or more occurrences of this item.

A.6. Format Punctuation

The separators comma and semicolon may be used anywhere the separator space is used in the formats. In the source program, these separators are interchangeable.

The separator period, when used in the formats, has the status of a required word. It must be followed by a space.

A.7. Use of Special Character Words in Formats

The special character words `+', `-', `>', `<', `=', `>=', `<=', and '<>' when appearing in formats, although not underlined, are required when such portions of the formats are used.

A.8. Documentation Only

d Lines with the symbol "d" in the left margin indicate that this phrase is used for documentation only; it does not in
d any way affect how the **ICOBOL** compiler syntaxes the source or generates executable code..

B. Rules

B.1. Syntax Rules

Syntax rules define or clarify the order in which words or elements must be arranged to form larger elements such as phrases, clauses, or statements. Syntax rules may also either impose restrictions on individual words or elements or relax restrictions implied by words or elements.

B.2. General Rules

General rules define or clarify the meaning or relationship of meanings of an element or set of elements. They are used to define or clarify the semantics of the statement and the effect that it has on either compilation or execution.

C. ICOBOL Dialects and Feature-Sets

The **ICOBOL** product described by this document is a COBOL language product that can be customized at compile-time to mimic one of several popular COBOL implementations, or *dialects*. The selection of a given dialect automatically affects a number of different language attributes, such as the set of reserved words, the syntax for particular statements, the storage format for data, and even run-time behavior.

In addition, the product implements a number of language enhancements that are selectable independently of the dialect selected. These enhancements are bundled in various combinations to form a *feature-set*.

C.1. Description of **ICOBOL** Dialects

Each dialect is selectable via a compiler switch. (See the Compiler Chapter of the Developer's Guide Section starting on page 699, for a description of compiler options.) Each dialect is named and described individually below. Note that whenever the term **ICOBOL** is used in this manual, it refers collectively to all of the supported dialects. Whenever the individual dialect name is used, it refers specifically to that dialect. The supported dialects are:

- ***ANSI 74***

  This is the fundamental dialect. It is consistent with traditional Interactive COBOL. It uses ANSI-74 file status codes and file handling semantics.

- ***ANSI 85***

  This is the stricter ANSI-85 dialect. It is consistent with **ICOBOL 2** code compiled with the -M 85 option. It uses ANSI-85 file status codes and file handling semantics.

- ***VXCOBOL***

  This dialect is consistent with the syntax and semantics used by Data General's AOS/VS COBOL and by Envyr Corporation's **VXCOBOL** product.

C.2. Notation of Dialect Differences

(1) Many language features and runtime behavior are common to all dialects. In that case, no dialect notation is necessary, and support with all dialects is assumed. The term "**ICOBOL**" refers to the product as a whole and includes all dialects, except where explicitly noted.

(2) Where there are differences, they are noted in the documentation with flags to note those exceptions. Most differences are between the following sets of dialects, and these are the most common flags you will see in the documentation. For example,

- (***ANSI 74*** and ***ANSI 85***)
- (***VXCOBOL***)

Less frequently, differences will be noted with the following flags:

- (***ANSI 74***)
- (***ANSI 85***)

(3) Some features and behavior are found only in one dialect and are so marked. For example in the DATA DIVISION:

FEEDBACK Clause (***VXCOBOL***)

and

RECORD Clause (***ANSI 74*** and ***ANSI 85***)

(4) Differences are flagged at the highest level appropriate. A COBOL statement may be supported in one dialect but not another; in that case, the notation will appear at the highest level for the statement, indicating which dialect(s) support the statement. Most COBOL statements are common to all dialects but have minor differences among dialects, as, for example in the following documentation excerpt from ACCEPT statement in the PROCEDURE DIVISION:

ANSI 74 and ***ANSI 85***:

(4) During the execution of an ACCEPT statement for a screen item that contains SECURE NO ECHO, any characters entered by the user will not be echoed, and the cursor will not move as the characters are entered.

VXCOBOL:

(5) During the execution of an ACCEPT statement, any characters entered by the user will not be echoed. Additionally, the cursor will not move as the characters are entered.

C.3. Description of Feature-sets

A feature-set is an enhancement or a set of enhancements that can be enabled independently of the specific dialect that is selected. In a manner similar to the dialect, however, a feature-set may affect the set of reserved words, that syntax for existing language features, additional syntax that is specific to the feature-set, and even run-time behavior. Each feature-set is denoted by a feature-set name and an optional level indicator. The naming reflects this scheme.

The feature-sets are as follows:

- ***ISQL-1***

  This is Integrated SQL level 1. This level includes integrated support for a number of the SQL data types and operators, as well as basic support for dynamic queries using PREPARE and EXECUTE.

- ***ISQL-2***

  This is integrated SQL level 2. **This level is not yet implemented**. It will extend the integrated support of the SQL language with a much larger set of supported SQL statements. Additional statements will include CREATE, DROP, SELECT, INSERT, and DELETE.

C.4. Notation of Feature-set Differences

Where there are differences created by the presence of a feature-set, they are noted in the documentation with flags to note those exceptions. When the differences are the same for all levels of a feature-set, they are denoted by using just the base feature-set name. For example,

- (***ISQL***) Applies to all levels of the ***ISQL*** feature-set
- (***ISQL-1***) Applies only to level-1 features within the ***ISQL*** feature-set
- (***ISQL-2***) Applies only to level-2 features within the ***ISQL*** feature-set

II. COBOL SOURCE PROGRAM

A. General Description

A COBOL source program is a syntactically correct set of COBOL statements.

B. Concepts

B.1. Character Set

The most basic and indivisible unit of the language is the character. The set of characters used to form COBOL character-strings and separators includes the letters of the alphabet, digits, and special characters. This character set consists of the characters as defined under COBOL Character Set in the glossary. In the case of nonnumeric literals, comment-entries, and comment lines, the character set is expanded to include the computer's entire character set. The characters allowable in each type of character-string and as separators are defined in the section below.

B.2. Language Structure

The individual characters of the language are concatenated to form character-strings and separators. A separator may be concatenated with another separator or with a character-string. A character-string may only be concatenated with a separator. The concatenation of character-strings and separators forms the text of a source program.

B.2.1 Separators

A separator is a character or two contiguous characters formed according to the following rules:

(1) *Space.* The punctuation character space is a separator. Anywhere a space is used as a separator or as part of a separator, more than one space may be used. All spaces immediately following the separators comma, semicolon, or period are considered part of that separator and are not considered to be the separator space.

(2) *Comma* and s*emicolon*. Except when the comma is used in a PICTURE character-string, the punctuation characters comma and semicolon, immediately followed by a space, are separators that may be used anywhere the separator space is used. They may be used to improve program readability.

(3) *Period*. The punctuation character period, when followed by a space is a separator. It must be used only to indicate the end of a sentence, or as shown in formats.

(4) *Parentheses.* The punctuation characters right and left parentheses are separators. Parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts, reference modifiers, arithmetic expressions, or conditions.

(5) *Quotation mark*. The punctuation character quotation mark is a separator. An opening quotation mark must be immediately preceded by a space or left parenthesis; a closing quotation mark, when paired with an opening quotation mark, must be immediately followed by one of the separators space, comma, semicolon, period, or right parenthesis.

(6) *Colon*. The punctuation character colon is a separator and is required when shown in the general formats.

(7) The separator space may optionally immediately precede all separators except:

a. As specified by reference format rules.

b. The separator closing quotation mark. In this case, a preceding space is considered as part of the nonnumeric literal and not as a separator.

(8) The separator space may optionally immediately follow any separator except the opening quotation mark. In this case, a following space is considered as part of the nonnumeric literal and not as a separator.

(9) Pseudo-text delimiters. Pseudo-text delimiters are separators. An opening pseudo-text delimiter must be immediately preceded by a space. A closing pseudo-text delimiter must be immediately followed by one of the separators space, comma, semi-colon, or period. Pseudo-text delimiters may appear only in balanced pairs delimiting pseudo-text.

Any punctuation character which appears as part of the specification of a PICTURE character-string or numeric literal is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by the separators space, comma, semicolon, or period.

The rules established for the formation of separators do not apply to the characters which comprise the contents of nonnumeric literals, comment-entries, or comment lines.

B.2.2 Character-Strings

A character-string is a character or a sequence of contiguous characters which forms a COBOL word, a literal, a PICTURE character-string, or a comment-entry. A character-string is delimited by separators.

B.2.2.1 COBOL Words

A COBOL word is a character-string of not more than 30 characters which forms a user-defined word, a system-name, or a reserved word. Each character of a COBOL word is selected from the set of letters, digits, and the hyphen. The hyphen may not appear as the first or last character. Each lowercase letter is considered to be equivalent to its corresponding uppercase letter. Within a source program, reserved words and user-defined words form disjoint sets; reserved words and system-names form disjoint sets; system-names and defined words form intersecting sets. The same COBOL word may be used as a system-name and as a user-defined word within a source program; and the class of a specific occurrence of this COBOL word is determined by the context of the clause or phrase in which it occurs.

NOTE: ANSI standard COBOL required that COBOL words be no more than 30 characters. The *VXCOBOL* dialect will issue an info message at compile time if a word exceeds 30 characters, but otherwise will allow up to 50 characters in a word.

B.2.2.1.1 User-Defined Words

A user-defined word is a COBOL word that must be supplied by the user to satisfy the format of a clause or statement. Each character of a user-defined word is selected from the set of characters `A', `B', `C', ... , `Z', `a', `b', `c', ... , `z', `0', ... , `9', and `-' except that the `-' may not appear as the first or last character.

(***ISQL-2***) A user-defined word of type sql-column-name, sql-table-name, or sql-name is selected from the set of characters `A', `B', `C', ... , `Z', `a', `b', `c', ... , `z', `0', ... , `9', and `_' (underscore). Additionally, it must begin with an alphabetic character, it may not have two or more contiguous underscores, and it may not have an underscore as the last character.

The types of user-defined words are:

1. alphabet-name
2. class-name *
3. condition-name
4. data-name
5. file-name
6. index-name
7. level-number
8. mnemonic-name
9. paragraph-name
10. program-name
11. record-name
12. screen-name
13. section-name
14. sql-column-name**
15. sql-table-name**
16. sql-name**
17. symbolic-character *
18. text-name

* this type is not used in ***VXCOBOL***
** this type is only used with ***ISQL-2***

Within a given source program, the defined words are grouped into the following disjoint sets:

1. alphabet-names
2. class-name *
3. condition-names, data-names, record-names, and screen-name
4. file-names
5. index-names
6. mnemonic-names
7. paragraph-names
8. program-names
9. section-names
10. symbolic-characters *
11. text-names
12. sql-table-names & sql-column-names**
13. sql-names**

* this type is not used in ***VXCOBOL***
** this type is only used with ***ISQL-2***

All user-defined words, except level-numbers, can belong to one and only one of these disjoint sets. Further, all user-defined words within a given disjoint set must be unique, except as specified in the rules for uniqueness of reference.

With the exception of section-names, paragraph-names, and level-numbers, all user-defined words must contain at least one alphabetic character. Level-numbers need not be unique; a given specification of a level-number may be identical to any other level-number.

B.2.2.1.1.1 Condition-Name

A condition-name is a name which is assigned to a specific value, set of values, or range of values, within a complete set of values that a data item may assume. The data item itself is called a conditional variable.

Condition-names may be defined in the Data Division or in the SPECIAL-NAMES paragraph within the Environment Division where a condition-name must be assigned to the on status or off status, or both, of user-defined switches.

A condition-name is used in conditions as an abbreviation for the relation condition; this relation condition posits that the associated conditional variable is equal to one of the set of values to which that condition-name is assigned. A condition-name is also used in a SET statement, indicating that the associated value is to be moved to the conditional variable.

B.2.2.1.1.2 Mnemonic-Name

A mnemonic-name assigns a user-defined word to a user-defined-switch. These associations are established in the SPECIAL-NAMES paragraph of the Environment Division (see The SPECIAL-NAMES Paragraph, page 74).

B.2.2.1.1.3 Paragraph-Name

A paragraph-name is a word which names a paragraph in the Procedure Division. Paragraph-names are equivalent if, and only if, they are composed of the same sequence of the same number of digits and/or characters.

B.2.2.1.1.4 Section-Name

A section-name is a word which names a section in the Procedure Division. Section-names are equivalent if, and only if, they are composed of the same sequence of the same number of digits and/or characters.

B.2.2.1.1.5 Other User-Defined Names

All other types of user-defined words are defined in the glossary.

B.2.2.1.2 System-Names

A system-name is a COBOL word which is used to communicate with the operating environment. Each character used in the formation of a system-name must be selected from the set of characters `A', `B', `C', ... , `Z', `0', ... , `9', and `-' except that the `-' may not appear as the first or last character.

B.2.2.1.3 Reserved Words

A reserved word is a COBOL word that is one of a specified list of words which may be used in COBOL source programs, but which must not appear in the program as user-defined words or system-names. Reserved words can only be used as specified in the general formats. The reserved word table can be found in APPENDIX K on page 847.

Reserved words satisfy the following conditions:

(1) Reserved words do not begin with the characters `0', ... , `9', `X', `Y', or `Z' except for the reserved words YYYYMMDD, YYYYDDD, ZERO, ZEROES, ZEROS, and ZONE (***ISQL***).

(2) Reserved words do not contain only one alphabetic character.

(3) Reserved words do not start with 1 or 2 characters followed by `-' except for the reserved words I-O, I-O-CONTROL, and reserved words which begin with `B-' or `DB-'.

(4) Reserved words do not contain two or more contiguous hyphens.

(5) Reserved words are always shown as uppercase, although they may be written in mixed or lowercase with each lowercase letter being equivalent to the corresponding uppercase letter.

There are three types of reserved words:

1. required words 2. optional words 3. special purpose words

B.2.2.1.3.1 Required Words

A required word is a word whose presence is required when the format in which the word appears is used in a source program.

Required words are of two types:

(1) Keywords. Within each format, such words are uppercase and underlined.

(2) Special character words. These are the arithmetic operators and relation characters.

B.2.2.1.3.2 Optional Words

Within each format, uppercase words that are not underlined are called optional words and may be specified at the user's option with no effect on the semantics of the format.

B.2.2.1.3.3 Special Purpose Words

There are two types of special purpose words:

1. figurative constants
2. special registers

B.2.2.1.3.3.1 Figurative Constants

Certain reserved words are used to name and reference specific constant values. These reserved words are specified under Figurative Constant Values on page 46.

B.2.2.1.3.3.2 Special Registers

Certain reserved words are used to name and reference special registers. Special registers are certain compiler-generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features. Unless specified otherwise in these specifications, one special register of each type is allocated for each program. In the general formats of this specification, a special register may be used, unless otherwise restricted, wherever data-name or identifier is specified provided that the special register is the same category as the data-name or identifier. If qualification is allowed, special registers may be qualified as necessary to provide uniqueness. See page 124 Qualification.

Special registers include: ADDRESS OF, LENGTH OF, LINAGE-COUNTER, and SQLSTATE (***ISQL***).

B.2.2.2 Literals

A literal is a character-string whose value is implied by an ordered set of characters of which the literal is composed, by specification of a reserved word which references a figurative constant, or (***ISQL***) by specification of a reserved word (or words) in combination with a non-numeric literal value. Every literal belongs to one of the following types:

(1) nonnumeric
(2) numeric
(3) date-time (***ISQL***)
(4) interval (***ISQL***)

NOTE: For simplicity in the formats that follow, the literals that make use of quotation marks or apostrophes as delimiters are only shown using quotation marks. Simply remember that the closing delimiter must match the opening delimiter.

B.2.2.2 .1 Nonnumeric Literals

A nonnumeric literal is a character-string enclosed in either quotation marks or apostrophes. The length of a nonnumeric literal applies to its representation in the object program.

B.2.2.2.1.1 General Format

"{*character-1*}... "

B.2.2.2.1.2 Syntax Rules

(1) *Character-1* may be any character in the computer's character set.

(2) If *character-1* is to represent the quotation mark, two contiguous quotation mark characters must be used to represent a single occurrence of that character, or the delimiting characters must be apostrophes.

(3) If *character-1* is to represent the apostrophe, two contiguous apostrophes characters must be used to represent a single occurrence of that character, or the delimiting characters must be quotations.

(4) (***ISQL***) There may be zero occurrences of *character-1*.

B.2.2.2.1.3 General Rules

(1) The value of a nonnumeric literal in the object program is the value represented by *character-1*.

(2) The separator quotation mark or apostrophe that delimits the nonnumeric literal is not part of the value of the nonnumeric literal.

(3) All nonnumeric literals are of category alphanumeric.

(4) With the -G n compiler switch, a single character may be represented by enclosing a value in angle brackets. For example, "<014>" represents the formfeed character, since octal 14 is the ASCII code for formfeed. See page 704 for complete details on the General switch to the compiler.

(5) (***ISQL***) When there are zero occurrences of *character-1* in the literal, it is known as the *null string* and it is a literal of zero length. When the value is moved to a an item of usage Character Varying, it results in the data item also having zero length. When used with items without the Varying attribute, normal padding rules apply.

B.2.2.2 .2 Nonnumeric Hexadecimal Literals

A nonnumeric hexadecimal literal is a special type of nonnumeric literal. It is a character string of one or more hexadecimal digits which is delimited at the beginning by the uppercase character 'X' followed immediately by a quotation mark or apostrophe and delimited at the end by a matching quotation mark or apostrophe. The length of a nonnumeric hexadecimal literal applies to its representation in the object program. Odd digit counts assume a leading zero to ensure an even number of bytes.

B.2.2.2.2.1 General Format

X"{*character-1*}..." or X '{*character-1*}...'

B.2.2.2.2.2 Syntax Rules

(1) *Character-1* may be the digits '0' through '9', the characters 'A' through 'F' or the characters 'a' through 'f'. The uppercase and lowercase characters are considered equivalent.

(2) *Character-1* may occur from one to 160 times. If *character-1* occurs an odd number of times, a '0' is assumed to immediately follow the opening quotation mark or apostrophe so that there are an even number of occurrences.

B.2.2.2.2.3 General Rules

(1) The value of a nonnumeric hexadecimal literal in the object program is the ASCII character represented by each pair of occurrences of *character-1*. (Each ASCII character is represented as a pair of hexadecimal digits.)

(2) The leading 'X' and quotation marks or apostrophes that delimit the nonnumeric hexadecimal literal are not part of the value of the literal.

(3) Nonnumeric hexadecimal literals are category alphanumeric.

(4) Nonnumeric hexadecimal literals may be used anywhere that a nonnumeric literal may be used.

B.2.2.2 .3 Numeric Literals

A numeric literal is a character-string whose characters are selected from the digits `0' through `9', the plus sign, the minus sign, and the decimal point. Numeric literals can be from 1 through 18 digits in length. The rules for the formation of numeric literals are as follows:

(1) A literal must contain at least one digit.

(2) A literal must not contain more than one sign character. If a sign is used, it must appear as the left-most character of the literal. If the literal is unsigned, the literal is nonnegative.

(3) A literal must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the right-most character. If the literal contains no decimal point, the literal is an integer.

(4) If a literal conforms to the rules for the formation of numeric literals but is enclosed in quotation marks, it is a nonnumeric literal and is treated as such by the compiler.

(5) The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal is category numeric. The size of a numeric literal in standard data format characters is equal to the number of digits in the string of characters as specified by the user.

B.2.2.2 .4 Numeric Hexadecimal Literals

A numeric hexadecimal literal is a special type of numeric literal. It is a character string of one or more hexadecimal digits which is delimited at the beginning by the uppercase character 'H' followed immediately by a quotation mark or apostrophe and delimited at the end by a matching quotation mark or apostrophe. The length of a numeric hexadecimal literal applies to its representation in the object program.

B.2.2.2.4.1 General Format

H"{*character-1*}..."

B.2.2.2.4.2 Syntax Rules

(1) *Character-1* may be the digits '0' through '9', the characters 'A' through 'F' or the characters 'a' through 'f'. The uppercase and lowercase characters are considered equivalent.

(2) *Character-1* may occur from one to 8 times.

B.2.2.2.4.3 General Rules

(1) The value of a numeric hexadecimal literal is the algebraic quantity represented by the characters within the quotes interpreted as a non-negative hexadecimal integer.

(2) The leading 'H' and quotation marks that delimit the numeric hexadecimal literal are not part of the value of the literal.

(3) The size of a numeric hexadecimal literal is the size of an equivalent decimal representation of the same algebraic quantity.

(4) Numeric hexadecimal literals are category numeric.

(5) Numeric hexadecimal literals may be used anywhere in the source program that a numeric literal may be used.

B.2.2.2 .5 Figurative Constant Values

Figurative constant values are generated by the compiler and referenced through the use of the reserved words given below. These words must not be bounded by quotation marks when used as figurative constants. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

The figurative constant values and the reserved words used to reference them are as follows:

(1) [ALL] ZERO, [ALL] ZEROS, [ALL] ZEROES - Represents the numeric value `0', or one or more of the character `0' from the computer's character set.

(2) [ALL] SPACE, [ALL] SPACES - Represents one or more of the character space from the computer's character set.

(3) [ALL] HIGH-VALUE, [ALL] HIGH-VALUES - Except in the SPECIAL-NAMES paragraph, represents one or more of the character that has the highest ordinal position in the program collating sequence.

(4) [ALL] LOW-VALUE, [ALL] LOW-VALUES - Except in the SPECIAL-NAMES paragraph, represents one or more of the character that has the lowest ordinal position in the program collating sequence.

(5) [ALL] QUOTE, [ALL] QUOTES - Represents one or more of the character ` " '. The word QUOTE or QUOTES cannot be used in place of a quotation mark in a source program to bound a nonnumeric literal. Thus QUOTE ABD QUOTE is incorrect as a way of stating the nonnumeric literal "ABD".

(6) ALL literal - Represents all or part of the string generated by successive concatenations of the characters comprising the literal. The literal must be a nonnumeric literal. The literal must not be a figurative constant.

NOTE: The following is supported in the *ANSI 74* and *ANSI 85* dialects and not the *VXCOBOL* dialect.

(7) [ALL] symbolic-character - Represents one or more of the character specified as the value of this symbolic-character in the SYMBOLIC CHARACTERS clause of the SPECIAL-NAMES paragraph.

NOTE: The following is supported in the *VXCOBOL* dialect and not the *ANSI 74* and *ANSI 85* dialects.

(8) [ALL] CR - Represents one or more NEW LINE characters.

When a figurative constant represents a string of one or more characters, the length of the string is determined by the compiler from context according to the following rules:

(1) When a figurative constant is specified in a VALUE clause, or when a figurative constant is associated with another data item (e.g., when the figurative constant is moved to or compared with another data item), the string of characters specified by the figurative constant is repeated character by character on the right until the size of the resultant string is greater than or equal to the number of character positions in the associated data item, This resultant string is then truncated from the right until it is equal to the number of character positions in the associated data item. This is done prior to and independent of the application of any JUSTIFIED clause that may be associated with the data item.

(2) When a figurative constant, other than ALL literal, is not associated with another data item as when the figurative constant appears in a DISPLAY, STOP, STRING, or UNSTRING statement, the length of the string is one character.

(3) When the figurative constant ALL literal is not associated with another data item, the length of the string is the length of the literal.

A figurative constant may be used whenever `literal' appears in a format with the following exceptions:

(1) If the literal is restricted to a numeric literal, the only figurative constant permitted is ZERO (ZEROS, ZEROES). **ICOBOL** also allows HIGH-VALUES and LOW-VALUES, although the compiler generates a warning.

(2) Associating the figurative constant ALL literal, where the length of the literal is greater than one, with a data item that is numeric or numeric edited is an obsolete feature in Standard COBOL. This obsolete feature is to be deleted from the next revision of Standard COBOL.

(3) When a figurative constant other than ALL literal is used, the word ALL is redundant and is used for readability only.

In all **ICOBOL** dialects, HIGH-VALUES are hex FF, and LOW-VALUES are hex 00.

Each reserved word that is used to reference a figurative constant value is a distinct character-string with the exception of the constructs using the word ALL, such as ALL literal, ALL SPACES, etc., which are composed of two distinct character-strings.

B.2.2.2 .6 Date Literals (***ISQL***)

A date literal specifies an SQL date value.

B.2.2.2.6.1 General Format

DATE "YYYY-MM-DD"

B.2.2.2.6.2 Syntax Rules

(1) *YYYY* specifies a numeric year field of exactly four digits.

(2) *MM* specifies a numeric month field of exactly two digits.

(3) *DD* specifies a numeric day field of exactly two digits.

B.2.2.2.6.3 General Rules

(1) The date literal is class date-time and category date.

(2) The value of year field may range from 0001 to 9999.

(3) The values of month and day fields must fulfill the rules for valid values within the Gregorian calendar.

(4) A date literal may appear anywhere the general formats allow an item of category date to appear and where the item is a sending (value) operand.

B.2.2.2 .7 Time Literals (***ISQL***)

A time literal specifies an SQL time value.

B.2.2.2.7.1 General Format

TIME "hh:mm:ss[.ffffff]"

B.2.2.2.7.2 Syntax Rules

(1) The brackets that appear in the format above are not part of the literal, but have their usual meaning of showing optional parts.

(2) *hh* specifies a numeric hours field of exactly two digits.

(3) *mm* specifies a numeric minutes field of exactly two digits.

(4) *ss* specifies a numeric seconds field of exactly two digits.

(5) .ffffff specifies a numeric fraction field of one to six digits, which is optional.

B.2.2.2.7.3 General Rules

(1) The time literal is of class date-time and category time.

(2) The value of hour field may range from 00 to 23.

(3) The values of minutes and seconds fields may range from 00 to 59.

(4) The value of the fraction field may range from .000000 to .999999.

(5) A time literal may appear anywhere the general formats allow an item of category time to appear and where the item is a sending (value) operand.

B.2.2.2 .8 Timestamp Literals (***ISQL***)

A timestamp literal specifies an SQL timestamp value, which is the combination of an SQL date value and an SQL time value separated by a single space.

B.2.2.2.8.1 General Format

TIMESTAMP "YYYY-MM-DD hh:mm:ss[.ffffff]"

B.2.2.2.8.2 Syntax Rules

(1) The brackets that appear in the format above are not part of the literal, but have their usual meaning of showing optional parts.

(2) The rules for the various fields are found in the preceding sections entitled Date Literals and Time Literals.

(3) The date part of the literal is separated from the time part of the literal by exactly one space.

B.2.2.2.8.3 General Rules

(1) The timestamp literal is of class date-time and category timestamp.

(2) The rules for the values of the various fields are found in the preceding sections that describe the general rules for date literals and time literals.

(3) A timestamp literal may appear anywhere the general formats allow an item of category timestamp to appear and where the item is a sending (value) operand.

B.2.2.2 .9 Interval Literals (***ISQL***)

An interval literal specifies an SQL interval value. Each interval has a start specification and an optional end specification. The start and end specifications may be used in various combinations to create different interval ranges. An SQL interval is one of two disjoint kinds: the year-month interval and the day-time interval. In order to help simplify the formats, we have divided the rules according to these two kinds. For the year-month interval, the start and end specifications are from the set YEAR and MONTH. For the day-time interval, the start and end specifications are from the set DAY, HOUR, MINUTE, and SECOND.

B.2.2.2 .9.1 Year-Month Interval Literals (***ISQL***)

Within the year-month literals, there are three combinations of the start and end specifications.

B.2.2.2.9.1.1 General Format

INTERVAL "[+/-][Y...]Y-[M]M" YEAR TO MONTH
INTERVAL "[+/-][Y...]Y" YEAR
INTERVAL "[+/-][M...]M" MONTH

B.2.2.2.9.1.2 Syntax Rules

(1) The brackets and ellipses in the format above are not part of the literal, but have their usual meaning of showing optional items and repeated items.

(2) *[+/-]* specifies an optional sign.

(3) *[Y...]Y* specifies a numeric number-of-years field of one to four digits.

(4) *[M]M* specifies a numeric number-of-months field of one or two digits.

(5) *[M...]M* specifies a numeric number-of-months field of one to six digits.

(6) There are no intervening spaces between the sign and the year or month field.

(7) The year and month fields are separated by a single intervening hyphen with no spaces.

B.2.2.2.9.1.3 General Rules

(1) The year-month literal is of class interval and category year-to-month.

(2) The value of number-of-years field may range from 0 to 9999.

(3) The value of number-of-months field may range from 0 to 11 when participating in a year-to-month interval, and from 0 to 999999 when participating in a month interval.

(4) The month field in a YEAR TO MONTH interval literal is always considered to have 2 digits of precision, even if it is specified with only a single digit in the source text.

(5) Leading zeros are allowed in the leftmost field and participate in determining the precision of that field, just as they do numeric literals. When used in comparisons, the algebraic value of the field is used. Thus, both INTERVAL "0023-01" YEAR TO MONTH and INTERVAL "23-1" YEAR TO MONTH represent the interval of 23 years and 1 month, but one has a precision of 4 and the other a precision of 2 for the year field.

(6) A year-month interval literal may appear anywhere the general formats allow an item of class interval and category year-month and the item is a sending (value) operand. In some cases, the general formats will allow an interval item and the general rules will define any restrictions on the category of the item.

B.2.2.2 .9.2 Day-Time Interval Literals (***ISQL***)

Within the day-time literals, there are multiple combinations of the start and end specifications.

B.2.2.2.9.2.1 General Format

INTERVAL "[+/-][D...]D" DAY
INTERVAL "[+/-][D...]D [h]h" DAY TO HOUR
INTERVAL "[+/-][D...]D [h]h:mm" DAY TO MINUTE
INTERVAL "[+/-][D...]D [h]h:mm:ss[.ff...]" DAY TO SECOND

INTERVAL "[+/-][h...]h" HOUR
INTERVAL "[+/-][h...]h:mm" HOUR TO MINUTE
INTERVAL "[+/-][h...]h:mm:ss[.ff...]" HOUR TO SECOND

INTERVAL "[+/-][m...]m" MINUTE
INTERVAL "[+/-][m...]m:ss[.ff...]" MINUTE TO SECOND

INTERVAL "[+/-][s...]s[.ff...]" SECOND

B.2.2.2.9.2.2 Syntax Rules

(1) The brackets and ellipses in the format above are not part of the literal, but have their usual meaning of showing optional items and repeated items.

(2) *[+/-]* specifies an optional sign.

(3) *[D...]D* specifies a numeric number-of-days field of one to seven digits.

(4) *[h]h* specifies a numeric number-of-hours field of one or two digits.

(5) *[m...]m* specifies a numeric number-of-minutes field of one to ten digits.

(6) *mm* specifies a numeric number-of-minutes field of exactly two digits.

(7) *[s...]s* specifies a numeric number-of-seconds field of one to twelve digits.

(8) *ss* specifies a numeric number-of-seconds field of exactly two digits.

(9) *[.ff...]s* specifies a numeric fractional seconds field of one to six digits.

(10) The day and hour fields are separated by a single intervening space.

(11) The hour, minute and second fields are separated by a single colon.

B.2.2.2.9.2.3 General Rules

(1) The day-time literal is of class interval and category day-to-time.

(2) When the field corresponds to the start specification for the interval (the leftmost, or most significant field), the value is bounded by the number of available digits as specified in the syntax rules of the field.

(3) Leading zeros are allowed in the leftmost field and participate in determining the precision of that field, just as they do in numeric literals. When used in comparisons, the algebraic value of the field is used.

(4) When the hours field is not the most significant field, it must range in value from 0 to 23.

(5) When the hours field is not the most significant field, it is assumed to have a precision of two digits, even if it is written with only a single digit.

(6) When the minutes field is not the most significant field, it must range in value from 0 to 59.

(7) When the seconds field is not the most significant field, it must range in value from 0 to 59.

(8) The fractional seconds field is limited to six digits, yielding a range from 0 to .999999. If the decimal point is specified, at least one fractional digit must be specified.

(9) A day-time interval literal may appear anywhere the general formats allow an item of class interval to appear and where the interval variable is a sending (value) operand. In some cases, the general formats will allow an interval item and the general rules will define any restrictions on the category of the item.

B.2.2.3 LINAGE-COUNTER

The reserved word LINAGE-COUNTER is a name for a line counter generated by the presence of a LINAGE clause in a file description entry. The implicit description is that of an unsigned integer whose size is equal to *integer-1* or the data item referenced by *data-name-1* in the LINAGE clause. LINAGE-COUNTER may be referenced only in Procedure Division statements; however only the input-output control system may change the value of LINAGE-COUNTER.

If you have more than one print-file, you can qualify LINAGE-COUNTER with the filename in the Procedure Division so that the compiler knows the output record you are using with LINAGE-COUNTER.

B.2.2.4 PICTURE Character-Strings

A PICTURE character-string consists of certain symbols which are composed of the currency symbol and certain combinations of characters in the COBOL character set. An explanation of the PICTURE character-string and the rules that govern its use are given under the PICTURE clause section, which begins on page 176.

Any punctuation character which appears as part of the specification of a PICTURE character-string is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string.

B.2.2.5 Comment-Entries

A comment-entry is an entry in the Identification Division that may be any combination of characters from the computer's character set. Comment-entry is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL. A comment-entry is delimited by the next character-string that begins in Area A.

B.3. Program and Run Unit Organization and Communication

Complete data processing problems are frequently solved by developing a set of separately compilable but logically coordinated programs which at some time prior to execution may be compiled and assembled into a complete problem solution. The organization of COBOL programs and run units supports this approach of dividing large problem solutions into small, more manageable, portions which may be programmed and validated independently.

B.3.1 Program and Run Unit Organization

There are two levels of computer programs in a COBOL environment. These are the source level and the object level.

At the source level, the most inclusive unit of a computer program is a source program. A source program is a syntactically correct set of COBOL statements as specified in this document and consists of an Identification Division followed optionally by an Environment Division and/or a Data Division and/or a Procedure Division. A source program can be converted by the COBOL compiler into an object program that either alone, or together with other object programs, is capable of being executed.

The Procedure Division of a source program is organized into a sequence of procedures of two types. Declarative procedures, normally termed declaratives, are procedures which will be executed only when special conditions occur during the execution of a program. Nondeclarative procedures are procedures which will be executed according to the normal flow of control within a program. Declaratives may contain nondeclarative procedures but these will be executed only during the execution of the declarative which contains them. Nondeclarative procedures may contain other nondeclarative procedures but must not contain a declarative. Neither declaratives nor nondeclarative procedures can contain programs. In other words, in COBOL the terms `procedure' and `program' are not synonyms.

At the object level the most inclusive unit of organization of computer programs is the run unit. A run unit is a complete problem solution consisting of an object program or of several inter-communicating object programs. A run unit is an independent entity that can be executed without communicating with, or being coordinated with, any other run unit except that it may process data files or set and test switches that were written or will be read by other run units.

When a program is called, parameters upon which it is to operate may be passed to it by the program which calls it. As any separately compiled program may be the first program executed in a run unit, the first executed program of a run unit may receive parameters.

B.3.2 Accessing Data and Files

Some data items have associated with them a storage concept determining where data item values and other attributes of data items are represented with respect to the programs of a run unit. Likewise, file connectors have associated with them a storage concept determining where information concerning the positioning and status of a file and other attributes of file processing are represented with respect to the programs of a run unit.

B.3.2.1 Names

A data-name names a data item. A file-name names a file connector.

A name may be used only to refer to the object with which it is associated from within the program in which the name is declared.

B.3.2.2 Objects

Accessible data items usually require that certain representations of data be stored. File connectors usually require that certain information concerning files be stored.

B.3.2.2.1 Object Types

B.3.2.2.1.1 Working Storage Records

Working storage records are allocations of sufficient storage to satisfy the record description entries in that section. Each record description entry in a program declares a different object. Renaming and redefining do not declare new objects; they provide alternate groupings or descriptions for objects which have already been declared.

B.3.2.2.1.2 File Connectors

File connectors are storage areas which contain information about a file and are used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

B.3.2.2.1.3 Record Areas for Files

No particular record description entry in the File Section is considered to declare the storage area for the record. Rather, the Storage area is the maximum required to satisfy associated record description entries. These entries may describe fixed or variable length records. In this discussion, record description entries are said to be associated in two cases. First, when record description entries are subordinate to the same file description entry, they are always associated. Second, when record description entries are subordinate to different file description entries and these file description entries are referenced in the same SAME RECORD AREA clause, they are associated. All associated record description entries are redefinitions of the same storage area.

B.3.2.2.1.4 Other Objects

Examples of other information declared in COBOL programs are the description entries and control information used by programs in a run unit as they communicate with each other.

B.3.2.2.2 The EXTERNAL Attribute of an Object

A data item or file connector is external if the storage associated with that object is associated with the run unit rather than with any particular program within the run unit. An external object may be referenced by any program which describes the object. References to an external object from different programs using separate descriptions of the object are always to the same object.

An object is internal if the storage associated with that object is associated only with the program which describes the object.

B.3.2.2.2.1 Working Storage Records

A data record described in the Working-Storage Section is given the external attribute by the presence of the EXTERNAL clause in its data description entry. Any data item described by a data description entry subordinate to an entry describing an external record also attains the external attribute. If a record or data item does not have the external attribute, it is part of the internal data of the program in which it is described.

B.3.2.2.2.2 File Connectors

A file connector is given the external attribute by the presence of the EXTERNAL clause in the associated file description entry. If the file connector does not have the external attribute, it is internal to the program in which the associated file-name is described.

B.3.2.2.2.3 Record Areas for Files

The data records described subordinate to a file description entry which does not contain the EXTERNAL clause or a sort-merge file description entry, as well as any data items described subordinate to the data description entries for such records, are always internal to the program describing the file-name. If the EXTERNAL clause is included in the file description entry, the data records and the data items attain the external attribute.

B.3.2.2.2.4 Other Objects

Data records, subordinate data items, and various associated control information described in the Linkage section of a program are always considered internal to the program describing that data.

B.3.2.2.2.5 Program Classes

All programs which form part of a run unit may optionally possess the initial attribute.

An initial program is one whose program state is initializes when the program is called. Thus, whenever an initial program is called, its program state is the same as when the program was first called in that run unit. During the process of initializing an initial program, that program's internal data is initialized; thus an item of the program's internal data whose description contains a VALUE clause is initialized to that defined value, but an item whose description does not contain a VALUE clause is initialized to an undefined value. Files with internal file connectors associated with the program are not in the open mode. The control mechanisms for all PERFORM statements contained in the program are set to their initial states. The initial attribute is attained by specifying the INITIAL phrase in the programs Identification Division.

B.3.3 Inter-program Communication

When the complete solution to a data processing problem is subdivided into more than one program, the programs that make up the run unit must be able to communicate with each other. This communication has two pieces: the transfer of control and the passing of parameters. The need for inter-program communication arises when the programs in a run unit are separately compiled.

B.3.3.1 Transfer of Control

The CALL statement provides the means whereby control may be transferred from one program to another program within a run unit. A called program may itself contain CALL statements.

When control is transferred to a called program, execution proceeds from statement to statement beginning with the first nondeclarative statement. If control reaches a STOP RUN statement, this signals the logical end of the run unit. If control reaches an EXIT PROGRAM statement, this signals the logical end of the called program only, and control then reverts to the next executable statement following the CALL statement in the calling program. Thus the EXIT PROGRAM statement terminates only the execution of the program in which it occurs, while the STOP RUN statement terminates the execution of a run unit.

In order to call a program, a CALL statement specifies the program's name. The names assigned to each of the separately compiled programs which constitute a run unit must be unique.

Any calling program may call any separately compiled program in the run unit.

A CALL statement may be used to call programs not written in COBOL, such as builtins or user-defined subroutines added via the **ICOBOL** Link Kit.

B.3.3.2 Passing Parameters to Programs

A program calls another program in order to have the called program perform, on behalf of the calling program, some defined part of the solution of a data processing problem. In many cases it is necessary for the calling program to define to the called program the precise part of the problem solution to be executed by making certain data values, which the called program requires, available to the called program. One method for ensuring the availability of these data values is by passing parameters to a program, as is described in this paragraph. Another method is to share the data. The data values passed as parameters also may identify some data to be shared; hence, the two methods are not mutually independent.

B.3.3.2.1 Identifying Parameters

Data passed as parameters by a program calling another program must be accessible to the calling program, and the data items receiving the data must be declared in the Data Division's Linkage Section in the called program. In the called program, the parameters are identified by listing the names in the Procedure Division header, in the USING phrase, as well as declaring them in linkage Section entries. In the calling program, the parameters to be passed by the calling program are identified by listing them in the USING phrase of the CALL statement.

Position in the list of parameters in the USING phrase, not name, is what establishes the correspondence between the parameters, as they are known to the calling and called programs. That is, the first parameter on one list corresponds to the first parameter on the other, the second to the second, etc.

Thus, a program which may be called by another program may include:

```
PROGRAM-ID.  EXAMPLE.
PROCEDURE DIVISION USING NUM, PCODE, COST.
```

and may be called by executing:

```
CALL "EXAMPLE" USING NBR, PTYPE, PRICE.
```

thereby establishing the following correspondence:

Called program (EXAMPLE)	Calling program
NUM	NBR
PCODE	PTYPE
COST	PRICE

EXAMPLE 1. Identifying parameters passed by a calling program

In addition, parameter count mismatch and parameter size mismatch are flagged only at runtime, with an EXCEPTION STATUS code. That is, the number of parameters and the size of each parameter must be identical in the calling and called programs.

B.3.3.2.2 Values of parameters

The individual parameters in the CALL statement's USING phrase are passed in one of two ways:

(1) BY REFERENCE. A called program is allowed to access and modify the value of the data referenced in the calling program's CALL statement. Both programs operate on the same data.

(2) BY CONTENT. This means that the values of the parameters are copied from the calling program to the called program. Values in the calling program remain unchanged, even if modified in the called program. Storage is not shared between calling and called programs.

The parameters referenced in a called program's Procedure Division header must be described in the Linkage Section of that program's Data Division.

B.3.4 Intra-program Communication

The procedures which constitute the Procedure Division of a program communicate with one another by transferring control or by referring to common data.

B.3.4.1 Transfer of Control

There are four methods of transferring control within a program:

(1) A GO TO statement.

(2) A PERFORM statement.

(3) A declarative procedure which is activated whenever certain conditions, including errors and exceptions, occur.

(4) An input procedure associated with a SORT statement, or an output procedure associated with either a SORT or MERGE statement.

An input-output procedure can be considered as an implicit PERFORM statement which is executed in conjunction with a SORT or MERGE statement; and for this reason, the restrictions on the PERFORM statement apply equally to input-output procedures.

Stricter restrictions than those for the PERFORM statement apply to declarative procedures.

B.3.4.2 Shared Data

All the data declared in a program's Data Division may be referenced by statements in the procedures and declaratives which constitute that program.

C. Organization

With the exception of the COPY statement, the statements, entries, paragraphs, and sections of a COBOL source program are grouped into four divisions, in the following order:

1. The Identification Division
2. The Environment Division
3. The Data Division
4. The Procedure Division

The end of a COBOL source program is indicated by the absence of additional source program lines.

D. Structure

The following gives the general format and order of presentation of the entries and statements which constitute a COBOL source program.

D.1. General Format

identification-division

[*environment-division*]

[*data-division*]

procedure-division

D.2 Syntax Rules

(1) The generic terms identification-division, environment-division, data-division, and procedure-division represent a COBOL Identification Division, a COBOL Environment Division, a COBOL Data Division, and a COBOL Procedure Division, respectively.

D.3 General Rules

(1) The beginning of a division in a program is indicated by the appropriate division header. The end of a division is indicated by one of the following:

a. The division header of a succeeding division in that program.

b. That physical position after which no more source program lines occur.

E. Divisions

The Identification Division identifies the program. In addition, the user may include the date the program is written, the date the compilation of the source program is accomplished and such other information as desired under the paragraphs in the general format shown below.

The Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependent upon the physical characteristics of a specific computer. This division allows specification of the configuration of the compiling computer and the object computer. In addition, information relative to input-output control, special hardware characteristics, and control techniques can be given.

The Data Division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output.

The Procedure Division may contain declarative and nondeclarative procedures.

Execution begins with the first statement of the Procedure Division, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules indicate some other order.

F. Reference Format (Source)

F.1. General Description

The reference (source) format, which provides a standard method of describing COBOL source programs and COBOL library text, is described in terms of character positions in a line on an input-output medium. Within these definitions, each compiler accepts source programs written in reference format and produces an output listing of the source program in reference format.

The rules for spacing given in the discussion of the reference format take precedence over all other rules for spacing.

The divisions of a COBOL source program must be ordered as follows: the Identification Division, then the Environment Division, then the Data Division, then the Procedure Division. Each division must be written according to the rules for the reference format.

The **ICOBOL** compiler supports two types of reference or source formats. These are ANSI Card Format and Free-Form Format also known as CRT format. The Format switch (-F c) must be given for ANSI Card format source to be accepted by the compiler. The **ICOBOL** compiler supports source lines up to 255 characters in length in either format.

F.2. ANSI Card Format

In ANSI Card Format, the reference format for a line is represented as follows:

```
Margin   Margin   Margin            Margin          Margin
L            C   A                 B               R
| 1 | ... | 6 | 7 | 8 | 9 | 10 | 11 | 12 | ... | 72 | 73 | ... | 255|
 **Sequence **  ↑   ****Area A*****   ****Area B****  ***Comment***
  Number Area   |                                       Area
           Indicator Area
```

Margin L is immediately to the left of the left-most character position of a line.

Margin C is between the 6th and 7th character positions of a line.

Margin A is between the 7th and 8th character positions of a line.

Margin B is between the 11th and 12th character positions of a line.

Margin R is immediately to the right of the 72nd character position of a line.

The sequence number occupies six character positions (1-6), and is between margin L and margin C. Characters in this area are placed in the listing, but are not further processed by the compiler.

The indicator area is the 7th character position of a line.

Area A occupies character positions 8, 9, 10, and 11, and is between margin A and B.

Area B occupies character positions 12 through 72. It begins immediately to the right of margin B and terminates immediately to the left of margin R.

The comment area occupies character positions 73 through 255. Characters in this area are placed in the listing but are not further processed by the compiler.

If a line is shorter than 73 characters, there is no comment area and Margin R is to the right of the last character.

If tabs are used, the following rules apply:

(**ANSI 74** and **ANSI 85**) A tab character in the sequence area indicates that the remainder of the line is to be treated like a Free-Form line (see Free-Form Format below).
(***VXCOBOL***) A tab character in the sequence area indicates that the next character begins Area A and Area B begins 4 characters to the right.
A tab in the indicator area is flagged with a warning and treated as a space.
A tab in Area A is treated as a space, and it indicates that Area B is to begin with the next character.
A tab in Area B is treated as a space.

F.3. Free-Form Format (CRT)

In Free-Form format (CRT), there are no sequence number or comment areas. The only restrictions imposed are the contents of areas A and B and the use of indicator characters. The compiler reads characters until it finds a line terminator or until it has read 255 characters, whichever comes first. A line longer than 255 characters will produce an error.

In Free-Form Format the reference format for a line is represented as one of the following:

```
Margin              Margin                                          Margin
C   A               B                                                    R
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | ... | 255 |
  ↑  ****Area A*****  ****Area B********** ...
Indicator Area


Margin          Margin                                              Margin
A               B                                                        R
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | ... | 255 |
 ****Area A*****  ****Area B********** ...
```

If the first character is a "-", "*", "/", "d", or "D", position 1 is the indicator area, Area A is positions 2 through 5, and Area B is in positions 6 to the end of the line. Otherwise, there is no indicator area and Area A is in positions 1 through 4 and Area B is in positions 5 to the end of the line.

Margin C is immediately to the left of the left-most character position of a line.

Margin A is between the 1st and 2nd character positions of a line with an Indicator area or immediately to the left of the left-most character of a line.

Margin B is between the 5th and 6th character positions of a line with an indicator area and between the 4th or 5th character positions otherwise.

Margin R is immediately to the right of the right-most character position of a line.

The indicator area is the 1st character position of a line.

Area A occupies character positions 2 through 5 with an indicator area and positions 1 through 4 otherwise. It is between margin A and margin B.

Area B begins immediately to the right of margin B and terminates immediately to the left of margin R.

If tabs are used in the source, the following rules apply:
A tab in Area A is treated as a space, and it indicates that Area B is to begin with the next character.
A tab in Area B is treated as a space.

F.4. Sequence Numbers (ANSI Card Format)

The sequence number area may be used to label a source program line. The content of the sequence number area is defined by the user and may consist of any character in the computer's character set. There is no requirement that the content of the sequence number area appears in any particular sequence or be unique.

F.5. Continuation of Lines

Any sentence, entry, phrase, or clause may be continued by starting subsequent line(s) in area B. These subsequent lines are called the continuation line(s). The line being continued is called the continued line. Any word, literal, or PICTURE character-string may be broken in such a way that part of it appears on a continuation line.

A hyphen in the indicator area of a line indicates that the first nonspace character in area B of the current line is the successor of the last nonspace character of the preceding line, excluding intervening comment lines or blank lines, without any intervening space. However, if the continued line contains a nonnumeric literal without closing quotation mark, the first nonblank character in area B of the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark. All spaces at the end of the continued line are considered part of the literal. Area A of a continuation line must be blank.

If there is no hyphen in the indicator area of a line, it is assumed that the first nonspace character in the line is preceded by a space.

For the purposes of line continuation, numeric and nonnumeric hexadecimal literals are handled in the same manner as nonnumeric literals.

F.6. Blank Lines

A blank line is one that is blank from margin C to margin R, inclusive. A blank line can appear anywhere in the source program.

F.7. Comments

A comment consists of a comment indicator followed by comment-text. Any combination of characters from the compile-time computer's coded character set may be included in comment-text.

Comments serve only as documentation and have no effect on the meaning of the compilation group. A comment may be a comment line or an inline comment.

F.7.1 Comment lines

A comment line is identified by either a fixed comment indicator (an asterisk or slant) or a floating comment indicator(*>). All characters following the comment indicator up to margin R are comment-text. A comment line may be written as any line of a compilation group.

If a source listing is being produced, a comment line identified by the fixed comment indicator slant (/) causes page ejection followed by printing of the comment line; comments identified by the fixed comment indicator asterisk (*) are printed at the next available line position of the listing.

F.7.2 Inline comments

A floating comment indicator (*>) preceded by one or more character-strings in the program-text area identifies an inline comment. All characters following the floating comment indicator up to margin R are comment-text. An

inline comment may be written on any line of a compilation group except on a line that contains a floating literal continuation indicator.

F.8. Debugging Lines

A debugging line is any line with a `d' or `D' in the indicator area of the line. When in Free-Form format the `d' or `D' must be followed by a space or tab. Any debugging line that consists solely of spaces from margin A to margin R is considered the same as a blank line.

The content of a debugging line must be such that a syntactically correct program is formed with or without the debugging lines being considered as comment lines.

After all COPY statements have been processed, a debugging line will be considered to have all the characteristics of a comment line, if the -G d compiler switch is not specified, and is treated as a regular source line if the -G d (with Debug) compiler switch is specified.

Successive debug lines are allowed.

A debugging line is only permitted in the separately-compiled program after the OBJECT-COMPUTER paragraph.

F.9. Division, Section, and Paragraph Formats

F.9.1 Division Header

The division header must start in area A.

F.9.2 Section Header

The section header must start in area A.

A section consists of zero, one, or more paragraphs in the Environment Division or Procedure Division or zero, one, or more entries in the Data Division.

F.9.3 Paragraph Header, Paragraph-Name, and Paragraph

A paragraph consists of a paragraph-name followed by the separator period and by zero, one, or more sentences, or a paragraph header followed by one or more entries.

The paragraph header or paragraph-name starts in area A of any line following the first line of a division or a section.

The first sentence or entry in a paragraph begins either on the same line as the paragraph header or paragraph-name or in area B of the next nonblank line that is not a comment line. Successive sentences or entries either begin in area B of the same line as the preceding sentence or entry or in area B of the next nonblank line that is not a comment line.

When the sentences or entries of a paragraph require more than one line, they may be continued on a subsequent line or lines.

F.10. DATA DIVISION Entries

Each Data Division entry begins with a level indicator or a level-number, followed by a space, followed by the name of the subject of entry, if specified, followed by a sequence of independent clauses describing the item. The last clause is always terminated by a separator period.

There are two types of such entries: those which begin with a level indicator and those which begin with a level-number.

In the Data Division, a level indicator is an FD or SD.

In those entries that begin with a level indicator, the level indicator begins in area A, followed by at least one space, and then followed with the name of the subject of entry and appropriate descriptive information.

Those entries that begin with level-numbers are called data description entries.

A level-number has a value taken from the set of values 01, 02, ... , 49, 66, 77, 88. Level-numbers in the range 01, 02, ... , 09 may be written either as a single digit or as a zero followed by a significant digit. At least one space must separate a level-number from the word following the level-number.

In those data description entries that begin with a level-number 01 or 77, the level-number begins in area A, followed by at least one space, and then followed with its associated record-name or item-name, if specified, and appropriate descriptive information.

Data description entries may be indented. Any indentation is with respect to margin A. Each new data description entry may begin any number of positions to the right of margin A, except data description entries that begin with level-number 01 or 77 must begin in area A. The extent of indentation is determined only by the width of the physical medium. The entries on the output listing need be indented only if the input is indented. Indentation does not affect the magnitude of a level-number.

F.11. DECLARATIVES

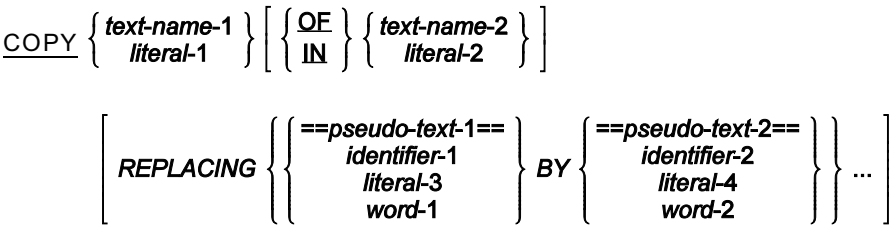
The DECLARATIVES and the pair of keywords END DECLARATIVES that precede and follow, respectively, the declaratives portion of the Procedure Division must each appear on a line by itself. Each must begin in area A and be followed by the separator period.

G. COPY Statement

G.1. Function

The COPY statement incorporates text into a COBOL source program.

G.2. General Format



G.3. Syntax Rules

(1) If more than one COBOL library is available during compilation, *text-name-1* must be qualified by *text-name-2* identifying the COBOL library in which the text associated with *text-name-1* resides. Within one COBOL library, *text-name-1* must be unique.

(2) The COPY statement must be preceded by a space and terminated by the separator period.

(3) *Pseudo-text-1* must contain one or more text words.

(4) *Pseudo-text-2* must contain zero, one or more text words.

(5) Character-strings within *pseudo-text-1* and *pseudo-text-2* may be continued.

(6) *Word-1* or *word-2* may be any single COBOL word except 'COPY'.

(7) A COPY statement may be specified in the source program anywhere a character-string or a separator, other than the closing quotation mark, may occur except that a COPY statement must not occur within a COPY statement.

(8) *Pseudo-text-1* must not consist entirely of a separator comma or a separator semicolon.

(9) If the word COPY appears in a comment-entry or in the place where a comment-entry may appear, it is considered part of the comment entry.

G.4 General Rules

(1) The compilation of a source program containing COPY statements is logically equivalent to processing all COPY statements prior to the processing of the resulting source program.

(2) The effect of processing a COPY statement is that the library text associated with *text-name-1* or *literal-1* is copied into the source program, logically replacing the entire COPY statement, beginning with the reserved word COPY and ending with the punctuation character period, inclusive. If the IN/OF clause is specified, *text-name-2* or *literal-2* represents the name of the directory containing *text-name-1* or *literal-1*.

(3) If the REPLACING phrase is not specified, the library text is copied unchanged. If the REPLACING phrase is specified. the library text is copied and each properly matched occurrence of *pseudo-text-1*, *identifier-1*,

word-1, and *literal-3* in the library text is replaced by the corresponding *pseudo-text-2*, *identifier-2*, *word-2*, or *literal-4*.

(4) For purposes of matching, *identifier-1*, *word-1*, and *literal-3* are treated as pseudo-text containing only *identifier-1*, *word-1*, or *literal-3*, respectively.

(5) The comparison operation to determine text replacement occurs in the following manner:

a. The leftmost library text word which is not a separator comma or a separator semicolon is the first text word used for comparison. Any text word or space preceding this text word is copied into the source program. starting with the first text word for comparison and first *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3* that was specified in the REPLACING phrase, the entire REPLACING phrase operand that precedes the reserved word BY is compared to an equivalent number of contiguous library text words.

b. *Pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3* match the library text if, and only if, the ordered sequence of text words that forms *pseudo-text-1*, *identifier-1*, *word-1,* or *literal-3* is equal, character for character, to the ordered sequence of library text words. For purposes of matching, each occurrence of a separator comma, semicolon, or space in *pseudo-text-1* or in the library text is considered to be a single space.

c. If no match occurs, the comparison is repeated with each next successive *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3*, if any, in the REPLACING phrase until either a match is found or there is no next successive REPLACING operand.

d. When all the REPLACING phrase operands have been compared and no match has occurred, the leftmost library text word is copied into the source program. The next successive library text word is then considered as the leftmost library text word, and the comparison cycle starts again with the first *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3*.

e. Whenever a match occurs between *pseudo-text-1*, *identifier-1*, *word-1*, or *literal-3* and the library text, the corresponding *pseudo-text-2*, *identifier-2*, *word-2*, or *literal-4* is placed into the source program. The library text word immediately following the rightmost text word that participated in the match is then considered as the leftmost text word. The comparison cycle starts again with the first *pseudo-text-1*, *literal-1*, *word-1*, or *literal-3* specified in the REPLACING phrase.

f. The comparison operation continues until the rightmost text word in the library text has either participated in a match or has been considered as a leftmost library text word and participated in a complete comparison cycle.

(6) Comment lines or blank lines occurring in the library text and in *pseudo-text-1* are ignored for purposes of matching; and the sequence of text words in the library text, if any, and in *pseudo-text-1* is determined by the rules for the reference format. (Reference Format is described on Page 59.) Comment lines or blank lines appearing in *pseudo-text-2* are copied into the resultant program unchanged whenever *pseudo-text-2* is placed into the source program as a result of text replacement. Comment lines or blank lines appearing in library text are copied into the resultant source program unchanged with the following exception: a comment line or blank line in library text is not copied if that comment line or blank line appears within the sequence of text words that match *pseudo-text-1*.

(7) Debugging lines are permitted within library text and pseudo-text. Text words within a debugging line participate in the matching rules as if the 'D' or 'd' did not appear in the indicator area. A debugging line is specified within pseudo-text if the debugging line begins in the source program after the opening pseudo-text-delimiter, but before the matching closing pseudo-text-delimiter.

(8) The syntactic correctness of the library text cannot be independently determined. Except for COPY statements, the syntactic correctness of the entire COBOL program cannot be determined until all COPY statements have been completely processed.

(9) Each text word copied from the library but not replaced is copied so as to start in the same area of the line in the resultant program as it begins in the line within the library. However, if a text word copied from the library

begins in Area A but follows another text word which also begins in Area A of the same line, and if replacement of a preceding text word in the line by replacement text of greater length occurs, the following text word begins in Area B if it cannot begin in Area A. Each text word in *pseudo-text-2* that is to be placed into the resultant program begins in the same area of the resultant program as it appears in *pseudo-text-2*. Each *identifier-2*, *literal-4,* and *word-2* that is to be placed in the resultant program begins in the same area of the resultant program as the leftmost library text word that participated in the match would appear if it had not been replaced.

Library text must conform to the rules of the COBOL reference format and be in the same format as the source program.

If additional lines are introduced into the source program as a result of a COPY statement, each text word introduced appears on a debugging line if the COPY statement begins on a debugging line or if the text word being introduced appears on a debugging line in library text. When a text word in the preceding cases, only those text words that are specified on debugging lines where the debugging line is within *pseudo-text-2* appear on debugging lines in the resultant program. If any literal specified as *literal-4* or within *pseudo-text-2* or library text is of too great length to be accommodated on a single line without continuation to another line in the resultant program and the literal is not being placed on a debugging line, additional continuation lines are introduced with contain the remainder of the literal. If replacement requires that the continued literal be continued on a debugging line, the program is in error.

(10) For purposes of compilation, text words after replacement are placed in the source program according to the rules for reference format. When copying text words of *pseudo-text-2* into the source program, additional spaces may be introduced only between text words where there already exists a space (including assumed space between source lines).

(11) If any additional lines are introduced into the source program as a result of the processing of COPY statements, the indicator area of the introduced line contains the same character as the line on which the text being replaced begins, unless the line contains a hyphen, in which case the introduced line contains a space. In the case where a literal is continued onto an introduced line which is not a debugging line, a hyphen is placed into the indicator area.

(12) If the REPLACING phrase is specified, the library text shall not contain a COPY statement and the source text that results from processing the REPLACING phrase shall not contain a COPY statement.

(13) If the REPLACING phrase is not specified, the library text may contain a COPY statement that does not include a REPLACING phrase. **ICOBOL** supports 10 levels of nesting, including the first COPY statement in the sequence. Recursive copying of library text is not permitted; that is, the library text being copied shall not cause the processing of a COPY statement that directly or indirectly copies itself.

III. IDENTIFICATION DIVISION

A. General Description

The Identification Division identifies the program. The Identification Division is required in a COBOL source program. In addition, the user may include the date the program is written and such other information as desired under the paragraphs in the general format shown below.

B. Organization

Paragraph headers identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional and may be included in this division at the user's choice, in order of presentation shown by the general format below.

The AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED and SECURITY paragraphs are obsolete elements in Standard COBOL because they are to be deleted from the next revision of Standard COBOL. We suggest that you convert them to comment lines.

B.1. Structure

The following is the general format of the paragraphs in the Identification Division, and it defines the order of presentation in the source program. Sections C and D on the following pages define the PROGRAM-ID and the DATE-COMPILED paragraphs. While the other paragraphs are not defined, each general format is formed in the same manner.

B.1.1 General Format

```
   IDENTIFICATION DIVISION.
   PROGRAM-ID.          program-name [ IS INITIAL PROGRAM ] .
d  [ AUTHOR.            [ comment-entry ]... ]
d  [ INSTALLATION.      [ comment-entry ]... ]
d  [ DATE-WRITTEN.      [ comment-entry ]... ]
d  [ DATE-COMPILED.     [ comment-entry ]... ]
d  [ SECURITY.          [ comment-entry ]... ]
```

B.1.2 Syntax Rules

(1) The *comment-entry* may be any combination of characters from the computer's character set. The continuation of the *comment-entry* by the use of the hyphen in the indicator area is not permitted; however, the *comment-entry* may be contained on one or more lines.

(2) A *comment-entry* is terminated by the next word in Area A.

(3) The optional phrases can be specified in any order.

C. PROGRAM-ID Paragraph

C.1. Function

The PROGRAM-ID paragraph specifies the name by which a program is identified and optionally assigns the INITIAL attribute to that program..

C.2. General Format

PROGRAM-ID. *program-name* [IS INITIAL PROGRAM] .

C.3. Syntax Rules

(1) The *program-name* must conform to the rules for formation of a user-defined word.

C.4. General Rules

(1) The *program-name* is currently used for documentation purposes only. The name identifying the object program and all listings is determined from the source file name and/or specific compile-time options.

(2) Although the ANSI COBOL 85 Standard requires that all CALL's use the *program-name* as specified in the PROGRAM-ID when performing CALL's, this is not enforced by **ICOBOL**.

(3) The INITIAL clause specifies that the program is initial. When an initial program is activated, the data items and file connectors contained in it are set to their initial states: VALUE clauses are applied, PERFORM controls are reset, files are put in the closed mode. See page 55 Program Classes for more information.

(4) External data is always in the last-used state except when the run unit is activated and it is in the initial state.

D. DATE-COMPILED Paragraph

D.1. Function

The DATE-COMPILED paragraph provides the compilation date in the Identification Division source program listing. The DATE-COMPILED paragraph is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

D.2. General Format

DATE-COMPILED. [*comment-entry*]...

D.3. Syntax Rules

(1) The *comment-entry* may be any combination of the characters from the computer's character set. The continuation of the *comment-entry* by the use of the hyphen in the indicator area is not permitted; however, the *comment-entry* may be contained on one or more lines.

(2) A *comment-entry* is terminated by the next word in Area A.

D.4. General Rules

(1) The paragraph-name DATE-COMPILED causes the current date to be inserted in the program listing during program compilation. If a DATE-COMPILED paragraph is present, it is replaced during compilation with a paragraph of the form:

```
DATE-COMPILED. current date.
```

IV. ENVIRONMENT DIVISION

A. General Description

The Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependent upon the physical characteristics of a specific computer. The Environment Division is optional in a COBOL source program.

B. Concepts

B.1. External Switch

An external switch is a software device, which is used to indicate that one of two alternate states exists. These alternate states are referred to as the on status and the off status of the associated external switch.

The status of an external switch may be interrogated by testing condition-names associated with that switch. The association of a condition-name with an external switch and the association of a user-specified mnemonic-name with the literal that names an external switch is established in the SPECIAL-NAMES paragraph of the Environment Division.

The scope of an external switch is the run unit and each literal that names such an external switch refers to one and only one such switch, the status of which is available to each object program functioning within that run unit.

An external switch may be altered by the SET statement, except in the ***VXCOBOL*** dialect.

C. Organization

Two sections make up the Environment Division: the Configuration Section and the Input-Output Section.

The Configuration Section deals with the characteristics of the source computer and the object computer. This section is divided into three paragraphs: the SOURCE-COMPUTER paragraph, which describes the computer configuration on which the source program is compiled; the OBJECT-COMPUTER paragraph, which describes the computer configuration on which the object program produced by the compiler is to be run; and the SPECIAL-NAMES paragraph, which provides a means for specifying the currency sign, choosing the decimal point, specifying symbolic-characters, relating switch literals to user-specified mnemonic-names, relating alphabet-names to character sets or collating sequences, and relating class-names to sets of characters.

The Input-Output Section deals with the information needed to control transmission and handling of data between external media and the object program. This section is divided into two paragraphs: the FILE-CONTROL paragraph which names and associates the files with external media; and the I-O-CONTROL paragraph which defines special control techniques to be used in the object program.

The following is the general format of the sections and paragraphs in the Environment Division, and defines the order of presentation in the source program.

ENVIRONMENT DIVISION.
[CONFIGURATION SECTION.
[SOURCE-COMPUTER. [*source-computer-entry*]]
[OBJECT-COMPUTER. [*object-computer-entry*]]
[SPECIAL-NAMES. [*special-names-entry*]]]

[INPUT-OUTPUT SECTION.
FILE-CONTROL.
{ *file-control-entry* }...

[I-O-CONTROL.

d [RERUN [ON *file-name*] EVERY { { [END OF] { REEL / UNIT } } OF *file-name* / *integer* RECORDS } / *integer* CLOCK-UNITS / *condition-name* }]...

[SAME [RECORD / SORT / SORT-MERGE] AREA FOR *file-name* { *file-name* }...]...

d [MULTIPLE FILE TAPE CONTAINS { *file-name* [POSITION *integer*] }...]...
.]]

D. CONFIGURATION SECTION

The Configuration Section is located in the Environment Division of a source program. The Configuration Section deals with the characteristics of the source computer and the object computer. This section also provides a means for specifying the currency sign; choosing the decimal point; specifying symbolic-characters; relating switch-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating sequences; and relating class-names to sets of characters. The Configuration Section is optional in the Environment Division of a COBOL source program.

The general format of the Configuration Section is shown below.

```
CONFIGURATION SECTION.
d [ SOURCE-COMPUTER.        [ source-computer-entry ] ]
  [ OBJECT-COMPUTER.        [ object-computer-entry ] ]
  [ SPECIAL-NAMES.          [ special-names-entry ] ]
```

D.1. SOURCE-COMPUTER Paragraph

D.1.1 Function

The SOURCE-COMPUTER paragraph provides a means of describing the computer upon which the program is to be compiled.

D.1.2 General Format

```
d SOURCE-COMPUTER. [ computer-name [ WITH DEBUGGING MODE ] . ]
```

D.1.3 Syntax Rules

(1) *Computer-name* is a user-defined word.

D.1.4 General Rules

(1) The WITH DEBUGGING MODE clause is ignored. All debugging lines are compiled as if they were comment lines. This behavior may be changed by using the -G d compiler switch.

(2) The SOURCE-COMPUTER *computer-name* is used for documentation purposes only.

D.2. OBJECT-COMPUTER Paragraph

D.2.1 Function

The OBJECT-COMPUTER paragraph provides a means of describing the computer on which the program is to be executed. The MEMORY SIZE clause is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

D.2.2 General Format (**ANSI 74** and **ANSI 85**)

```
d  OBJECT-COMPUTER.  [ computer-name [ MEMORY SIZE integer { WORDS      } ]
                                                           { CHARACTERS }
                                                           { MODULES    }
d       [ PROGRAM COLLATING SEQUENCE IS alphabet-name ] .]
```

D.2.3 General Format (***VXCOBOL***)

```
OBJECT-COMPUTER.
d       [ computer-name [ MEMORY SIZE integer { WORDS      } ]
                                              { CHARACTERS }
                                              { MODULES    }
d       [ PROGRAM COLLATING SEQUENCE IS { ASCII      }                  ]
                                        { EBCDIC     }
                                        { NATIVE     }
                                        { STANDARD-1 }
                                        { alphabet-name }
d       [ SEGMENT-LIMIT IS integer ] ] .
```

D.2.4 Syntax Rules

(1) *Computer-name* is a user-defined word.

D.2.5 General Rules

(1) The OBJECT-COMPUTER paragraph is used for documentation purposes only.

D.3. SPECIAL-NAMES Paragraph

D.3.1 Function

The SPECIAL-NAMES paragraph provides a means for specifying the currency sign, choosing the decimal point, relating switches to user-specified mnemonic-names and relating alphabet-names to character sets or collating sequences. ***ANSI 74*** and ***ANSI 85*** provide a way to specify symbolic characters and to relate class-names to sets of characters.

D.3.2 General Format (***ANSI 74*** and ***ANSI 85***)

```
SPECIAL-NAMES.
d     [ literal-1 IS mnemonic-1 ]...
      [ "@AUDIT" IS mnemonic-1 ]
      [ SWITCH literal-2 [ IS mnemonic-name ] [ { ON | OFF } STATUS IS condition-name ]... ]...
d     [ alphabet-name-1 IS { NATIVE | STANDARD } ]...
d     [ ALPHABET alphabet-name-1 IS { NATIVE | STANDARD | STANDARD-1 | STANDARD-2 |
                { literal-1 [ { THROUGH | THRU } literal-2 | { ALSO literal-3 }... ] }... } ]...
      [ SYMBOLIC CHARACTERS { { { symbolic-character-1 }... { IS | ARE } { integer-1 }... }...
          [ IN alphabet-name-2 ] } ]...
      [ CLASS class-name-1 IS { literal-4 [ { THROUGH | THRU } literal-5 ] }... ]...
      [ CURRENCY SIGN IS literal-6 ] [ DECIMAL-POINT IS COMMA ] . ]
```

D.3.3 General Format (***VXCOBOL***)

```
SPECIAL-NAMES.
d     [ literal-1 IS mnemonic-name-1 ]...
      [ CHANNEL integer-1 IS identifier ]...
      [ SWITCH literal-2 [ IS mnemonic-name-2 ] [ { ON | OFF } STATUS IS condition-name ]... ]...
      [ alphabet-name-1 IS { ASCII | NATIVE | STANDARD-1 | EBCDIC |
                { literal-3 [ { THRU | THROUGH } literal-4 | { ALSO literal-5 }... ] }... } ]...
      [ CURRENCY SIGN IS literal-6 ]
      [ DECIMAL-POINT IS COMMA ] . ]
```

D.3.4 Syntax Rules (***ANSI 74*** and ***ANSI 85***)

(0) *Literal-1* is an alphanumeric literal that specifies the name of a system I/O device or file.

(1) *Literal-2* must be a nonnumeric literal that is either a single or multiple character alphanumeric literal. *Mnemonic-name* may be specified only in the SET statement.

(2) If the literal phrase of the ALPHABET clause is specified, a given character must not be specified more than once in that clause.

(3) The literals specified in the literal phrase of the ALPHABET clause:

a. If numeric, must be unsigned integers and have a value within the range of one through the maximum number of characters in the native character set (256).

b. If nonnumeric and associated with a THROUGH or ALSO phrase, must each be one character in length.

(4) *Literal-1*, *literal-2*, *literal-3*, *literal-4*, and *literal-5* must not specify a symbolic-character figurative constant.

(5) The words THRU and THROUGH are equivalent.

(6) The same *symbolic-character-1* must appear only once in a SYMBOLIC CHARACTERS clause.

(7) The relationship between each *symbolic-character-1* and the corresponding *integer-1* is by position in the SYMBOLIC CHARACTERS clause. The first *symbolic-character-1* is paired with the first *integer-1*; the second *symbolic-character-1* is paired with the second *integer-1*; and so on.

(8) There must be a one-to-one correspondence between occurrences of *symbolic-character-1* and *integer-1*.

(9) The ordinal position specified by *integer-1* must exist in the native character set. If the IN phrase is specified, the ordinal position must exist in the character set named by *alphabet-name-2*.

(10) The literals specified in the *literal-4* phrase:

a. If numeric, must be unsigned integers and must have a value within the range of one through the maximum number of characters in the native character set (256).

b. If nonnumeric and associated with a THROUGH phrase, must each be one character in length.

(11) *Literal-6* must not specify a figurative constant.

(12) The ALPHABET phrase that does not have the ALPHABET keyword is an ***ANSI 74*** format that is supported for compatibility purposes only. A warning will be issued when it is used.

(13) @AUDIT must be all upper-case.

D.3.5 Syntax Rules (***VXCOBOL***)

(1) *Literal-1* is an alphanumeric literal that specifies the name of a system I/O device or file.

(2) *Mnemonic-name-1* is a mnemonic-name used in the program to refer to *literal-1*.

(3) *Integer-1* is an integer literal that specifies a channel number with a value from 1 through 12.

(4) *Literal-2* must be a nonnumeric literal that is either a single or multiple character alphanumeric literal.

(5) If the literal phrase of the alphabet clause is specified, a given character must not be specified more than once in that clause.

(6) The literals specified in the literal phrase of the alphabet clause:

a. If numeric, must be unsigned integers and have a value within the range of one through the maximum number of characters in the native character set (256).

b. If nonnumeric and associated with a THROUGH or ALSO phrase, must each be one character in length.

(7) The words THRU and THROUGH are equivalent.

(8) *Literal-6* must not specify a figurative constant.

D.3.6 General Rules (***ANSI 74*** and ***ANSI 85***)

(0) The "*literal-1* IS *mnemonic-1*" clause is for documentation purposes only.

(1) The on status and/or off status of a switch literal may be associated with *condition-names*. The status of that switch may be interrogated by testing these *condition-names*.

(2) The status of a switch may be altered by execution of a Format 3 SET statement which specifies as its operand the *mnemonic-name* associated with that switch. See the SET Statement.

(3) The ALPHABET clause provides a means for relating a name to a specified character code set and/or collating sequence. When *alphabet-name-1* is referenced in the SYMBOLIC CHARACTERS clause, the ALPHABET clause specifies a character code set.

a. If the STANDARD-1 phrase is specified, the character code set or collating sequence identified is that defined in the ANSI X3.4-1977, Code for Information Interchange. If the STANDARD-2 phrase is specified, the character code set identified is the International Reference Version of the ISO 7-bit code defined in International Standard 646, 7-bit Coded Character Set for Information Processing Interchange. Each character of the standard character set is associated with its corresponding character of the native character set.

b. If the NATIVE phrase is specified, the native character code set or collating sequence is used.

c. If the literal phrase is specified, the alphabet-name may not be referenced in a CODE-SET clause. The collating sequence identified is that defined according to the following rules:

1) The value of each literal specifies:

a) The ordinal number of a character within the native character set, if the literal is numeric. This value must not exceed the value which represents the number of characters in the native character set (256).

b) The actual character within the native character set, if the literal is nonnumeric. If the value of the nonnumeric literal contains multiple characters, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified.

2) The order in which literals appear in the ALPHABET clause specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified.

3) Any characters within the native collating sequence, which are not explicitly specified in the literal phrase, assume a position, in the collating sequence being specified, greater than any of the explicitly specified characters. The relative order within the set of these unspecified characters is unchanged from the native collating sequence.

4) If the THROUGH phrase is specified, the set of contiguous characters in the native character set beginning with the character specified by the value of *literal-1*, and ending with the characters specified by the value of *literal-2*, is assigned a successive ascending position in the collating sequence being specified. In addition, the set of contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.

5) If the ALSO phrase is specified, the characters of the native character set specified by the value of *literal-1* and *literal-3* are assigned to the same ordinal position in the collating sequence being specified or in the character code set that is used to represent the data, and if *alphabet-name-1* is referenced in a SYMBOLIC CHARACTERS clause, only *literal-1* is used to represent the character in the native character set.

(4) When specified as literals in the SPECIAL-NAMES paragraph, the figurative constants HIGH-VALUE and LOW-VALUE are associated with those characters having the highest and lowest positions, respectively, in the native collating sequence.

(5) If the IN phrase is not specified, *symbolic-character-1* represents the character whose ordinal position in the native character set is specified by *integer-1*. If the IN phrase is specified, *integer-1* specifies the ordinal position of the character that is represented in the character set name by *alphabet-name-2*.

(6) The internal representation of *symbolic-character-1* is the internal representation of the character that is represented in the native character set.

(7) The CLASS clause provides a means for relating a name to the specified set of characters listed in that clause. *Class-name* can be referenced only in a class-condition. The characters specified by the values of the literals in this clause define the exclusive set of characters of which this class-name consists.

The value of each literal specifies:

a. The ordinal number of a character within the native character set, if the literal is numeric. This value must not exceed the value which represents the number of characters in the native character set.

b. The actual character within the native character set, if the literal is nonnumeric. If the value of the nonnumeric literal contains multiple characters, each character in the literal is included in the set of characters identified by *class-name-1*.

If the THROUGH phrase is specified, the contiguous characters in the native character set beginning with the character specified by the value *literal-4*, and ending with the character specified by the value of *literal-5*, are included in the set of characters identified by *class-name-1*. In addition, the contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.

(8) *Literal-6* which appears in the CURRENCY SIGN clause is used in the PICTURE clause to represent the currency symbol. The literal must be nonnumeric and is limited to a single character. It may be any character from the computer's character set except one of the following:

a. digits 0 through 9;

b. alphabetic characters consisting of the uppercase letters A, B, C, D, P, R, S, V, X, Z; the lowercase letters a through z; or the space;

c. Special characters * + - , . ; ( ) " = /

If this clause is not present, only the currency sign defined in the COBOL character set ($) may be used as the currency symbol in the PICTURE clause.

(9) The clause DECIMAL-POINT IS COMMA means that the functions of comma and period are exchanged in the character-string of the PICTURE clause and in numeric literals.

D.3.7 General Rules (***VXCOBOL***)

(1) The DEVICE clause (literal-1 IS mnemonic-name-1) is for documentation purposes only.

(2) The CHANNEL clause declares a line printer control channel. You can use channel names in the ADVANCING clause of a WRITE statement to format printed forms.

(3) The on status and/or off status of a switch literal may be associated with condition-names. The status of that switch may be interrogated by testing these condition-names.

(4) The ALPHABET clause provides a means for relating a name to a specified character code set and/or collating sequence.

a. If the ASCII, NATIVE, or STANDARD-1 phrase is specified, the character code set or collating sequence identified is that defined in the ANSI X3.4-1977, Code for Information Interchange. Each character of the standard character set is associated with its corresponding character of the native character set. APPENDIX I, on page 844, provides a copy of the ASCII character set.

b. If the EBCDIC phrase is specified, the EBCDIC character code set or collating sequence is used. APPENDIX J, on page 846, provides a copy of the EBCDIC character set.

c. The collating sequence identified is that defined according to the following rules:

1) The value of each literal specifies:

a) The ordinal number of a character within the native character set, if the literal is numeric. This value must not exceed the value which represents the number of characters in the native character set.

b) The actual character within the native character set, if the literal is nonnumeric. If the value of the nonnumeric literal contains multiple characters, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified.

2) The order in which literals appear in the ALPHABET clause specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified.

3) Any characters within the native collating sequence, which are not explicitly specified in the literal phrase, assume a position, in the collating sequence being specified, greater than any of the explicitly specified characters. The relative order within the set of these unspecified characters is unchanged from the native collating sequence.

4) If the THROUGH phrase is specified, the set of contiguous characters in the native character set beginning with the character specified by the value of literal-3, and ending with the characters specified by the value of literal-4, is assigned a successive ascending position in the collating sequence being specified. In addition, the set of contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.

5) If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-3 and literal-5 are assigned to the same ordinal position in the collating sequence being specified or in the character code set that is used to represent the data.

(5) When specified as literals in the SPECIAL-NAMES paragraph, the figurative constants HIGH-VALUE and LOW-VALUE are associated with those characters having the highest and lowest positions, respectively, in the native collating sequence.

(6) *Literal-6*, which appears in the CURRENCY SIGN clause, is used in the PICTURE clause to represent the currency symbol. The literal must be nonnumeric and is limited to a single character. It may be any character from the computer's character set except one of the following:

a. digits 0 through 9;

b. alphabetic characters consisting of the uppercase letters A, B, C, D, P, R, S, V, X, Z; the lowercase letters a through z; or the space;

c. Special characters * + - , . ; ( ) " = /

If this clause is not present, only the currency sign defined in the COBOL character set ($) may be used as the currency symbol in the PICTURE clause.

(7) The DECIMAL-POINT IS COMMA clause means that the functions of comma and period are exchanged in the character-string of the PICTURE clause and in numeric literals.

D.3.8 Examples

(1) This example shows how a program switch is defined in the SPECIAL-NAMES paragraph in the ENVIRONMENT DIVISION, and then illustrates how it is used it in the procedure division paragraph, CHECK-SECURITY.

```
SPECIAL-NAMES.
       SWITCH "MGR" ON STATUS IS MANAGER-JOB.

CHECK-SECURITY.
       IF MANAGER-JOB MOVE 9 TO WS-SECURITY-CODE.
```

EXAMPLE 2. Using a Program Switch

The next several examples show how to modify the collating sequence from the native order, for the current program.

(2) The following will cause characters to collate in the order: <space>, 1-9, a-z, null, |, ', #, ... All characters not explicitly defined will follow in their native order.

ANSI 74 and ***ANSI 85***:

```
SPECIAL-NAMES.
    ALPHABET NEW-SEQ IS " ", "1" THRU "9", "a" THRU "z".
```

VXCOBOL:

```
SPECIAL-NAMES.
    NEW-SEQ IS " ", "1" THRU "9", "a" THRU "z".
```

EXAMPLE 3. Modifying the collating sequence for a program

(3) The following leaves the native system intact, with the exception of making the underscore (95) the highest character.

ANSI 74 and ***ANSI 85***:

```
       SPECIAL-NAMES.
           ALPHABET NEW-SEQ IS 1 THRU 94, 96 THRU 126, 95.
```

VXCOBOL:

```
       SPECIAL-NAMES.
           NEW-SEQ IS 1 THRU 94, 96 THRU 126, 95.
```

EXAMPLE 4. Changing 1 character in the collating sequence

(4) The following equates the space with zero, giving them the same ordinal position in the collating sequence.

ANSI 74 and ***ANSI 85***:

```
SPECIAL-NAMES.
    ALPHABET NEW-SEQ IS " " ALSO "0".
```

VXCOBOL:

```
SPECIAL-NAMES.
    NEW-SEQ IS " " ALSO "0", "a" ALSO "A".
```

EXAMPLE 5. Making multiple characters the same in the collating sequence

(5) The following reverses the typical collating sequence for digits and uppercase alphabet characters. Note, however, that all other characters not explicitly defined will follow in their usual order.

ANSI 74 and ***ANSI 85***:

```
SPECIAL-NAMES.
    ALPHABET REVERSE-SEQ IS "9" THRU "0", "Z" THRU "A".
```

VXCOBOL:

```
SPECIAL-NAMES.
    REVERSE-SEQ IS "9" THRU "0", "Z" THRU "A".
```

EXAMPLE 6. Reversing collating sequence for digits, uppercase alphabet

E. INPUT-OUTPUT SECTION

The Input-Output Section is located in the Environment Division of a source program. The Input-Output Section deals with the information needed to control transmission and handling of data between external media and the object program. The Input-Output Section is optional in the Environment Division of a COBOL source program.

ANSI 74 and ***ANSI 85***:

```
INPUT-OUTPUT SECTION.

FILE-CONTROL.
    { file-control-entry }...

[ I-O-CONTROL.

d     [ RERUN [ ON file-name-1 ] EVERY { { [ END OF ] { REEL } } OF file-name-2 }
                                       { { UNIT }     }
                                       { integer-1 RECORDS }
                                       { integer-2 CLOCK-UNITS }
                                       { condition-name-1 }

      [ SAME [ RECORD ]                AREA FOR file-name-3 { file-name-4 }... ]...
             [ SORT   ]
             [ SORT-MERGE ]

d     [ MULTIPLE FILE TAPE CONTAINS { file-name-5 [ POSITION integer-3 ] }... ]...
  .]
```

VXCOBOL

```
INPUT-OUTPUT SECTION.

FILE-CONTROL.
    { file-control-entry }...

[ I-O-CONTROL.

      [ SAME [ RECORD ]                AREA FOR file-name-1 { file-name-2 }... ]...
             [ SORT   ]
             [ SORT-MERGE ]

d     [ MULTIPLE FILE TAPE CONTAINS { file-name-3 [ POSITION integer ] }... ]...
  .]
```

E.1. FILE-CONTROL Paragraph

E.1.1 Function

The FILE-CONTROL paragraph allows specification of file-related information.

E.1.2 General Format

```
FILE-CONTROL.
    { file-control-entry }...
```

E.2. File Control Entry

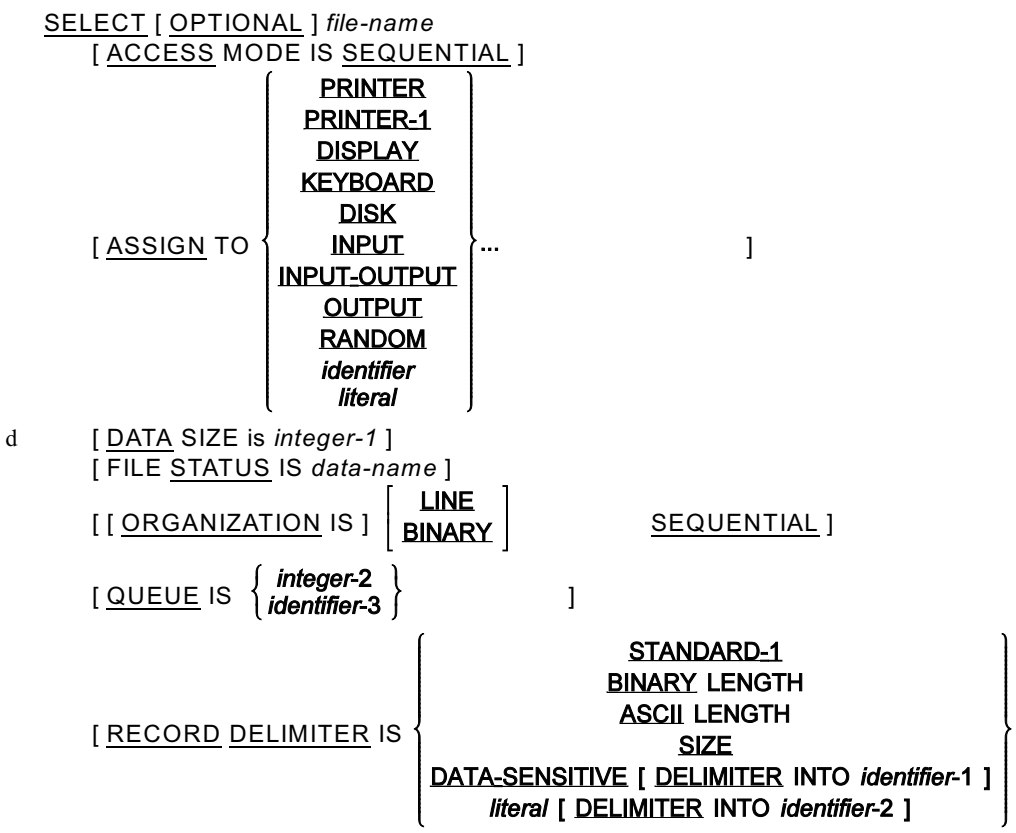
E.2.1 Function

The file control entry declares the relevant physical attributes of a sequential, relative, indexed, sort, or merge file.

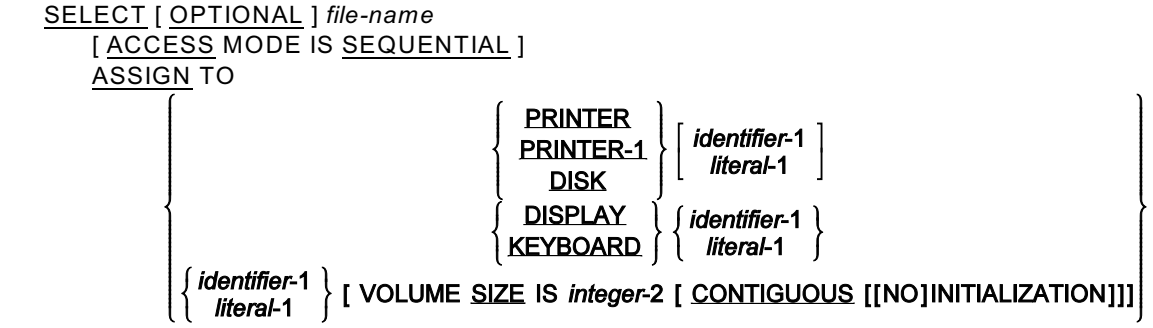
E.2.2 General Format

The clauses for each SELECT are given in alphabetical order since they are order independent.

Sequential File (*ANSI 74* and *ANSI 85*):



Sequential File (*VXCOBOL*):



d [PARITY IS { ODD / EVEN }]

d [RESERVE *integer* [AREA / AREAS]] .

Relative File (*ANSI 74* and *ANSI 85*):

SELECT [OPTIONAL] file-name

[ACCESS MODE IS { { SEQUENTIAL [RELATIVE KEY IS data-name] / { RANDOM / DYNAMIC } RELATIVE KEY IS data-name }]

[ASSIGN TO { DISK / identifier / literal }...]

d [DATA SIZE IS *integer*]

d [INDEX SIZE IS *integer*]

[DELETE IS { LOGICAL / PHYSICAL }]

[FILE STATUS IS *data-name*]

[ORGANIZATION IS] RELATIVE .

Relative File (*VXCOBOL*):

SELECT *file-name*

[ACCESS MODE IS { { SEQUENTIAL [RELATIVE KEY IS data-name] / { RANDOM / DYNAMIC } RELATIVE KEY IS data-name }]

ASSIGN TO

{ DISK { *identifier*-1 / *literal*-1 } / { *identifier*-1 / *literal*-1 } [VOLUME SIZE IS *integer*-2 [CONTIGUOUS [[NO]INITIALIZATION]]] }

[FILE STATUS IS *data-name*]

[INFOS STATUS IS *data-name*]

[ORGANIZATION IS] RELATIVE

d [RESERVE *integer* [AREA / AREAS]] .

Indexed File (*ANSI 74* and *ANSI 85*):

SELECT [OPTIONAL] *file-name*

[ACCESS MODE IS { SEQUENTIAL / RANDOM / DYNAMIC }]

[ALTERNATE RECORD KEY IS *id-1* [= *id-2* { ALSO *id*-3... / PLUS *id*-3... / [PLUS *id*-3...] OCCURS *integer* TIMES }]

[ORDER BY ALPHABETIC-UPPER]

[SUPPRESS WHEN *literal*]

```
        [ VALUES ARE { ASCENDING  }                    ]
                     { DESCENDING }
        [ WITH DUPLICATES ] ]...
    [ ASSIGN TO { DISK       }...                      ]
                { identifier }
                { literal    }
d   [ DATA SIZE IS integer ]
d   [ INDEX SIZE IS integer ]
    [ DELETE IS { LOGICAL  }                           ]
                { PHYSICAL }
    [ FILE STATUS IS data-name ]
    [ ORGANIZATION IS ] INDEXED
    RECORD KEY IS id-1 [ = id-2 PLUS { id-3 }... ]
    [ ORDER BY ALPHABETIC-UPPER ]
    [ VALUES ARE { ASCENDING  }                        ] .
                 { DESCENDING }
```

Indexed File (*VXCOBOL*):

```
SELECT file-name
    [ ACCESS MODE IS { SEQUENTIAL }                    ]
                     { RANDOM     }
                     { DYNAMIC    }
    [ ALTERNATE RECORD { KEY IS  }                    data-name
                       { KEYS ARE }
        [ KEY LENGTH IS integer ]
        [ WITH DUPLICATES ] ]...
    ASSIGN INDEX TO
    { DISK { identifier-1 }                                                    }
    {      { literal-1    }                                                    }
    { {identifier-1} [ MERIT integer][ VOLUME SIZE IS integer-2[ CONTIGUOUS [[NO]INITIALIZATION]]]  }
    { {literal-1   }                                                           }
d       [ ROOT MERIT IS integer ]
d       [ SPACE MANAGEMENT ]
d       [ TEMPORARY ]
d       [ HIERARCHICAL ]
        [ LRU          ]
d   { [ COMPRESSION ]                                  }
    { [ KEY COMPRESSION ] [ DATA COMPRESSION ]         }
d   [ DATA SIZE IS integer ]
d   [ INDEX SIZE IS integer ]
    [ FILE STATUS IS data-name ]
    [ INFOS STATUS IS data-name ]
    [ ORGANIZATION IS ] INDEXED
    RECORD { KEY IS  }                                 data-name
           { KEYS ARE }
        [ KEY LENGTH IS integer ]
d   [ RESERVE integer INDEX [ AREA  ]                  ]
                            [ AREAS ]
d   [ RESERVE integer DATA [ AREA  ]                   ] .
                           [ AREAS ]
```

Sort-Merge File (ANSI 74 and ANSI 85):

SELECT *file-name* [ASSIGN TO { DISK / identifier / literal }...] .

Sort-Merge File (*VXCOBOL*):

SELECT *file-name* ASSIGN TO { DISK / *identifier* / *literal* }... [[ORGANIZATION IS] SEQUENTIAL].

INFOS Files (*VXCOBOL*):

SELECT *file-name*
[ACCESS MODE IS { SEQUENTIAL / RANDOM / DYNAMIC }]
[ALLOW SUB-INDEX
[LEVELS IS *integer*]]
ASSIGN INDEX TO { *identifier* / *literal* } [MERIT *integer*] [VOLUME SIZE IS *integer* [CONTIGUOUS
d [[NO] INITIALIZATION]]] }...
d [TEMPORARY]
[SPACE MANAGEMENT]
[ROOT MERIT IS *integer*]
d [HIERARCHICAL / LRU]
[ASSIGN DATA TO { *identifier* / *literal* } [MERIT *integer*]
d [VOLUME SIZE IS *integer* [CONTIGUOUS [[NO] INITIALIZATION]]] }...
[SPACE MANAGEMENT]]
{ [COMPRESSION] / [KEY COMPRESSION] [DATA COMPRESSION] }
d [DATA SIZE IS *integer*]
d [INDEX SIZE IS *integer*]
[FILE STATUS IS *data-name*]
[INFOS STATUS IS *data-name*]
[ORGANIZATION IS] INDEXED
RECORD { KEY IS / KEYS ARE } { *data-name*
[KEY LENGTH IS { *identifier* / *literal* }]
[WITH DUPLICATES [OCCURRENCE IS *identifier*]] }...
d [RESERVE *integer* INDEX [AREA / AREAS]]
d [RESERVE *integer* DATA [AREA / AREAS]] .

E.2.3 Syntax Rules:

(1) The SELECT clause must be specified first in the file control entry. The clauses which follow the SELECT clause may appear in any order.

(2) Each *file-name* in the Data Division must be specified only once in the FILE-CONTROL paragraph. Each *file-name* specified in the SELECT clause must have a file description entry in the Data Division of the same program.

(3) *Literal* must be a nonnumeric literal and must not be a figurative constant.

(4) Each sort or merge file in the Data Division must be specified only once in the FILE-CONTROL paragraph. Each sort or merge file specified in the SELECT clause must have a sort-merge file description entry in the Data Division of the same program.

(5) For Sort-Merge Entry. Since file-name represents a sort or merge file, only the ASSIGN clause is permitted to follow file-name in the FILE-CONTROL paragraph.

(6) The OPTIONAL phrase is only allowed for ***ANSI 85*** and ***VXCOBOL***.

For *ANSI 74* and *ANSI 85*.

(7) The ORDER BY ALPHABETIC-UPPER phrase applies only to version 7 or greater ICISAM indexed files.

(8) The DELETE clause applies only to version 7 or greater ICISAM files.

(9) The RECORD DELIMITER clause can only be specified on SEQUENTIAL files with the RECORD IS VARYING clause in the FD. If the RECORD DELIMITER clause is absent and RECORD IS VARYING is specified, the length of the record written is determined by the DEPENDING ON variable or implied by the record definitions.

E.2.4 General Rules

(1) For ***ANSI 85***, the OPTIONAL phrase applies only to files opened in input, I-O, or extend mode. Its specification is required for files that are not necessarily present each time the object program is executed. See OPEN for more information.

For ***VXCOBOL***, the OPTIONAL phrase applies only to sequential files opened in input mode. Its specification is required for files that are not necessarily present each time the object program is executed. If you specify this clause and the file is not present, the first READ statement for the file signals an end-of-file condition.

(2) For ***VXCOBOL***, the PARITY, DATA SIZE, INDEX SIZE, INITIALIZATION, TEMPORARY, HIERARCHICAL/LRU, RESERVE, RESERVE INDEX, and RESERVE DATA clauses are used for documentation purposes only.

For relative, indexed, and INFOS files:

(3) The native character set is assumed for data on the external media.

(4) For an indexed file or INFOS, the collating sequence associated with the native character set is assumed. This is the sequence of values of a given key of reference used to process the file sequentially.

(5) The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium referenced by the specified name or literal.

(6) For ***ANSI 74*** and ***ANSI 85***, the INDEX SIZE and DATA SIZE clauses are used for documentation purposes only.

E 3. ACCESS MODE Clause

E.3.1 Function

The ACCESS MODE clause specifies the order in which records are to be accessed in the file.

E.3.2 General Format

Sequential File:

ACCESS MODE IS SEQUENTIAL

Relative File:

```
                  { SEQUENTIAL [RELATIVE KEY IS data-name-1] }
ACCESS MODE IS    { { RANDOM  } RELATIVE KEY IS data-name-1 }
                  { { DYNAMIC }                             }
```

Indexed File (all **ICOBOL** dialects) **and INFOS File** (***VXCOBOL***):

```
                  { SEQUENTIAL }
ACCESS MODE IS    { RANDOM     }
                  { DYNAMIC    }
```

E.3.3 Syntax Rules:

(1) ACCESS MODE is DYNAMIC or RANDOM can only be used for relative, indexed, or INFOS files.

For relative files:

(1) *Data-name-1* may be qualified.

(2) *Data-name-1* must reference an unsigned integer data item whose description does not contain the PICTURE symbol `P'.

(3) If a relative file is referenced by a START statement, the RELATIVE KEY phrase within the ACCESS MODE clause must be specified for that file.

(4) For **ANSI 74** and **ANSI 85**, *data-name-1* must not be defined in a record description entry associated with that file-name.

E.3.4 General Rules

(1) If the ACCESS MODE clause is not specified, sequential access is assumed.

(2) Records in the file are accessed in the sequence dictated by the file organization. For sequential files, this sequence is specified by predecessor-successor record relationships established by the execution of WRITE statements when the file is created or extended. For relative files, this sequence is ascending relative record number of existing records in the file. For indexed or INFOS files, this sequence is ascending within a given key of reference according to the collating sequence of the file.

For relative files:

(3) If the access mode is random, the value of the relative key data item for relative files indicates the record to be accessed.

(4) If the access mode is dynamic, records in the file may be accessed sequentially and/or randomly.

(5) All records stored in a relative file are uniquely identified by relative record numbers. The relative record number of a given record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers 2, 3, ...

(6) The data item specified by *data-name-1* is used to communicate a relative record number between the user and the file system.

(7) The relative key data item associated with the execution of an input-output statement is the data item referenced by *data-name-1* in the ACCESS MODE clause.

For indexed files:

(8) If the access mode is random, the value of a record key data item for indexed files indicates the record to be accessed.

For INFOS files (***VXCOBOL***):

(9) If the access mode is random, the value of a series of record key data items (with their associated occurrence values, if any) and a relative motion specifier indicates the record to be accessed.

(10) If the access mode is sequential, the sequence of access is in ascending order by keys within a given index or subindex. The subindex may be changed with a relative motion specifier.

E.4. ALLOW SUB-INDEX and LEVELS Clauses (***VXCOBOL***)

E.4.1 Function

The ALLOW SUB-INDEX and LEVELS clauses must be used to define the maximum number of subindex levels permitted in an INFOS file.

E.4.2 General Format

ALLOW SUB-INDEX [LEVELS IS *integer*]

E.4.3 Syntax Rules

(1) Integer is a positive integer between 1 and 8 that specifies the maximum number of index and subindex levels the file can have.

E.4.4 General Rules

(1) ALLOW SUB-INDEX must be specified for any file that already has subindexing or will allow subindexing.

(2) The LEVELS clause indicates the expected maximum number of index and subindex levels that the file will have. If you do not specify this clause on file creation, the maximum number of levels will default to the number of keys in the RECORD KEY clause.

(3) The maximum number of levels for a U/FOS file is 8.

E.5. ALTERNATE RECORD KEY Clause (***ANSI 74*** and ***ANSI 85***)

E.5.1 Function

The ALTERNATE RECORD KEY clause specifies an alternate record key access path to the records in an indexed file. The ALSO, ORDER BY ALPHABETIC-UPPER, PLUS, SUPPRESS, and VALUES phrases are extensions to ANSI COBOL.

E.5.2 General Format

```
ALTERNATE RECORD KEY IS id-1 [ = id-2 { ALSO id-3...                          }
                                      { PLUS id-3...                          }
                                      { [ PLUS id-3... ] OCCURS integer TIMES }
    [ ORDER BY ALPHABETIC-UPPER ]
    [ SUPPRESS WHEN literal ]
    [ VALUES ARE { ASCENDING  } ]
                 { DESCENDING }
    [ WITH DUPLICATES ]
```

E.5.3 Syntax Rules

(0) The ALTERNATE RECORD KEY clause may occur at most 16 times for version 7 or greater ICISAM files and at most 4 times for versions less than 7.

(1) The phrases following the ALTERNATE RECORD KEY clauses (ORDER BY, SUPPRESS WHEN, VALUES ARE, and WITH DUPLICATES) may be specified in any order.

(2) *id-1* must not reference an item whose left-most character position corresponds to the left-most character position of the primary record key or of any other alternate record key associated with this file. **NOT ENFORCED BY ICOBOL.**

(3) If *id-2* is not specified, *id-1* may be qualified and must reference a data-item of category alphanumeric within a record description entry associated with the file-name to which the ALTERNATE RECORD KEY is subordinate. *Id-1* must not reference a group item containing a variable occurrence data-item.

If *id-2* is specified, *id-1* must be a unique word within the program and is not defined elsewhere. *Id-1* may be referenced only in the KEY IS phrases of the READ or START statements.

(4) Each instance of *id-2* or *id-3* must reference a data-item of category alphanumeric within a record description entry associated with the file-name to which the ALTERNATE RECORD KEY is subordinate. No instance of *id-2* or *id-3* may reference a group item which contains a variable occurrence data-item

(5) If the ALSO phrase is specified, *id-3* may be specified up to six times. If the ALSO phrase is not specified (i.e., the PLUS case), *id-3* may be specified up to three times.

(6) If *id-2* is not specified, the length of *id-1* may not exceed 255 bytes for version 7 or greater ICISAM files and may not exceed 100 bytes for indexed files with versions less than 7.

If *id-2* is specified, each instance of *id-2* and *id-3* must have a length that does not exceed 255 bytes. If the ALSO phrase is specified, each *id-3* must have the same length as *id-2*. If the ALSO phrase is not specified, the sum of the lengths of *id-2* and each *id-3* must not exceed 255 bytes.

(7) If the OCCURS phrase is specified, *integer* must be in the range from 1 to 31. *id-2* and each *id-3* must each be in the same record definition. Additionally, they must be subordinate to a common OCCURS phrase which is defined as occurring integer times. Each of the identifiers must be specified without a subscript.

If the OCCURS phrase is not specified, none of the identifiers may have an OCCURS phrase in their description or be subordinate to an item which has an OCCURS phrase its definition.

(8) Within the record definition the byte positions of *id-2* and each *id-3* must be disjoint, i.e., they may not overlap.

(9) If the SUPPRESS WHEN phrase is specified, *lit* may be either a single character alphanumeric literal or a figurative constant.

(10) If the index files contains variable length records, each alternate record key must be contained within the first *x* character positions of the record where *x* equals the minimum record size for the file.

E.5.4 General Rules

(1) The ALTERNATE RECORD KEY clause specifies an alternate record key for the file with which this clause is associated. It may specify that one or more key values to be entered into the associated index for each record.

The alternate key may consist of a single data-item (*id-1* with no additional phrases). It may also be a composite key (identified by the key name *id-1*) defined as a root key (*id-2*) plus one or more key suffixes (*id-3*). The value of a composite alternate key is determined by appending the values of the root key and each key suffix together in the order in which they appear in the ALTERNATE RECORD KEY clause.

Multiple key values (inversions) may be entered into the index for a given alternate key in two ways:

a. The ALSO phrase may be specified. In this case, the key-name *id-1* represents an alternate record key which supports multiple key values. A value is entered into the index for *id-2* and each instance of *id-3*. This allows for scattered fields in the record to be entered into the index as key values.

b. The OCCURS phrase may be specified. In this case, the key-name *id-1* represents an alternate record key which supports multiple key values in a tabularized form. For each occurrence of *id-2* in the record definition, optionally suffixed by occurrences of *id-3*, a key value is entered into the index.

(2) The data description and relative location within a record of *id-1* (if it is used alone) and of *id-2* and each *id-3* must be the same as that used when the file was created.

(3) If the file has more than one record description entry, *id-1* (if it is used alone) or *id-2* and each *id-3* need only be described in one of these record description entries except when the OCCURS phrase is present. If the OCCURS phrase is present *id-2* and each *id-3* must be in the same record description entry. In all cases, the identical character positions referenced by *id-1* (if it is used alone), *id-2*, and each *id-3* that appear in one record description are implicitly referenced as keys for all other record description entries of that file.

(4) The ORDER BY ALPHABETIC-UPPER phrase applies to version 7 or greater ICISAM files. It specifies that all values for this alternate key are entered into the index as uppercase only. Lookups for this key path will be performed in uppercase. The effect is that the keys on this key path are processed in a case insensitive manner. If ORDER BY ALPHABETIC-UPPER is not present, then key values are entered and looked up as they appear in the record.

(5) The SUPPRESS WHEN lit phrase specifies that when all characters of a key value are equal to the character specified by lit, that key value should not be entered into the index. This phrase applies to version 7 or greater ICISAM files.

(6) The VALUES ARE phrase is used to specify the order in which key values are entered into the index. If the ASCENDING phrase is specified, key values are entered in ascending order. That is, key values appear with increasing values. If the DESCENDING phrase is specified, key values are entered in descending order. That is,

key values appear with decreasing values -- in reverse sequential order. If the VALUES ARE phrase is not present, VALUES ARE ASCENDING is implied. This phrase applies to version 7 or greater ICISAM files.

(7) The WITH DUPLICATES phrase specifies that the value or values of the associated ALTERNATE RECORD KEY may be duplicated within any of the records in the file and within the record itself if multiple key values are specified. If the WITH DUPLICATES phrase is not specified, the value or values of the associated alternate key must not be duplicated among any of the records in the file or within the record itself if multiple key values are specified. If the phrase is not present, duplicate key values are not allowed. When using versions 5 or 6 indexed files, the duplicates option must be set or a runtime error (9A) will occur. Version 7 indexed files observe the duplicates option correctly.

(8) Alternate record keys are sorted based on the following criteria:

a. ascending root segment position of *id-1* (if it is a data-item) or by *id-2* if it is present.

b. ascending root segment length of *id-1* (if it is a data-item) or by *id-2* if it is present.

c. absence of ALSO keys and, if present, ascending number of ALSO and ascending ALSO's position.

d. absence of suffixes and, if present, ascending number of suffixes, ascending suffix position, and ascending suffix length.

e. absence of OCCURS and, if present, ascending number of OCCURS and ascending occurs span.

f. absence of duplicates allowed.

g. absence of descending order.

h. absence of uppercase conversion.

i. absence of SUPPRESS when value and, if present, ascending suppress when value.

(9) If the associated file connector is an external file connector, every file control entry in the run unit which is associated with that file connector must specify the same data description entry for *data-name-1*, the same relative location within the associated record, the same number of alternate record keys, and the same DUPLICATES phrase.

E.6. ALTERNATE RECORD KEY Clause (***VXCOBOL***)

E.6.1 Function

The ALTERNATE RECORD KEY clause specifies an alternate record key access path to the records in an indexed file.

E.6.2 General Format

```
ALTERNATE RECORD { KEY IS  }                    data-name
                 { KEYS ARE }

    [ KEY LENGTH IS integer ]
    [ WITH DUPLICATES ]
```

E.6.3 Syntax Rules

(1) Data-name may be qualified.

(2) Data-name must be defined as a data item of the category alphanumeric within a record description entry associated with the file-name to which the ALTERNATE RECORD KEY clause is subordinate. *Data-name* must not reference a group item that contains a variable occurrence data-item.

(3) Data-name must not reference an item whose left-most character position corresponds to the left-most character position of the primary record key or of any other alternate record key associated with this file. **NOT ENFORCED BY ICOBOL.**

(4) *integer* must be exactly the length of the item referenced by *data-name*.

(5) If the index file contains variable length records, *data-name* must be contained within the maximum record size number of characters. If *data-name* is not contained within the specified minimum record size, the minimum record size will be adjusted upward to contain *data-name*.

E.6.4 General Rules

(1) An ALTERNATE RECORD KEY clause specifies an alternate record key for the file with which this clause is associated. The ALTERNATE RECORD KEY clause may be specified no more than 16 times.

(2) The data description of data-name as well as its relative location within a record must be the same as that used when the file was created. The number of alternate record keys for the file must also be the same as that used when the file was created.

(3) The DUPLICATES phrase specifies that the value of the associated alternate record key may be duplicated within any of the records in the file. If the DUPLICATES phrase is not specified, the value of the associated alternate record key must not be duplicated among any of the records in the file.

(4) If the file has more than one record description entry, data-name need only be described in one of these record description entries. The identical character positions referenced by data-name in any one record description entry are implicitly referenced in keys for all other record description entries of that file.

(5) Alternate keys are sorted by their leftmost character position. Under **ICOBOL**, if multiple alternate keys start at the same position, they are sorted in ascending order by length (smallest to largest).

(6) If the associated file connector is an external file connector, every file control entry in the run unit which is associated with that file connector must specify the same data description entry for *data-name*, the same relative location within the associated record, the same number of alternate record keys, and the same DUPLICATES phrase.

E.7. ASSIGN Clause

E.7.1 Function

The ASSIGN clause specifies the association of the file referenced to a defined storage medium.

E.7.2 General Format (*ANSI 74* and *ANSI 85*)

Sequential File:

```
ASSIGN TO { PRINTER
          | PRINTER-1
          | DISPLAY
          | KEYBOARD
          | DISK
          | INPUT
          | INPUT-OUTPUT
          | OUTPUT
          | RANDOM
          | identifier-1
          | literal-1 } ...
```

Relative, Indexed, and Sort-Merge Files:

```
ASSIGN TO { DISK
          | identifier-1
          | literal-1 } ...
```

E.7.3 General Format (*VXCOBOL*)

Sequential:

```
ASSIGN TO
{ { PRINTER   } [ identifier-1 ]
{ { PRINTER-1 } [ literal-1    ]
{ { DISK      }
{
{ { DISPLAY   } { identifier-1 }
{ { KEYBOARD  } { literal-1    }
{
{ { identifier-1 } [ VOLUME SIZE IS integer-2 [ CONTIGUOUS [[NO]INITIALIZATION]]]
{ { literal-1    }
```

Relative:

```
ASSIGN TO
{ DISK { identifier-1 }
{      { literal-1    }
{
{ { identifier-1 } [VOLUME SIZE IS integer-2[ CONTIGUOUS [[NO]INITIALIZATION]]]
{ { literal-1    }
```

Indexed:

```
ASSIGN INDEX TO
    { DISK { identifier-1 } }
    { { identifier-1 } [VOLUME SIZE IS integer-2[ CONTIGUOUS [[NO]INITIALIZATION]]] }
    { { literal-1    } }
d   [ ROOT MERIT IS integer ] [ SPACE MANAGEMENT ] [ TEMPORARY ]
d   [ HIERARCHICAL ]
    [ LRU          ]
```

Sort-Merge File:

```
ASSIGN TO { DISK       }...
          { identifier }
          { literal    }
```

INFOS Files:

```
    ASSIGN INDEX TO { identifier } [ MERIT integer ] [ VOLUME SIZE IS integer [ CONTIGUOUS
                    { literal    }
d       [ [ NO ] INITIALIZATION] ] ] }...
d       [ TEMPORARY ]
        [ SPACE MANAGEMENT ]
        [ ROOT MERIT IS integer ]
d       [ HIERARCHICAL ]
        [ LRU          ]

    [ ASSIGN DATA TO { identifier } [ MERIT integer ]
                     { literal    }
d       [ VOLUME SIZE IS integer [ CONTIGUOUS [ [ NO ] INITIALIZATION] ] ] }...
        [ SPACE MANAGEMENT ] ]
```

E.7.4 Syntax Rules

(1) Only one storage medium (PRINTER, PRINTER-1, DISPLAY, KEYBOARD, DISK, INPUT, INPUT-OUTPUT, OUTPUT, or RANDOM) may be specified.

(2) Only one external filename specifier (*identifier-1* or *literal-1*) may be specified except for INFOS files.

For ***VXCOBOL***:

(3) *Integer-1* and *integer-4* are positive integer literals between 1 and 32 that specify the merit factor of a volume. If not specified the merit factor is 1.

(4) *Integer-2* and *integer-5* are positive integer literals that specify a number of 512-byte blocks.

(5) *Integer-3* is a positive integer literal between 1 and 32 that specifies which volume priority has the highest level root node.

E.7.5 General Rules

(1) If no storage medium is specified DISK is assumed.

(2) If no *identifier-1* or *literal-1* is specified, the default external filename is defined below for sequential files.

Device	Default Filename ***VXCOBOL***	Default Filename **ANSI 74/85**
PRINTER	@LPT	$LPT
PRINTER-1	@LPT1	$LPT1
DISPLAY	@CONSOLE	$TTO
KEYBOARD	@CONSOLE	$TTI
DISK, INPUT, INPUT-OUTPUT, OUTPUT, RANDOM	Characters of the internal filename with $ replacing -	First ten character of the internal filename with $ replacing -.

TABLE 1. Default External Filenames for Sequential Files

NOTE: The -N h compiler switch will suppress the translation of "-" to "$" in the generation of default filenames.

(3) For ***VXCOBOL***, relative, indexed, and INFOS files have no default external filename. For **ANSI 74** and **ANSI 85**, an external filename is generated from the internal name by selecting the first 10 characters of the internal name and replacing - with $.

(4) Only sequential files may be ASSIGN'ed to PRINTER, PRINTER-1, KEYBOARD, DISPLAY, INPUT, INPUT-OUTPUT, or OUTPUT.

(5) For INFOS files, VOLUME SIZE sets the maximum volume size. It is ignored on sequential, relative, and indexed files.

(6) For sequential files, RANDOM is equivalent to DISK.

(7) When INPUT is specified, the assigned storage medium is DISK and the compiler restricts the file usage to only those operations that are compatible with an input-only usage: OPEN INPUT, READ, and as a USING file in a SORT-MERGE operation.

(8) When OUTPUT is specified, the assigned storage medium is DISK, and the compiler restricts the file usage to only those operations that are compatible with an output-only usage: OPEN OUTPUT or EXTEND, WRITE, and as a GIVING file in a SORT-MERGE operation.

(9) When INPUT-OUTPUT is specified, the assigned storage medium is DISK with no further restrictions.

E.8. COMPRESSION Clauses (*VXCOBOL*)

E.8.1 Function

The COMPRESSION clauses enable INFOS space saving.

E.8.2 General Format

```
{ [ KEY COMPRESSION ] [ DATA COMPRESSION ] }
{                [ COMPRESSION ]           }
```

E.8.3 General Rules

(1) KEY COMPRESSION enables space saving in an INFOS indexed file.

(2) DATA COMPRESSION enables space saving in an INFOS data file.

(3) COMPRESSION enables both KEY COMPRESSION and DATA COMPRESSION.

(4) U/FOS ignores the KEY COMPRESSION clause and the implied key compression in the COMPRESSION clause.

E.9. DELETE LOGICAL/PHYSICAL Clause (***ANSI 74*** and ***ANSI 85***)

E.9.1 Function

The DELETE LOGICAL/PHYSICAL clause specifies whether DELETE record operations should be either logical (thus allowing the record to be UNDELETE'd) or physical (allowing reuse of the record area for a new record and thus NOT allowing an UNDELETE). DELETE LOGICAL/PHYSICAL is an extension to ANSI COBOL.

E.9.2 General Format

DELETE IS { LOGICAL / PHYSICAL }

E.9.3 Syntax Rules

(1) The DELETE clause applies to version 7 or greater ICISAM (relative and indexed) files.

E.9.4 General Rules

(1) The DELETE clause specifies the value of the "delete-is-physical" attribute in version 7 ICISAM files and controls the default behavior for record deletions. If the DELETE IS LOGICAL clause is specified, a deleted record is retained in the file and is simply flagged as being deleted. It may be undeleted. If DELETE IS PHYSICAL is specified, the space used by the deleted record is made available for reuse. The record may not be undeleted. The default behavior may be overridden by including the LOGICAL or PHYSICAL phrases on the DELETE statement.

(2) If this clause is specified and an existing version 7 file is opened, the value of the specification must agree with the value of the file's "delete-is-physical" attribute.

(3) If this clause is omitted and a version 7 file is created, the default is DELETE IS LOGICAL.

E.10. FILE STATUS Clause

E.10.1 Function

The FILE STATUS clause specifies a data item which contains the status of an input-output operation.

E.10.2 General Format

FILE STATUS IS *data-name*

E.10.3 Syntax Rules

(1) *Data-name* may be qualified.

(2) *Data-name* must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section.

E.10.4 General Rules

(1) If the FILE STATUS clause is specified, the data item referenced by *data-name* is always updated to contain the value of the I-O status whenever the I-O status is updated. This value indicates the status of execution of the statement. See I-O Status, page 237 or the APPENDIX on FILE STATUS codes for the values.

(2) The data item referenced by *data-name* which is updated during the execution of an input-output statement is the one specified in the file control entry associated with that statement.

For ***VXCOBOL***

(3) If either FILE STATUS or INFOS STATUS clause is specified for a file, then even if there is no declaratives to trap an exception, the program proceeds. Only if there is neither a FILE STATUS nor INFOS STATUS nor a declaratives entry will the program abort with a Fatal Error.

(4) FILE STATUS and INFOS STATUS are updated at the same time.

(5) INFOS STATUS values are either an octal number representing an INFOS or AOS/VS compatible error code or a string beginning with an ‘X’ followed by a decimal number representing an exception status code.

E.11. INDEX SIZE, DATA SIZE Clauses

E.11.1 Function

These clauses are comment fields. They allow older programs with these clauses to compile without errors. INDEX SIZE and DATA SIZE are extensions to ANSI COBOL.

E.11.2 General Format

DATA SIZE is *integer*
INDEX SIZE is *integer*

E.11.3 General Rules

(1) The INDEX clause can only be used for relative, indexed, and INFOS files.

(2) The DATA SIZE and INDEX SIZE clauses are used for documentation purposes only.

E.12. INFOS STATUS Clause (***VXCOBOL***)

E.12.1 Function

The INFOS STATUS clause specifies a data item which contains the INFOS status of an input-output operation.

E.12.2 General Format

INFOS STATUS IS *data-name*

E.12.3 Syntax Rules

(1) *Data-name* may be qualified.

(2) *Data-name* must be defined in the Data Division as a four-character to eleven-character data item of the category alphanumeric and must not be defined in the File Section.

E.12.4 General Rules

(1) If the INFOS STATUS clause is specified, the data item referenced by *data-name* is always updated to contain the value of the INFOS STATUS whenever the status is updated. This value indicates the status of execution of the statement.

(2) The data item referenced by *data-name* which is updated during the execution of an input-output statement is the one specified in the file control entry associated with that statement.

(3) If either a FILE STATUS or INFOS STATUS clause is specified for a file, then even if there is no declaratives to trap an exception, the program proceeds. Only if there is neither a FILE STATUS nor INFOS STATUS nor a declaratives entry will the program abort with a Fatal Error.

(4) FILE STATUS and INFOS STATUS are updated are the same time.

(5) INFOS STATUS values are either an octal number representing an INFOS or AOS/VS compatible error code or a string beginning with an 'X' followed by a decimal number representing an exception status code.

E.13. ORGANIZATION Clause

E.13.1 Function

The ORGANIZATION clause specifies the type (sequential, relative, or indexed) of organization as the logical structure of a file and, for sequential files, may also imply information about the record format.

E.13.2 General Format

E.7.2 General Format ()

ANSI 74 and ***ANSI 85*** **Sequential File:**

[ORGANIZATION IS] [LINE / BINARY] SEQUENTIAL

Others:

[ORGANIZATION IS] { SEQUENTIAL / RELATIVE / INDEXED }

E.13.3 General Rules

(1) The ORGANIZATION IS SEQUENTIAL clause specifies sequential organization as the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

(2) Sequential organization is a permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

(3) The ORGANIZATION IS LINE SEQUENTIAL clause specifies sequential organization and it specifies that the record format is data sensitive. If the ORGANIZATION IS LINE SEQUENTIAL clause is specified, the RECORDING MODE clause of the file's FD may not be specified.

(4) The ORGANIZATION IS BINARY SEQUENTIAL clause specifies sequential organization and it specifies that the record format is binary and not data-sensitive. If the ORGANIZATION IS BINARY SEQUENTIAL clause is specified, the RECORDING MODE clause of the file's FD may not be specified.

(5) When the ORGANIZATION clause is not specified, sequential organization is implied (without the optional LINE or BINARY option).

(6) The ORGANIZATION IS RELATIVE clause specifies relative organization as the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

(7) Relative organization is a permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

(8) The ORGANIZATION IS INDEXED clause specifies indexed organization as the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed. For ***VXCOBOL***, the file may be either an indexed file or INFOS file.

(9) Indexed organization is a permanent logical file structure in which each record is identified by the value of one or more keys within that record.

E.14. QUEUE Clause (**ANSI 74** and **ANSI 85**)

E.14.1 Function

The QUEUE clause allows the specification of a destination printer control queue for the sequential file.

E.14.2 General Format

QUEUE IS { *integer* / *identifier* }

E.14.3 Syntax Rules

(1) *integer* must be in the range 0 through 2047 inclusive. (Was 127 in pre-3.30 versions).

(2) *identifier* may be qualified, but may not be subscripted.

(3) *identifier* must be defined in the Data Division as an integer data-item and must not be defined in the File Section.

E.14.4 General Rules

(1) If *identifier* is specified, its value must be in the range 0 through 2047 inclusive.

(2) The value specified in *identifier* us used to represent a particular printer control queue (PCQ). Zero identifies @PCQ0, one identifies @PCQ1, two identifies @PCQ2, etc. The name of the specified file will be entered into that queue.

(3) At runtime, the selected queue should correspond to a queue which is available.

E.15. RECORD DELIMITER Clause (***ANSI 74*** and ***ANSI 85***)

E.15.1 Function

The RECORD DELIMITER clause indicates the method of determining the length of a variable-length record on the external medium.

E.15.2 General Format

```
RECORD DELIMITER IS { STANDARD-1                                          }
                    { BINARY LENGTH                                       }
                    { ASCII LENGTH                                        }
                    { SIZE                                                }
                    { DATA-SENSITIVE [ DELIMITER INTO identifier-1 ]      }
                    { literal [ DELIMITER INTO identifier-2 ]             }
```

E.15.3 Syntax Rules

(1) The RECORD DELIMITER clause may be specified only for sequential files that have variable-length records. Such a file contains the RECORD IS VARYING clause in the FD.

(2) If the RECORD DELIMITER clause is absent and RECORD IS VARYING is specified, the implied RECORD DELIMITER for a file with LINE SEQUENTIAL organization is DATA-SENSITIVE, and for others is BINARY LENGTH.

(3) If the RECORD DELIMITER clause is specified, the RECORDING MODE clause of the file's FD may not be specified.

(4) *identifier-1* must be a 2-byte alphanumeric data-item, not defined in the FILE Section.

(5) *identifier-2* must be a 1-byte alphanumeric data-item not defined in the FILE Section.

E.15.4 General Rules

(1) The RECORD DELIMITER options are described below:

a. STANDARD-1 is for documentation purposes only and is processed in the same manner as SIZE.

b. BINARY LENGTH indicates the presence of a record header with the length of the record stored as a 2-byte big-endian unsigned binary value. This is the traditional **ICOBOL** format. The stored length does not include the length of the header. The BINARY LENGTH option may not be specified if the organization is LINE SEQUENTIAL.

c. ASCII LENGTH indicates the presence of a record header with the length of the record stored as 4 ASCII digits. This is the traditional ***VXCOBOL*** and AOS/VS format. The stored length includes the 4 bytes occupied by the header. The ASCII LENGTH option may not be specified if the organization is LINE SEQUENTIAL.

d. SIZE indicates that the size of the record is determined completely by the record length requested. The file itself has no underlying structure and is simply a stream of bytes. If RECORD DELIMITER IS SIZE is specified, then the RECORD IS VARYING clause must include a DEPENDING ON id from which the record's size is obtained for both read and write operations. The SIZE option may not be specified if the organization is LINE SEQUENTIAL.

e. DATA-SENSITIVE indicates that the size of the record is determined by the presence of a delimiter from the set NL, CR, FF, NUL and the CR-NL pair. On WRITE operations, the length of the record is the minimum of that which is explicitly specified in the RECORD IS VARYING clause and the size determined due to the presence of a delimiter within the record itself. If a delimiter is in the record, it is emitted on the WRITE. Otherwise, the standard delimiter for the operating system is emitted, i.e. NL **on UNIX** and the CR-NL pair **on Windows**. For READ operations with the ASSIGN TO KEYBOARD phrase, the delimiter is included in the record area. If the DELIMITER INTO phrase is present, the delimiter is stored in the identifier. (The delimiter will be stored with a LOW-VALUE as its second character if it is any delimiter other than the CR-NL pair.) The DATA-SENSITIVE option may not be specified if the organization is BINARY SEQUENTIAL.

f. *Literal* is an alphanumeric literal in which each character serves as a record delimiter. On WRITE operations, the length of the record is the minimum of that which is explicitly specified in the RECORD IS VARYING clause and the size determined due to the presence of a delimiter within the record itself. If a delimiter is in the record it is emitted on the WRITE. Otherwise, the character from the literal with the lowest ASCII value is emitted as the record delimiter. For READ operations with the ASSIGN TO KEYBOARD phrase, the delimiter is included in the record area in other cases it is not. If the DELIMITER INTO phrase is present, the delimiter is stored in the identifier. The *Literal* option may not be specified if the organization is BINARY SEQUENTIAL.

(2) At the time of a successful execution of an OPEN statement, the record delimiter is the one specified in the RECORD DELIMITER clause in the file control entry associated with the file-name specified in the OPEN statement.

E.16. RECORD KEY Clause

E.16.1 Function

The RECORD KEY clause specifies the primary record key access path to the records in an indexed file. For an INFOS file, it specifies the valid indexes for this file. The ORDER BY ALPHABETIC-UPPER, PLUS, VALUES ARE, KEY LENGTH, and OCCURRENCE phrases are extensions to ANSI COBOL.

E.16.2 General Format (***ANSI 74*** and ***ANSI 85***)

```
RECORD KEY IS id-1 [ = id-2 PLUS { id-3 }... ]
    [ ORDER BY ALPHABETIC-UPPER ]
    [ VALUES ARE { ASCENDING  }                ]
                 { DESCENDING }
```

E.16.3 General Format (***VXCOBOL***)

Indexed:

```
RECORD { KEY IS  }    data-name-1 [ KEY LENGTH IS literal-1 ]
       { KEYS ARE }
```

INFOS:

```
RECORD { KEY IS  }    { data-name-1 [ KEY { LENGTH IS } { identifier-1 } ]
       { KEYS ARE }                       { literal-1 }                    ]
    [ WITH DUPLICATES [ OCCURRENCE IS identifier-2 ] ] }...
```

E.16.4 Syntax Rules (***ANSI 74*** and ***ANSI 85***)

(1) The phrases following the RECORD KEY clause (ORDER BY and VALUES ARE) may be specified in any order.

(2) If *id-2* is not specified, *id-1* may be qualified and must reference a data-item of category alphanumeric within a record description entry associated with the file-name to which the RECORD KEY is subordinate. *Id-1* must not reference a group item which contains a variable occurrence data item.

If *id-2* is specified, *id-1* must be a unique word within the program and is not defined elsewhere. *Id-1* may be referenced only in the KEY IS phrases of the READ or START statements.

(3) Each instance of *id-2* or *id-3* must reference a data-item of category alphanumeric within a record description entry associated with the file-name to which the RECORD KEY is subordinate. No occurrence of *id-2* or *id-3* may reference a group item which contains a variable occurrence data item.

(4) If *id-2* is not specified, the length of *id-1* may not exceed 255 bytes for version 7 indexed files and may not exceed 100 bytes for indexed files with versions less than 7.

If *id-2* is specified, each instance of *id-2* and *id-3* must have a length that does not exceed 255 bytes. The sum of the lengths of *id-2* and each *id-3* must not exceed 255 bytes.

(5) Within the record definition the byte positions of *id-2* and each *id-3* must be disjoint, i.e., they may not overlap.

(6) *id-3* may be specified at most three(3) times.

(7) If the indexed file contains variable length records, *id-1* or all occurrences of *id-2* and *id-3* must be contained in the first x character positions of the record where x equals the minimum record size specified for the file.

E.16.5 Syntax Rules (***VXCOBOL***)

(1) *Data-name-1* may be qualified.

(2) For indexed files, *data-name-1* must reference a data item of the category alphanumeric within a record description entry associated with the file-name to which the RECORD KEY clause is subordinate. *Data-name-1* must not reference a group item that contains a variable occurrence data item.

(3) *Identifier-1* or *literal-1* specifies the length of the associated key. *Identifier-1* must be an unsigned integer data item and *literal-1* must be a positive integer literal. If neither is specified, the key length defaults to be the length of the item referenced by *data-name-1*. When used with an indexed file, *literal-1* must be exactly equal to the length of the item referenced by *data-name-1*.

(4) *Identifier-2* is an unsigned integer or alphanumeric data item that receives an occurrence number. It can hold up to 10 digits (PIC 9(10)). It must be defined in Working-Storage.

(5) If the indexed file contains variable length records, *data-name-1* must be contained within the maximum record size number of characters. If *data-name-1* is not contained within the specified minimum record size, the minimum record size will be adjusted upward to contain *data-name-1*.

E.16.5 General Rules (***ANSI 74*** and ***ANSI 85***)

(1) The RECORD KEY clause specifies the primary key record key for the file with which this clause is associated. The values of the primary key must be unique among all records of the file. The record key may consist of a single data-item (*id-1* with no additional phrases). It may also be a composite key (identified by the key name *id-1*) defined as a root key (*id-2*) plus one or more key suffixes (*id-3*). The value of a composite primary key is determined by appending the values of the root key and each key suffix together in the order in which they appear in the RECORD KEY clause.

(2) The data description and relative location within a record of *id-1* (if it is used alone) and of *id-2* and each *id-3* must be the same as that used when the file was created.

(3) If the file has more than one record description entry, *id-1* (if it is used alone) or *id-2* and each *id-3* need only be described in one of these record description entries. In all cases, the identical character positions referenced by *id-1* (if it is used alone), *id-2*, and each *id-3* that appear in one record description are implicitly referenced as keys for all other record description entries of that file.

(4) The ORDER BY ALPHABETIC-UPPER phrase applies to version 7 or greater ICISAM files. It specifies that all values for this alternate key are entered into the index as uppercase only. Lookups for this key path will be performed in uppercase. The effect is that the keys on this key path are processed in a case insensitive manner. If ORDER BY ALPHABETIC-UPPER is not present, then key values are entered and looked up as they appear in the record.

(5) The VALUES ARE phrase is used to specify the order in which key values are entered into the index. If the ASCENDING phrase is specified, key values are entered in ascending order. That is, key values appear with increasing values. If the DESCENDING phrase is specified, key values are entered in descending order. That is, key values appear with decreasing values -- in reverse sequential order. If the VALUES ARE phrase is not present, VALUES ARE ASCENDING is implied. This phrase applies to version 7 or greater ICISAM files.

(6) If the associated file connector is an external file connector, all file description entries in the run unit which are associated with that file connector must specify the same data description entry for *data-name-1* with the same relative location within the associated record.

E.16.7 General Rules (***VXCOBOL***)

(1) For indexed, the RECORD KEY clause specifies the primary record key for the file with which this clause is associated. The values of the primary record key must be unique among records of the file. For INFOS, the RECORD KEY clause specifies a list of data-items which may be used as keys. These items may occur in any order and there may be more or less keys specified that subindex levels in the file.

(2) For indexed, the data description of *data-name-1* as well as its relative location within a record must be the same as that used when the file was created.

(3) For indexed, if the file has more than one record description entry, *data-name-1* need only be described in one of these record description entries. The identical character positions referenced by *data-name-1* in any one record description entry are implicitly referenced as keys for all other record description entries of that file.

(4) For INFOS, if *identifier-1* or *literal-1* is given, then on an open of a file for output, that value is the maximum key length for the main level, on a WRITE statement the value represents the number of characters in *data-name-1* that will be stored as the value of that record's index, and when you specify a READ with a GENERIC clause, the value represents the number of characters in *data-name-1* that must be matched in order to access a given record.

(5) If you use the OCCURRENCE clause, an occurrence number is assigned for each duplicate key. With INFOS II, occurrence numbers are only unique within a subindex. With U/FOS, occurrence numbers are unique through the entire database.

(6) The occurrence number and length of a key can be obtained by issuing a RETRIEVE KEY statement.

(7) After a WRITE or a RETRIEVE statement, the occurrence number associated with the first key named in the SELECT is updated.

(8) If the associated file connector is an external file connector, all file description entries in the run unit which are associated with that file connector must specify the same data description entry for *data-name-1* with the same relative location within the associated record.

E.17. RESERVE Clause (***VXCOBOL***)

E.17.1 Function

The RESERVE clause allows the user to specify the number of input-output areas allocated.

E.17.2 General Format

Sequential and Relative:

d RESERVE *integer* [AREA / AREAS]

Indexed and INFOS:

d RESERVE *integer* DATA [AREA / AREAS]

d RESERVE *integer* INDEX [AREA / AREAS]

E.17.3 General Rules

(1) Under **ICOBOL**, the RESERVE clause is used for documentation only. **ICOBOL** buffers sequential disk files as part of its implementation.

E.18. I-O-CONTROL Paragraph

E.18.1 Function

The I-O-CONTROL paragraph specifies the memory area which is to be shared by different files.

E.18.2 General Format (***ANSI 74*** and ***ANSI 85***)

```
I-O-CONTROL.
d     [ RERUN [ ON file-name-1 ] EVERY { { [ END OF ] { REEL } } OF file-name-2 }
                                       { { UNIT }     }
                                       { integer-1 RECORDS                      }
                                       { integer-2 CLOCK-UNITS                  }
                                       { condition-name-1                       }
      [ SAME [ RECORD ]                AREA FOR file-name-1 { file-name-2 }... ]...
             [ SORT   ]
             [ SORT-MERGE ]
d     [ MULTIPLE FILE TAPE CONTAINS { file-name-5 [ POSITION integer-3 ] }... ]...
      .
```

E.18.3 General Format (***VXCOBOL***)

```
I-O-CONTROL.
      [ SAME [ RECORD ]                AREA FOR file-name-1 { file-name-2 }... ]...
             [ SORT   ]
             [ SORT-MERGE ]
d     [ MULTIPLE FILE TAPE CONTAINS { file-name-5 [ POSITION integer-3 ] }... ]...
      .
```

E.18.4 Syntax Rules

(1) The order of appearance of the clauses is immaterial.

(2) The RERUN and MULTIPLE FILE TAPE clauses are used for documentation purposes only. Both clauses are obsolete elements in Standard COBOL are to be deleted from the next revision of the standard.

E.19. SAME Clause

E.19.1 Function

The SAME clause specifies the memory area which is to be shared by different files.

E.19.2 General Format

```
SAME [ RECORD ]                    AREA FOR file-name-1 { file-name-2 }...
     [ SORT   ]
     [ SORT-MERGE ]
```

E.19.3 Syntax Rules

(1) *File-name-1* and *file-name-2* must be specified in the FILE-CONTROL paragraph of the same program.

(2) More than one SAME clause may be included in the program, subject to the following restrictions:

a. A file-name must not appear in more than one SAME AREA clause.

b. A file-name must not appear in more than one SAME RECORD AREA clause.

(3) The files referenced in the SAME AREA or SAME RECORD AREA clause need not all have the same organization or access.

(4) SORT and SORT-MERGE are equivalent.

(5) A file-name that represents a sort or merge file must not appear in the SAME clause unless the SORT, SORT-MERGE, or RECORD phrase is used, i.e. it may not appear in a SAME AREA clause.

(6) *filename-1* and *filename-2* may not reference external file connectors.

E.19.4 General Rules

(1) The SAME AREA clause is for documentation purposes only. We recommend that you remove them or make them comment lines.

(2) The SAME RECORD AREA clause specifies that two or more files referenced by *file-name-1*, *file-name-2* are to use the same memory area for processing of the current logical record. All of these files may be in the open mode at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each file open in the output mode whose file-name appears in this SAME RECORD AREA clause and of the most recently read file open in the input mode whose file-name appears an this SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the area, i.e., records are aligned on the left-most character position.

(3) If the SAME SORT AREA or SAME SORT-MERGE AREA is used, at least one of the file-names must represent a sort or merge file. The SAME SORT AREA and SAME SORT-MERGE AREA clause is for documentation purposes only. We recommend that you remove them or make them comment lines.

V. DATA DIVISION

A. General Description

The Data Division describes the data that is to be processed by the object program. The Data Division is optional in a COBOL source program.

B. Concepts

To make data as computer-independent as possible, the characteristics or properties of the data are described in relation to a standard data format rather than an equipment-oriented format. This standard data format is oriented to general data processing applications and uses the decimal system to represent numbers (regardless of the radix used by the computer) and all characters of the COBOL character set to describe nonnumeric data items.

B.1. Logical Record Concept

In order to separate the logical characteristics of data from the physical characteristics of the data storage media, separate clauses or phrases are used. The following paragraphs discuss the characteristics of files.

B.1.1 Physical Aspects of a File

The physical aspects of a file describe the data as it appears on the input or output media and includes the means by which the file can be identified.

B.1.2 Conceptual Characteristics of a File

The conceptual characteristics of a file are the explicit definition of each logical entity within the file itself. In a COBOL program, the input or output statements refer to one logical record.

A COBOL logical record is a group of related information, uniquely identifiable, and treated as a unit,

In this document, references to records mean references to logical records.

The concept of a logical record is not restricted to file data but is carried over into the definition or working storage. Thus, working storage is grouped into logical records and defined by a series of record description entries.

B.1.3 Record Concepts

The record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name, if required, followed by a series of independent clauses, as required.

B.2. Concept of Levels

A level concept is inherent in the structure of a logical record. This concept arises from the need to specify subdivision of a record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data referral.

The most basic subdivisions of a record, that is, those not further subdivided, are called elementary items; consequently, a record is said to consist of a sequence of elementary items, or the record itself may be an elementary item.

In order to refer to a set of elementary items, the elementary items ate combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into groups of two or more groups, etc. Thus, an elementary item may belong to more than one group.

B.2.1 Level-Numbers

A system of level-numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 01. Less inclusive data items are assigned higher (not necessarily successive) level-numbers not greater in value than 49. There are special level-numbers, 66, 77, and 88, which are exceptions to this rule (see below). Separate entries are written in the source program for each level-number used.

A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. All items which are immediately subordinate to a given group item must be described using identical level-numbers greater than the level-number used to describe that group item.

Three types of entries exist for which there is no true concept of level. These are:

(1) Entries that specify elementary items or groups introduced by a RENAMES clause. Entries describing items by means of RENAMES clauses for the purpose of re-grouping data items have been assigned the special level-number 66.

(2) Entries that specify noncontiguous working storage and linkage data items. Entries that specify noncontiguous data items, which are not subdivisions of other items, and are not, themselves, subdivided, have been assigned the special level-number 77.

(3) Entries that specify condition-names. Entries that specify condition-names, to be associated with particular values of a conditional variable, have been assigned the special level-number 88.

B.3. Concept of Class and Category of Data

Every elementary data item, every literal, and every identifier has a class and a category. The class and category of a data item are defined by its picture character-string, by the BLANK WHEN ZERO clause, or by its usage. The class and category of an identifier are the class and category of the unique data item referenced by that identifier, as defined in the section on identifiers on page 127. The class and category of a literal are defined in the section on literals beginning on page 43. The following table depicts the relationship of the class and categories of data items.

The class and category of a group item is alphanumeric.

(***ISQL***) The class and category of an item with usage CHARACTER is alphanumeric; and the class and category of an item with usage INTEGER, SMALLINT, or NUMERIC is numeric.

LEVEL OF ITEM	CLASS	CATEGORY
Elementary	Alphabetic Numeric Alphanumeric " " Index Date-Time** " " Interval** " Indicator**	Alphabetic Numeric Numeric edited Alphanumeric edited Alphanumeric Index Date** Time** Timestamp** Year-to-Month** Day-to-Time** Indicator**
Nonelementary (group)	Alphanumeric	Alphanumeric
	** ***ISQL*** only	** ***ISQL*** only

TABLE 2. Relationship of the Class and Categories of Data Items

B.4. Selection of Character Representation and Radix

The value of a numeric item may be represented in either binary or decimal form depending on the equipment. In addition there are several ways of expressing decimal. Since these representations are actually combinations of bits, they are commonly called binary-coded decimal forms. The selection of radix is generally dependent upon the arithmetic capability of the computer. If more than one arithmetic radix is provided, the selection is dependent upon the specification of the USAGE clause.

The size of an elementary data item or a group item is the number of characters in standard data format of the item. Synchronization and usage may cause a difference between this size and that required for internal representation.

B.5. Algebraic Signs

Algebraic signs fall into two categories: operational signs, which are associated with signed numeric data items and signed numeric literals to indicate their algebraic properties; and editing signs, which appear, for example on edited reports to identify the sign of the item.

The SIGN clause permits the programmer to state explicitly the location of the operational sign. This clause is optional; if it is not used, operational signs will be represented as defined by **ICOBOL**. See The USAGE clause, page 189, 192, 226.

Editing signs are inserted into a data item through the use of the sign control symbols of the PICTURE clause.

B.6. Standard Alignment Rules

The standard rules for positioning data within an elementary item depend on the category of the receiving item. These rules are:

(1) If the receiving data item is described as numeric:

a. The data is aligned by decimal point and is moved to the receiving digit positions with zero fill or truncation on either end as required.

b. When an assumed decimal point is not explicitly specified, the data item is treated as if it has an assumed decimal point immediately following its right-most digit and is aligned as in paragraph 1a.

(2) If the receiving data item is a numeric edited data item, the data moved to the edited data item is aligned by decimal point with zero fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeros.

(3) If the receiving data item is alphanumeric (other than a numeric edited data item), alphanumeric edited, or alphabetic, the sending data is moved to the receiving character positions and aligned at the left-most character position in the data item with space fill or truncation to the right, as required.

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified.

(4) (***ISQL***) If the receiving data item is an interval, the data is aligned by the fields that compose the interval with re-computation of high-order fields or truncation of low order fields as necessary. With regard to fractional seconds, the seconds and fractional seconds are treated as a standard numeric data item with regard to alignment. For example, moving INTERVAL “48:12:13.1234" HOUR TO SECOND interval to INTERVAL DAY TO MINUTE will result in the value INTERVAL “2 0:12" DAY TO MINUTE, where the high-order is re-computed and the low-order is truncated.

B.7. Item Alignment for Increased Object-Code Efficiency

Some computer memories are organized in such a way that there are natural addressing boundaries in the computer memory (e.g., word boundaries, half-word boundaries, byte boundaries). The way in which data is stored is determined by the object program, and need not respect these natural boundaries.

However, certain uses of data (e.g., in arithmetic operations or in subscripting) may be facilitated if the data is stored so as to be aligned on these natural boundaries. Specifically, additional machine operations in the object program may be required for the accessing and storage of data if portions of two or more data items appear between adjacent natural boundaries, or if certain natural boundaries bifurcate a single data item.

Data items which are aligned on these natural boundaries in such a way as to avoid such additional machine operations are defined to be synchronized.

Synchronization can be accomplished in two ways:

(1) By use of the SYNCHRONIZED clause.

(2) By recognizing the appropriate natural boundaries and organizing the data suitably without the use of the SYNCHRONIZED clause.

ICOBOL treats the SYNCHRONIZED clause as documentation. However, it aligns each 77 and 01 level item on an even byte address. (This default alignment can be altered with the -B compiler switch to select 1, 2, or 4 byte alignment.)

B.8. Table Handling

Tables of data are common components of business data processing problems. Although the repeating items that make up a table could be otherwise described by a series of separate data description entries all having the same level-number and all subordinate to the same group item, there are two reasons why this approach is not satisfactory. First, from a documentation standpoint, the underlying homogeneity of the items would not be readily apparent; and second, the problem of making available an individual element of such a table would be severe when there is a decision as to which element is to be made available at object time.

Tables of data items are defined in COBOL by including the OCCURS clause in their data description entries. This clause specifies that the item is to be repeated as many times as stated. The item is considered to be a table element and its name and description apply to each repetition or occurrence. Since each occurrence of a table element does not have assigned to it a unique data-name, reference to a desired occurrence may be made only by specifying the data-name of the table element together with the occurrence number of the desired table element. The occurrence number is known as a subscript.

The number of occurrences of a table element may be specified to be fixed or variable.

(***ISQL***) An SQL table is very similar to a simple two-dimensional data table in COBOL. It can be defined quite simply as one or more columns and zero or more rows with each row containing one elementary value for each column.

B.8.1 Table Definition

To define a one-dimensional table, the programmer uses an OCCURS clause as part of the data description of the table element, but the OCCURS clause must not appear in the description of group items which contain the table element. The following example shows a one-dimensional table defined by the item TABLE-ELEMENT.

```
01 TABLE-1.
       02 TABLE-ELEMENT OCCURS 20 TIMES.
              03 DOG...
              03 FOX...
```

EXAMPLE 7. Definition for a one-dimensional table

In the next example, TABLE-ELEMENT defines a one-dimensional table, but DOG does not since there is an OCCURS clause in the description of the group item (TABLE-ELEMENT) which contains DOG.

```
02 TABLE-1.
    03 TABLE-ELEMENT OCCURS 20 TIMES.
        04 DOG OCCURS 5 TIMES.
             05 EASY...
             05 FOX...
```

EXAMPLE 8. Another one-dimensional table

In both of the two previous examples, the complete set of occurrences of TABLE-ELEMENT has been assigned the name TABLE-1. However, it is not necessary to give a group name to the table unless it is desired to refer to the complete table as a group item.

None of the three one-dimensional tables which appear in the following two examples has a group name.

Example 9A:

```
01 ABLE.
    02 BAKER...
    02 CHARLIE OCCURS 20 TIMES...
    02 DOG...
```

Example 9B:

```
01 ABLE.
    02 BAKER OCCURS 20 TIMES...
    02 CHARLIE...
    02 DOG OCCURS 5 TIMES...
```

EXAMPLE 9. Three one-dimensional tables without group names

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table gives rise to a two-dimensional table. To define a two-dimensional table, then, an OCCURS clause must appear in the data description of the element of the table, and in the description of only one group item which contains that table element. Thus, in the next example, DOG is an element of a two-dimensional table; it occurs 5 times within each element of the item BAKER which itself occurs 20 times. BAKER is an element of a one dimensional table.

```
01 ABLE.
    02 BAKER OCCURS 20 TIMES...
        03 CHARLIE...
        03 DOG OCCURS 5 TIMES...
```

EXAMPLE 10. Definition for a two-dimensional table

In the general case, to define an n-dimensional table, the OCCURS clause should appear in the data description of the element of the table and in the descriptions of (n - 1) group items which contain the element.

B.8.2 Initial Values of Tables

In the Working-Storage Section, initial values of elements within tables are specified in one of the following ways:

(1) The table may be described as a series of separate data description entries all subordinate to the same group item, each of which specifies the value of an element, or part of an element, of the table. In defining the record and its elements, any data description clause (USAGE, PICTURE, etc.) may be used to complete the definition, where required. The hierarchical structure of the table is then shown by use of the REDEFINES entry and its associated subordinate entries. The subordinate entries, following the REDEFINES entry, which are repeated due to OCCURS clauses, must not contain VALUE clauses.

(2) All the dimensions of a table may be initialized by associating the VALUE clause with the description of the entry defining the entire table. The lower level entries will show the hierarchical structure of the table; lower level entries must not contain VALUE clauses.

(3) The value of selected table elements may be specified using VALUE clauses.

B.8.3 References to Table Items

Whenever the user references a table element or a condition-name associated with a table element, the reference must indicate which occurrence of the element is intended. For access to a one-dimensional table the occurrence number of the desired element provides complete information. For tables of more than one dimension, an occurrence number must be supplied for each dimension of the table. In the previous example, then, a reference to the fourth BAKER or the fourth CHARLIE would be complete, whereas a reference to the fourth DOG would not. To reference DOG, which is an element of a two-dimensional table, the user must reference, for example, the fourth DOG in the fifth BAKER.

```
01 ABLE.
    02 BAKER OCCURS 20 TIMES...
        03 CHARLIE...
        03 DOG OCCURS 5 TIMES...

Invalid (DOG needs 2 subscripts):

        DISPLAY BAKER(4) CHARLIE(4) DOG(4).

Valid:
        DISPLAY BAKER(4) CHARLIE(4) DOG(5,4).
```

EXAMPLE 11. Referencing single- and multi-dimensional table elements

B.8.4 Subscripting

Occurrence numbers are specified by appending one or more subscripts to the data-name.

The subscript can be represented either by an integer, a data-name which references an integer numeric elementary item, or an index-name associated with the table. A data-name or index-name may be followed by either the operator + or the operator - and an integer, which is used as an increment or decrement, respectively. It is permissible to mix integers, data-names, and index-names. In addition to these standard subscripting options, **ICOBOL** allows any arithmetic expression which evaluates to a positive integer to be used as a subscript.

The subscripts, enclosed in parentheses, are written immediately following any qualification for the name of the table element. The number of subscripts in such a reference must equal the number of dimensions in the table whose element is being referenced. That is, there must be a subscript for each OCCURS clause in the hierarchy containing the data-name including the data-name itself.

When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization. If a multi-dimensional table is thought of as a series of nested tables and the most inclusive or outermost table in the nest is considered to be the major table with the innermost or least inclusive table being the minor table, the subscripts are written from left to right in the order major, intermediate, and minor.

A reference to an item must not be subscripted if the item is not a table element or an item or condition-name within a table element.

The lowest permissible occurrence number is 1. The highest permissible occurrence number in any particular case is the maximum number of occurrences of the item as specified in the OCCURS clause.

B.8.4.1 Subscripting Using Integers or Data-Names

When an integer or data-name is used to represent a subscript, it may be used to reference items within different tables. These tables need not have elements of the same size. The same integer or data-name may appear as the only subscript with one item and as one of two or more subscripts with another item.

B.8.4.2 Subscripting Using Index-Names

In order to facilitate such operations as table searching and manipulating specific items, a technique called indexing is available. To use this technique, the programmer assigns one or more index-names to an item whose data description entry contains an OCCURS clause. An index associated with an index-name acts as a subscript, and its value corresponds to an occurrence number for the item to which the index-name is associated.

The INDEXED BY phrase, by which the index-name is identified and associated with its table, is an optional part of the OCCURS clause. There is no separate entry to describe the index associated with index-name since its definition is completely hardware oriented. At object time the contents of the index correspond to an occurrence number for that specific dimension of the table with which the index is associated. The initial value of an index at object time is undefined, and the index must be initialized before use. The initial value of an index is assigned with the PERFORM statement with the VARYING phrase, the SEARCH statement with the ALL phrase, or the SET statement.

The use of an arithmetic-expression or data-name as a subscript referencing a table element or an item within a table element does not cause the alteration of any index associated with that table.

An index-name can be used to reference only the table to which it is associated via the INDEXED BY phrase.

Relative indexing is an additional option for making references to a table element or to an item within a table element. When the name of a table element is followed by a subscript of the form (index-name + or - integer), the occurrence number required to complete the reference is the same as if index-name were set up or down by integer via the SET statement before the reference. The use of relative indexing does not cause the object program to alter the value of the index.

The value of an index can be made accessible to an object program by storing the value in an index data item. Index data items are described in the program by a data description entry containing a USAGE IS INDEX clause. The index value is moved to the index data item by the execution of a SET statement.

The following example illustrates the subscripts needed for various elements in an example table.

EXAMPLE 12. Referencing elements in 1-, 2-, and 3-dimensional tables

```
Consider the following data definition:

        02 XCOUNTER...

        02 BAKER OCCURS 20 TIMES INDEXED BY BAKER-INDEX...
            03 CHARLIE...
            03 DOG OCCURS 5 TIMES...
               04 EASY
               88 MAX VALUE IS...
               04 FOX...
                   05 GEORGE OCCURS 10 TIMES...
                       06 HARRY...
                       06 JIM...

The number of subscripts required to reference various table
elements is as follows, with an example for each:

        1 subscript:   BAKER(20)
                       CHARLIE(12)

        2 subscripts:  DOG(20,5)
                       EASY(5,5)
                       MAX(11,3)
                       FOX(5,1)

        3 subscripts:  GEORGE(20,5,10)
                       HARRY(5,5,5)
                       JIM(12,1,1)
```

B.9. Uniqueness of Reference

The purpose of every user-defined name in a COBOL program is to name a resource that is to be used in solving a data processing problem. (See User-defined words, on Page 40.) In order to use a resource, a statement in a COBOL program must contain a reference that uniquely identifies the resource. In order to ensure uniqueness of reference, a user-defined name may be qualified, subscripted , or reference modified, as described in the following paragraphs.

When the same name has been assigned in separate programs to two or more occurrences of a resource of a given type, and when qualification by itself does not allow the reference in one of those programs to differentiate between the two identically named resources, then certain conventions which limit the scope of names apply. These conventions ensure that the resource identified is that described in the program containing the reference.

Unless otherwise specified by the rules for a statement, any subscripting and reference modification are evaluated only once as the first operation of the execution of that statement.

B.9.1 Qualification

Every user-defined name explicitly referenced in a COBOL source program must be uniquely referenced because either:

(1) No other name has the identical spelling and hyphenation.

(2) It is unique within the context of a REDEFINES clause.

(3) The name exists within a hierarchy of names such that reference to the name can be made unique by mentioning one or more of the higher level names in the hierarchy.

These higher level names are called qualifiers and this process that specifies uniqueness is called qualification. Identical user-defined names may appear in a source program; however, uniqueness must then be established through qualification for each user-defined name explicitly referenced, except in the case of redefinition. All available qualifiers need not be specified so long as uniqueness is established. The LINAGE-COUNTER identifier requires qualification to provide uniqueness of reference whenever a source program would result in more than one occurrence of the identifier.

Regardless of the above, the same data-name must not be used as the name of an external record and as the name of any other external data item described in any program contained within or containing the program which describes that external data record.

The general formats for qualification are:

Format 1:

$$\left\{\begin{array}{l}\textit{data-name-1}\\ \textit{condition-name}\end{array}\right\}\left\{\begin{array}{c}\left\{\left\{\begin{array}{c}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\}\textit{data-name-2}\right\}\ldots\left[\left\{\begin{array}{c}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\}\textit{file-name}\right]\\ \left\{\begin{array}{c}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\}\textit{file-name}\end{array}\right\}$$

Format 2:

$$\textit{paragraph-name}\left\{\begin{array}{c}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\}\qquad\textit{section-name}$$

Format 3:

$$\text{LINAGE-COUNTER}\left\{\begin{array}{c}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\}\qquad\textit{file-name}$$

The rules for qualification are as follows:

(1) For each non-unique user-defined name that is explicitly referenced, uniqueness must be established through a sequence of qualifiers which precludes any ambiguity of reference.

(2) A name can be qualified even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used.

(3) IN and OF are logically equivalent.

(4) In Format 1, each qualifier must be the name associated with a level indicator, the name of a group item to which the item being qualified is subordinate, or the name of the conditional variable with which the condition-name being qualified is associated. Qualifiers are specified in the order of successively more inclusive levels in the hierarchy.

(5) In Format 1, *data-name-1* or *data-name-2* may be a record-name.

(6) If explicitly referenced, a *paragraph-name* must not be duplicated within a section. When a *paragraph-name* is qualified by a *section-name*, the word SECTION must not appear. A *paragraph-name* need not be qualified when referred to from within the same section.

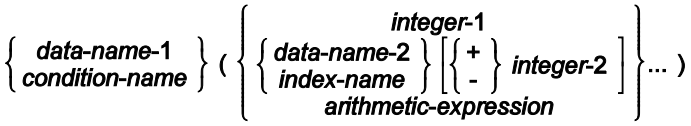
(7) LINAGE-COUNTER must be qualified each time it is referenced if more than one file description entry containing a LINAGE clause has been specified in the source program.

B.9.2 Subscripting

B.9.2.1 Function

Subscripts are used when reference is made to an individual element within a table of like elements that have not been assigned individual data-names.

B.9.2.2 General Format



B.9.2.3 Syntax Rules

(1) The data description entry containing *data-name-1* or the data-name associated with *condition-name* must contain an OCCURS clause or must be subordinate to a data description entry which contains an OCCURS clause.

(2) Except as defined in syntax rule 4, when a reference is made to a table element, the number of subscripts must equal the number of OCCURS clauses in the description of the table element being referenced. When more than one subscript is required, the subscripts are written in the order of successively less inclusive dimensions of the table.

(3) *Index-name* must correspond to a data description entry in the hierarchy of the table being referenced which contains an INDEXED BY phrase specifying that *index-name*.

(4) Each table element reference must be subscripted except when such reference appears:

a) in a REDEFINES clause.

b) as subject of a SEARCH statement,

c) in the KEY IS phrase of an OCCURS clause.

(5) *Data-name-2* may be qualified and must be a numeric elementary item representing an integer.

(6) *Integer-1* may be signed and, if signed, it must be positive.

(7) *Arithmetic-expression* is any arithmetic expression that evaluates to a positive integer not more than the number of occurrences specified in the OCCURS clause(s) associated with the table element being referenced. (Note that all other forms are special cases of the arithmetic expression and are presented only for clarity.)

B.9.2.4 General Rules

(1) The value of the subscript must be a positive integer. The lowest possible occurrence number represented by a subscript is 1. The first element of any given dimension of a table is referenced by an occurrence number of 1. Each successive element within that dimension of the table is referenced by occurrence numbers of 2, 3, ... The highest permissible occurrence number for any given dimension of the table is the maximum number of occurrences of the item as specified in the associated OCCURS clause.

(2) The value of the index referenced by *index-name* corresponds to the occurrence number of an element in the associated table.

(3) The value of the index referenced by *index-name* must be initialized before it is used as a subscript. An index may be given an initial value by either a PERFORM statement with the VARYING phrase, or a SET statement. An index may be modified only by the PERFORM and SET statements.

(4) If *integer-2* is specified, the value of the subscript is determined by incrementing by the value of *integer-2* (when the operator + is used) or by decrementing by the value of *integer-2* (when the operator - is used) either the occurrence number represented by the value of the index referenced by *index-name* or the value of the data item referenced by *data-name-2*.

(5) If *arithmetic-expression* is specified, the value of the subscript is determined by evaluating the expression and using this result to specify the occurrence number. This value must evaluate to a positive integer between 1 and the specified maximum for the associated OCCURS clause.

B.9.3 Identifiers

B.9.3.1 Identifier

An identifier is a sequence of character-strings and separators used to reference a data item uniquely.

B.9.3.1.1 General format

Format 1 (function-identifier):

function-identifier-1

Format 2 (qualified-data-name-with-subscripts):

qualified-data-name-with-subscripts-1

Format 3 (reference-modification):

identifier-1 reference-modifier-1

Format 4 (predefined-address):

NULL

Format 5 (address-identifier):

ADDRESS OF *identifier-1*

Format 6 (qualified-linage-counter):

LINAGE-COUNTER { IN / OF } *filename-1*

Format 7 (sqlstate-identifier):

SQLSTATE

Format 8 (length-identifier):

LENGTH OF *identifier-1*

B.9.3.1.2 Syntax rules

All Formats

(1) Identifier is defined recursively: whenever the format for an identifier allows another identifier to be specified, that other identifier may be any of the formats for an identifier, including the one being defined provided the rules for each format are followed.

Format 1

(2) Function-identifier is defined on page 129.

Format 2

(3) Qualified-data-name-with-subscripts is defined on page 126, under Subscripting.

Format 3

(4) Reference-modification is defined on page 130.

Format 4

(5) Predefined-address is defined on page 131. This format is not available under ***VXCOBOL***.

Format 5

(6) Address-identifier is defined on page 132. This format is not available under ***VXCOBOL***.

Format 6

(7) Qualified-linage-counter is defined on page 133.

Format 7

(8) (***ISQL***) Sqlstate-identifier is defined on page 133.

Format 8

(9) Length-identifier is defined on page 132.

B.9.3.1.3 General rules

(1) The order in which the various components of an identifier are applied is as follows, with the first to be applied listed first:

a. a *qualified-data-name-with-subscript*; a *function-identifier* without arguments; a *qualified-linage-counter*, a *sqlstate-identifier* or a predefined address are atomic identifiers

b. an *address-identifier* or *length-identifier* applies to an identifier on the right

c. a *function-identifier* with arguments applies the function-name on the left to a list of arguments enclosed

in parentheses on the right

d. a *reference-modifier* applies to the identifier on the left.

B.9.3.2 Function-identifier

A *function-identifier* references the unique data item that results from the evaluation of a function.

B.9.3.2.1 General format

FUNCTION { *intrinsic-function-name-1* } [([*argument-1*]...)]

B.9.3.2.2 Syntax rules

(1) A *function-identifier* shall not be specified as a receiving operand.

(2) The word FUNCTION is required.

(3) If a function's definition permits arguments and a left parenthesis immediately follows *intrinsic-function-name-1*, the left parenthesis is always treated as the left parenthesis of that function's arguments.

```
NOTE — For a function that may be referenced either with or without
arguments, such as the RANDOM function, careful coding is necessary to
ensure correct interpretation.

For example, in the following

       FUNCTION MAX (FUNCTION RANDOM (A) B)

'A' is treated as an argument to the RANDOM function. If 'A' is instead
meant to be a second argument to the MAX function, different coding is
necessary - either:

       FUNCTION MAX ( (FUNCTION RANDOM) (A) B)
or
       FUNCTION MAX (FUNCTION RANDOM () A B)
or
       FUNCTION MAX (FUNCTION RANDOM A B).
```

EXAMPLE 13. Referencing an intrinsic function with and without arguments

(4) *Argument-1* shall be an identifier, a literal, or an arithmetic expression. Specific rules governing the number, class, and category of *argument-1* are given for intrinsic functions in the definition of that intrinsic function.

(5) A numeric function shall not be specified where an integer operand is required, even though a particular reference of the numeric function might yield an integer value.

(6) An integer function other than the integer form of the ABS function shall not be specified where an unsigned integer is required.

B.9.3.2.3 General rules

(1) A *function-identifier* references a temporary data item whose value is determined when the function is referenced at runtime.

If *intrinsic-function-name-1* is specified, the temporary data item is an elementary data item whose description and category are specified by the definition of that intrinsic function. The Intrinsic Functions section begins on Page 579.

(2) At the time reference is made to a function, its arguments are evaluated individually in the order specified in the list of arguments, from left to right. An argument being evaluated may itself be a function-identifier or may be an expression containing function-identifiers. There is no restriction preventing the function referenced in evaluating an argument from being the same function as that for which the argument is specified. Additional rules for intrinsic functions are given in the definitions for each intrinsic function, beginning on Page 579.

(3) If a required argument is omitted, the **ICOBOL** compiler gives an error. There is no runtime error for a missing argument.

(4) Evaluation of the function-identifier proceeds as follows:

a. Each *argument-1* is evaluated at the beginning of the evaluation of the function-identifier. If an exception condition exists, no function is activated. If an exception condition does not exist, the values of argument-1 are made available to the activated function at the time control is transferred to that function.

b. he function specified by the *function-identifier* is made available for execution and control is transferred to the activated function in a manner consistent with the call convention for the function.

c. After control is returned from the activated function, any exception condition (e.g. SIZE ERROR) is propagated from the function and execution continues.

B.9.3.3 Reference-modifier

Reference modification defines a unique data item by specifying an identifier, a leftmost position, and a length.

B.9.3.3.1 General format

identifier-1 (*leftmost-position* : [*length*])

B.9.3.3.2 Syntax rules

(1) *Identifier-1* shall reference a data item that is an alphanumeric, elementary item, a group item, or a numeric item with USAGE DISPLAY..

(2) If *identifier-1* is a function-identifier, it shall reference an alphanumeric function.

(3) *Identifier-1* shall not be a reference-modification format identifier.

(4) *Leftmost-position* and *length* shall be arithmetic expressions.

(5) Unless otherwise specified, reference modification is allowed anywhere an identifier referencing a data item of class alphanumeric is permitted.

B.9.3.3.3 General rules

(1) *Leftmost-position* shall represent an alphanumeric position.

(2) If the data item referenced by *identifier-1* is explicitly or implicitly described as usage DISPLAY and its category is other than alphanumeric, it shall be operated upon for purposes of reference modification as if it were

redefined as a data item of class and category alphanumeric of the same size as the data item referenced by *identifier-1*.

(3) Each position of the data item referenced by *identifier-1* is assigned an ordinal number incrementing by one from the leftmost position to the rightmost position. The leftmost position is assigned the ordinal number one. If the data description entry for *identifier-1* contains a SIGN IS SEPARATE clause, the sign position is assigned an ordinal number within that data item.

(4) Reference modification creates a unique data item that is a subset of the data item referenced by *identifier-1*. This unique data item is defined as follows:

a. Positions used in evaluation are character positions.

b. The evaluation of *leftmost-position* specifies the ordinal position of the leftmost character of the unique data item in relation to the leftmost character of the data item referenced by *identifier-1*. Evaluation of *leftmost-position* shall result in a positive nonzero integer less than or equal to the number of positions in the data item referenced by *identifier-1*.

c. The evaluation of *length* specifies the number of character positions of the data item to be used in the operation. The evaluation of *length* shall result in a positive nonzero integer. The sum of *leftmost-position* and *length* minus the value one shall be less than or equal to the number of positions in the data item referenced by *identifier-1*. If *length* is not specified, the unique data item extends from and includes the position identified by *leftmost-position* up to and including the rightmost position of the data item referenced by *identifier-1*.

If the evaluation of *leftmost-position* or *length* results in a non-integer value or a value that references a position outside the area of *identifier-1*, the **ICOBOL** runtime system will halt the program executing with an appropriate error.

(5) The unique data item is considered to be an elementary data item without the JUSTIFIED clause. The unique data item has the same class, category, and usage as that defined for *identifier-1*, except that the categories numeric, numeric-edited, and alphanumeric-edited are considered class and category alphanumeric.

B.9.3.4 Predefined-address

NULL is a predefined address of class pointer.

B.9.3.4.1 General Format

NULL

B.9.3.4.2 Syntax Rules

(1) This format may be used only as a sending operand in a SET statement, in the VALUE clause of an item with usage POINTER, or in a data-pointer relation-condition.

B.9.3.4.3 General Rules

(1) The predefined address NULL references a data item of category data-pointer that contains the null address; i.e., it does not represent the address of any data item.

B.9.3.5 Data-address-identifier

A data-address-identifier references the unique data item that contains the address of a data item.

B.9.3.5.1 General Format

ADDRESS OF *identifier-1*

B.9.3.5.2 Syntax Rules

(1) *Identifier-1* shall reference a data item defined in the file section, working-storage section, or linkage section.

(2) This identifier format shall not be specified as a receiving operand in a SET statement or in a data-pointer relation condition.

B.9.3.5.3 General Rules

(1) Data-address-identifier creates a unique data item of class pointer and category data-pointer that contains the address of *identifier-1*.

B.9.3.6 Length-identifier

A length-identifier references the unique data item that contains the length of a data item.

B.9.3.6.1 General Format

LENGTH OF *identifier-1*

B.9.3.6.2 Syntax Rules

(1) *Identifier-1* shall reference a data item defined in the file section, working-storage section, or linkage section.

(2) This identifier format shall not be specified as a receiving operand in a SET statement or in a data-pointer relation condition.

B.9.3.6.3 General Rules

(1) LENGTH OF references a temporary unsigned integer data item of class and category numeric whose size is equal to the number of character positions in *identifier-1*.

B.9.3.7 LINAGE-COUNTER

The LINAGE-COUNTER identifier is generated by the presence of a LINAGE clause in a file description entry.

B.9.3.7.1 General format

LINAGE-COUNTER { IN / OF } *filename-1*

B.9.3.7.2 Syntax rules

(1) LINAGE-COUNTER shall only be referenced in procedure division statements.

(2) The LINAGE-COUNTER identifier shall not be referenced as a receiving operand or as an operand in the USING list of a CALL or CALL PROGRAM statement..

(3) Qualification requirements for LINAGE-COUNTER are defined on Page 124, under Qualification.

B.9.3.7.3 General rules

(1) LINAGE-COUNTER references a temporary unsigned integer data item of class and category numeric whose size is equal to the page size specified in the LINAGE clause.

(2) The semantics of the LINAGE-COUNTER identifier is described on Page 153, under the LINAGE clause General Rules.

B.9.3.8 SQLSTATE (***ISQL***)

The SQLSTATE identifier is generated by the presence of an ***ISQL*** feature-set. Conceptually it is similar to a FILE STATUS item.

B.9.3.8.1 General format

SQLSTATE

B.9.3.8.2 Syntax rules

(1) SQLSTATE shall only be referenced in procedure division statements.

(2) The SQLSTATE identifier shall not be referenced as a receiving operand or as an operand in the USING list of a CALL or CALL PROGRAM statement.

(3) The SQLSTATE identifier shall not be subscripted, but it may be reference modified.

B.9.3.8.3 General rules

(1) SQLSTATE references a predefined data item of class and category alphanumeric whose size is exactly five characters and whose scope is the run unit.

(2) The value of the SQLSTATE data item is initialized to “00000" when the run unit is initialized.

(3) The value of the SQLSTATE data item is modified by the execution of the following ***ISQL-1*** statements: CONNECT, DISCONNECT, EXECUTE, FETCH, PREPARE, and SET CONNECTION.

(4) The value of the SQLSTATE data item is defined to be composed of a two-character class field followed by a three-character subclass field. Some common class field values are:

00 - Successful completion
01 - Warning
02 - Data not found
07 - Dynamic SQL error
08 - Connection error
0A - Feature not supported
21 - Cardinality violation
22 - Data exception
23 - Constraint violation
24 - Invalid cursor
25 - Invalid transaction state
26 - Invalid SQL identifier
40 - Rollback
42 - Syntax or access error
44 - Check option violation
HY -
IC - Generated by ICOBOL ISQL driver
IM - Generated by ODBC Driver Manager

(5) Some common values and their meaning:

00000 "Success"

From runtime/ISQL:

01000 "General Warning: The statement identifier does not exist"
01503 "The number of result columns is larger than the number of INTO items
02000 "No data was affected by the operation"
07001 "More data is needed"
07001 "The number of USING items is not the same as the number of parameter markers"
07004 "The USING clause is required for dynamic parameters"
07006 "Restricted data type attribute violation"
07500 "Numeric parameter conversion error"
07501 "Date parameter conversion error"
07502 "Time parameter conversion error"
07503 "Timestamp parameter conversion error"
07504 "Interval parameter conversion error"
08001 "Client unable to establish connection"
08002 "Connection name in use"
08003 "Connection does not exist"
08004 "Server rejected connection"
08S01 "Communication link failure"

22002 "Indicator variable required but not supplied"
22003 "Numeric value out of range"
22007 "Invalid datetime format"
22015 "Interval field overflow"
22018 "Invalid character value for cast specification"
24000 "Invalid cursor state"
26501 "The statement identifier does not exist"
28000 "Invalid authorization"
28001 "Authorization failure: ICSQL License could not be opened"

HY000 "General error"
HY001 "Memory allocation error"
HY004 “Invalid SQL type"
HY009 "Invalid use of null pointer"
HY010 "Invalud sequence error"
HY013 "Memory management error"
HY090 "Invalid string or buffer length"

IC001 “General error: SQL is not loaded"
IC002 “Unable to load ODBC"
IC003 “Unable to load ODBC symbols"
IC004 “The ISQL subsystem is not properly initialized”
IC005 “Get Diagnostics exception number is out of range"
IC006 “Unable to allocate ODBC environment”
IC007 “Memory allocation error"
IC008 “Internal error"
IC009 “Unexpected Error from ODBC"
IC010 “Invalid Handle error from ODBC"

From driver/driver manager:

01001 “Cursor operation conflict”
01002 “Disconnect error”
01003 “NULL value eliminated in set function”
01004 “String data right truncated”
07002 “COUNT field incorrect”
08001 "Client unable to establish connection"
08002 "Connection name in use"
08003 "Connection does not exist"
08004 "Server rejected connection"
08S01 "Communication link failure"
23000 “Integrity constraint violation”
24000 "Invalid cursor"
25000 “Invalid transaction state”
28000 "Invalid authorization"

42000 “Syntax error or access violation”
HY000 "General error"
HY001 "Memory allocation error"
HY009 "Invalid use of null pointer"
HY010 "Invalud sequence error"
HY013 "Memory management error"
HY090 "Invalid string or buffer length"

HYC00 "Optional feature not implemented"
HYT00 "Timeout expired before the connection was made"
HYT01 "Connection timeout expired before the data source responded"
IM001 "Driver does not support this function"
IM002 “Database not found”
IM003 "Specified driver could not be connected to"
IM004 "Allocate on Environment failed"
IM005 "Allocate on DBC failed"
IM009 "Unable to load translation DLL"
IM010 "Data source name too long"

These are only some messages. The Driver Manager and/or Driver may have many more. Use the GET DIAGNOSTICS statement to retrieve the text of the messages.

B.9.4. Condition-Name

A condition-name identifies a specific value, set of values, or range of values, within a complete set of values that a data item may assume. The data item itself is called a conditional variable.

Condition-names may be defined in the data division or in the SPECIAL-NAMES paragraph within the environment division where a condition-name shall be assigned to the on or off status, or both of implementor-defined switches.

A condition-name is used in conditions as an abbreviation for the relation condition; this relation condition posits that the associated conditional variable is equal to one of the set of values to which that condition-name is assigned. A condition-name is also used in a SET statement, indicating either that a value is moved to the associated conditional variable that make the condition-name either 'true' or 'false', depending on the format of the SET statement, or that an implementor-defined switch is set to 'on' or 'off' status.

If explicitly referenced, a condition-name must be unique or be made unique through qualification and/or subscripting except when the scope of the names conventions by themselves ensure uniqueness of reference.

If qualification is used to make a condition-name unique, the associated conditional variable may be used as the first qualifier. If qualification is used, the hierarchy of names associated with the conditional variable itself must be used to make the condition-name unique.

If references to a conditional variable require subscripting, reference to any of its condition-names also requires the same combination of subscripting.

The format and restrictions on the combined use of qualification and subscripting of condition-names is exactly that of a Format 2 `identifier' . See page 126 under Subscripting.

In the general format of the chapters that follow, `condition-name' refers to a condition-name qualified or subscripted, as necessary.

C. Organization

The Data Division is subdivided into sections. These are the File, Working-Storage, Linkage, and Screen sections. With the ***VXCOBOL*** dialect, there is an additional section: Virtual-Storage.

C.1.1 Function

The *File Section* defines the structure of data files. Each file is defined by a file description entry and one or more record description entries, or by a file description entry and one or more report description entries. Record description entries are written immediately following the file description entry.

The *Virtual-Storage Section* (***VXCOBOL***) and the *Working-Storage Section* describe records and subordinate data items which are not part of external data files but are developed and processed internally. Also described in these sections are data items whose values are assigned in the source program and whose values do not change during the execution of the object program.

The *Linkage Section* appears in the called program and describes data items that are to be referred to by the calling program and the called program. Its structure is the same as the Working-Storage Section.

The *Screen Section* describes various input and output structures called screens that can be used by the ACCEPT and DISPLAY verbs to present and/or get entire screen of data including literal fields.

C.1.2 General Format

The following gives the general format of the sections in the Data Division, and defines the order of their presentation in the source program. The VIRTUAL-STORAGE section is available only with the ***VXCOBOL*** dialect.

```
DATA DIVISION.

[ FILE SECTION.
  [ file-description-entry { record-description-entry }...
  [ sort-merge-file-description-entry { record-description-entry }... ]...

[ WORKING-STORAGE SECTION.
[ 77-level-description-entry ]
[ record-description-entry ] ... ]

[ VIRTUAL-STORAGE SECTION.                    (VXCOBOL only)
[ 77-level-description-entry ]
[ record-description-entry ] ... ]

[ LINKAGE SECTION.
[ 77-level-description-entry ]
[ record-description-entry ] ... ]

[ SCREEN SECTION.
 [ screen-description-entry ]... ]
```

D. FILE SECTION

The File Section is located in the Data Division of a source program. The File Section defines the structure of data files and sort files and merge files. Each data file is defined by a file description entry and one or more record description entries. Each sort or merge file is defined by a sort-merge file description entry and one or more record description entries. Record description entries are written immediately following the file description entry.

The general format of the File Section is shown below.

```
FILE SECTION.
[      file-description-entry {record-description-entry}...
 sort-merge-file-description-entry {record-description-entry}... ]...
```

D.1. File Description Entry/Sort-Merge Description Entry

In a COBOL program the file description entry (FD entry) represents the highest level of organization in the File Section. The File Section header is followed by a file description entry consisting of a level indicator (FD), a file-name, and a series of independent clauses. The clauses of a file description entry (FD entry) specify a number of attributes of the file. The entry itself is terminated by a period.

In a COBOL program the sort-merge file description entry (SD entry) represents the highest level of organization in the File Section. The File Section header is followed by a sort-merge file description entry consisting of a level indicator (SD), a file-name, and a series of independent clauses. The clauses of a sort-merge file description entry (SD entry) specify the size and the names of the data records associated with a sort file or a merge file. There are no label procedures which the user can control, and the rules for blocking and internal storage are peculiar to the SORT and MERGE statements. The entry itself is terminated by a period.

D.1.1 Function

The file description entry furnishes information concerning the physical structure, identification, and record-names pertaining to a file.

The sort-merge file description entry furnishes information concerning the physical structure and record-names pertaining to a sort or merge file.

D.1.2. General Format

Below is the general format with each phrase in alphabetical order since they are order independent.

Sequential File: (***ANSI 74*** and ***ANSI 85***)

```
FD file-name [ IS EXTERNAL ]
d     [ BLOCK CONTAINS integer [ TO integer ] { RECORDS | CHARACTERS } ]
      [ CODE-SET IS { ASCII | STANDARD-1 | NATIVE | EBCDIC | alphabet-name }
                    [ { FIELD [IS] | FIELDS [ARE] } identifier, ... ] ]
d     [ DATA { RECORD IS | RECORDS ARE } { data-name }... ]
d     [ LABEL { RECORD IS | RECORDS ARE } { STANDARD | OMITTED } ]
      [ [ LINAGE IS { identifier | literal } LINES [ WITH FOOTING AT { identifier | literal } ]
                                                   [ LINES AT TOP { identifier | literal } ]
        [ LINES AT BOTTOM { identifier | literal } ] ]
      [ RECORD { CONTAINS integer CHARACTERS
               | IS VARYING IN SIZE [ [ FROM integer ] [ TO integer ] CHARACTERS ]
                    DEPENDING ON data-name
               | CONTAINS integer TO integer CHARACTERS } ]
      [ RECORDING MODE IS { VARIABLE | FIXED } ] .
```

Sequential File: (***VXCOBOL***)

FD *file-name* [IS EXTERNAL]

d [BLOCK CONTAINS integer [TO integer] { RECORDS / CHARACTERS }]

[CODE-SET [IS] { ASCII / STANDARD-1 / NATIVE / EBCDIC / *alphabet-name* } [{ FIELD [IS] / FIELDS [ARE] } *identifier*, ...]

d [DATA { RECORD IS / RECORDS ARE } { *data-name* }...]

d [LABEL { RECORD IS / RECORDS ARE } { ASCII [*integer*] / NATIVE [*integer*] / STANDARD [*integer*] / EBCDIC [*integer*] / OMITTED }]

[[LINAGE IS { *identifier* / *literal* } LINES [WITH FOOTING AT { *identifier* / *literal* } { *identifier* / *literal* }] LINES AT TOP

[LINES AT BOTTOM { *identifier* / *literal* }]]

d [MULTIPLE I-O PROCEDURES]

d [PAD CHARACTER IS { *identifier* / *literal* }]

d [RECORD CONTAINS *integer* [TO *integer*] CHARACTERS]

[RECORDING MODE IS

{ FIXED / UNDEFINED [RECORD LENGTH IS identifier] / VARIABLE [RECORD LENGTH IS identifier] / DATA-SENSITIVE [DELIMITER IS literal] [RECORD LENGTH IS identifier] / DYNAMIC RECORD LENGTH IS identifier }]

d [VALUE OF [OWNER IS identifier] [EXPIRATION DATE IS identifier]

d [SEQUENCE NUMBER IS identifier] [GENERATION NUMBER IS identifier]

d [ACCESSIBILITY IS identifier] [OFFSET IS identifier]

d [VOLUME STATUS IS identifier] [USER VOLUME { LABEL IS / LABELS ARE } identifier, ...]

d [USER HEADER { LABEL IS / LABELS ARE } identifier, ...]

d [USER TRAILER { LABEL IS / LABELS ARE } identifier, ...]] .

Relative File & Indexed File: (***ANSI 74*** and ***ANSI 85***)

```
    FD file-name [ IS EXTERNAL ]
d       [ BLOCK CONTAINS integer [ TO integer ] { RECORDS | CHARACTERS } ]
d       [ DATA { RECORD IS | RECORDS ARE } { data-name }... ]
d       [ LABEL { RECORD IS | RECORDS ARE } { STANDARD | OMITTED } ]
        [ RECORD { CONTAINS integer CHARACTERS
                 | IS VARYING IN SIZE [ [ FROM integer ] [ TO integer ] CHARACTERS ]
                        DEPENDING ON data-name
                 | CONTAINS integer TO integer CHARACTERS } ]
```

Relative File: (***VXCOBOL***)

```
    FD file-name [ IS EXTERNAL ]
d       [ BLOCK CONTAINS integer [ TO integer ] { RECORDS | CHARACTERS } ]
d       [ DATA { RECORD IS | RECORDS ARE } { data-name }... ]
d       [ LABEL { RECORD IS | RECORDS ARE } { STANDARD | OMITTED } ]
d       [ PAD CHARACTER IS { identifier | literal } ]
d       [ RECORD CONTAINS integer [ TO integer ] CHARACTERS ]
        [ RECORDING MODE IS FIXED ] .
```

Indexed File: (***VXCOBOL***)

```
    FD file-name [ IS EXTERNAL ]
d       [ DATA BLOCK CONTAINS integer [ TO integer ] { RECORDS | CHARACTERS } ]
d       [ DATA { RECORD IS | RECORDS ARE } { data-name }... ]
d       [ FEEDBACK IS identifier ]
d       [ MERIT IS identifier ]
d       [ INDEX BLOCK CONTAINS [ integer TO ] integer CHARACTERS ]
d       [ INDEX NODE SIZE IS integer CHARACTERS ]
d       [ LABEL { RECORD IS | RECORDS ARE } { STANDARD | OMITTED } ]
d       [ RECORD CONTAINS integer [ TO integer ] CHARACTERS ]
        [ RECORDING MODE IS { FIXED
                            | VARIABLE [ RECORD LENGTH IS identifier ] } ]
```

INFOS File: (***VXCOBOL***)

```
FD file-name [ IS EXTERNAL ]
    [ DATA BLOCK CONTAINS [ integer TO ] integer { RECORDS | CHARACTERS } ]
d   [ DATA { RECORD IS | RECORDS ARE } { data-name }... ]
    [ FEEDBACK IS identifier ]
d   [ MERIT IS identifier ]
    [ INDEX BLOCK CONTAINS [ integer TO ] integer CHARACTERS ]
d   [ INDEX NODE SIZE IS integer CHARACTERS ]
d   [ LABEL { RECORD IS | RECORDS ARE } { STANDARD | OMITTED } ]
    [ PARTIAL RECORD IS identifier ]
d   [ RECORD CONTAINS integer [ TO integer ] CHARACTERS ]
    [ RECORDING MODE IS VARIABLE [ RECORD LENGTH IS identifier ] ] .
```

Sort-Merge File: (***ANSI 74*** and ***ANSI 85***)

```
SD file-name
d   [ DATA { RECORD IS | RECORDS ARE } { data-name }... ]
    [ RECORD CONTAINS integer [ TO integer ] CHARACTERS ] .
```

Sort-Merge File: (***VXCOBOL***)

```
SD file-name
d   [ BLOCK CONTAINS integer [ TO integer ] { RECORDS | CHARACTERS } ]
d   [ RECORDING MODE IS FIXED ]
d   [ DATA { RECORD IS | RECORDS ARE } { data-name }... ]
d   [ RECORD CONTAINS integer [ TO integer ] CHARACTERS ] .
```

D.1.3 Syntax Rules

(1) The level indicator FD identifies the beginning of a file description entry and must precede file-name.

(2) The level indicator SD identifies the beginning of a sort-merge file description entry and must precede *file-name*.

(3) The clauses which follow *file-name* may appear in any order.

(4) One or more record description entries must follow the file description entry.

(5) One or more record description entries must follow the sort-merge file description entry; however no input-output statements may be executed for this sort or merge file.

D.1.4 General Rules

(1) A file description entry associates *file-name* with a file connector.

(2) The following chart lists the file description clauses for all of the **ICOBOL** dialects, by file type. It also indicates which ones are for documentation purposes only. The clauses are presented on the following pages in alphabetical order, with one exception: INDEX BLOCK is described with DATA BLOCK. Of the "documentation only" clauses, only BLOCK CONTAINS, DATA RECORD and LABEL RECORD, are included.

File Description Clause	ANSI 74 & 85	*VXCOBOL*
BLOCK CONTAINS	Sequential (doc only) Relative & Indexed (doc only)	Sequential (doc only) Relative & Indexed (doc only)
CODE-SET	Sequential	Sequential
DATA BLOCK	N/A	Indexed (doc only) INFOS files
DATA RECORD	Sequential (doc only) Relative & Indexed (doc only)	Sequential (doc only) Relative & Indexed (doc only)
EXTERNAL	Sequential Relative & Indexed	Sequential Relative & Indexed INFOS
FEEDBACK	N/A	Indexed (doc only) INFOS (doc only)
INDEX BLOCK	N/A	Indexed (doc only) INFOS
INDEX NODE	N/A	Indexed (doc only) INFOS (doc only)
LABEL RECORD	Sequential (doc only) Relative & Indexed (doc only)	Sequential (doc only) Relative & Indexed (doc only) INFOS (doc only)
LINAGE	Sequential	Sequential
MERIT	N/A	Indexed & INFOS (doc only)
MULTIPLE	N/A	Sequential (doc only)
PAD CHARACTER	N/A	Sequential (doc only) Relative (doc only)
PARTIAL RECORD	N/A	INFOS
RECORD	Sequential Relative & Indexed	Sequential (doc only) Relative & Indexed (doc only) INFOS (doc only)
RECORDING MODE	Sequential	Sequential Relative & Indexed INFOS
VALUE OF	N/A	Sequential (doc only)

TABLE 3. File Description Clauses by **ICOBOL** dialect and file type, noting which are documentation only

D.2. Record Description Structure

A record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by the data-name or FILLER clause, if specified, followed by a series of independent clauses as required. A record description may have a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description and the elements allowed in a record description entry are explained under Concept of Levels on Page 117 and under Data Description Entry on Page 166.

D.3. Initial Values

The initial value of a data item in the File Section is undefined.

D.4. BLOCK CONTAINS Clause

D.4.1 Function

The BLOCK CONTAINS clause specifies the size of a physical record.

It is used for documentation purposes only.

D.4.2 General Format

d BLOCK CONTAINS *integer-1* [TO *integer-2*] { RECORDS / CHARACTERS }

D.4.3 General Rules

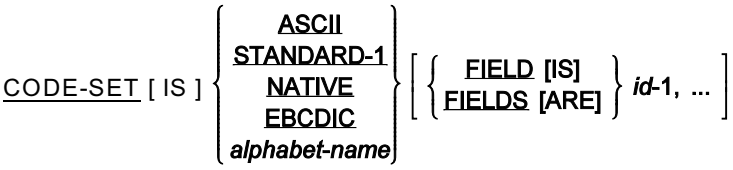
(1) The BLOCK CONTAINS clause is used for documentation purposes only, although the compiler does make some simple consistency checks on the values of *integer-1* and *integer-2*.

D.5. CODE-SET Clause

D.5.1 Function

The CODE-SET clause specifies the character code convention used to represent data on the external media.

D.5.2 General Format



D.5.3 Syntax Rules

(1) If the CODE-SET clause is specified for a file, all data in that file must be described as USAGE IS DISPLAY and any signed numeric data must be described with the SIGN IS SEPARATE clause.

(2) The alphabet-name clause referenced by the CODE-SET clause must not specify the literal phrase.

(3) If specified, each *id-1* must not be subscripted.

(4) *id-1* must appear in a record-description for the associated file-connector, and if more than one *id-1* is specified all must appear within the same record-description.

(5) No two occurrences of *id-1* may reference all or part of the same storage area.

D.5.4 General Rules

(1) ASCII, STANDARD-1, and NATIVE are equivalent and all represent the native character set of the computer. For **ICOBOL**, this is ASCII. EBCDIC represents the EBCDIC character set.

(2) If the CODE-SET clause is specified:

a. Upon successful execution of an OPEN statement, the character set used to represent the data on the external media is the one referenced by *alphabet-name* in the file-description entry associated with the file-name specified in the OPEN statement.

b. It specifies the algorithm for converting the character set on the external media from/to the native character set during the execution of an input or output operation. In particular, data is translated from the specified character to the native character set upon execution of a READ statement, and from the native character set to the specified character set upon execution of a WRITE or REWRITE statement. Note also that these translations also occur as part of SORT and MERGE statements when records are read or written pursuant to processing the USING or GIVING clauses.

c. If the FIELD IS/FIELDS ARE clause appears, the representation and conversion of data is restricted to the fields referenced by each *id-1*. Otherwise, the entire data record is affected.

(3) If the CODE-SET is not specified, the native character set is assumed for data on the external media.

(4) If the associated file-connector is an external file connector, all CODE-SET clauses in the run unit which are associated with that file connector must have the same character set. In addition, if the FIELD IS/FIELDS ARE

clause is specified, the number of occurrences and the offset and length of each *id-1* within the data record must be the same for each file-connector in the run unit that is associated with the external file-connector.

(5) If the associated file-connector is specified with RECORD DELIMITER IS DATA-SENSITIVE or literal (**ANSI 74 and ANSI 85**) or with RECORDING MODE IS DATA-SENSITIVE (***VXCOBOL***), any delimiter characters specified are assumed to be in the native character set and are translated to the character set specified by the CODE-SET clause.

(6) If the associated file-connector is specified with RECORD DELIMITER IS ASCII LENGTH (**ANSI 74 and ANSI 85**) or with RECORDING MODE IS VARIABLE (***VXCOBOL***), only the data contained within the record is translated. The record header, which contains the length, is assumed to be in the native character set and is not translated.

(7) If the associated file-connector is specified with ASSIGN TO PRINTER or ASSIGN TO DISPLAY, only the data contained within the record is translated. All carriage control is assumed to be in the native character set and is not translated.

(8) The record area accessible to the program is always specified in the native character set.

D.6. DATA BLOCK and INDEX BLOCK Clauses (***VXCOBOL***)

D.6.1 Function

The DATA BLOCK and INDEX BLOCK clauses specifies the page sizes used when creating an INFOS file.

D.6.2 General Format

DATA BLOCK CONTAINS [*integer-1* TO] *integer-2* { RECORDS / CHARACTERS }

INDEX BLOCK CONTAINS [*integer-3* TO] *integer-4*] CHARACTERS

D.6.3 Syntax Rules

(1) *Integer-1* and *integer-3* are ignored.

(2) *Integer-2* is a positive integer literal that specifies the maximum number of characters or records that a logical block in a data file can contain.

(3) *Integer-4* is a positive integer literal that specifies the maximum number of characters that a logical block in an indexed file can contain.

(4) Both clauses are ignored if specified for an indexed file.

D.6.4 General Rules

(1) DATA BLOCK and INDEX BLOCK clauses are used to specify the page sizes used when an INFOS file is created. Only two page sizes are allowed: 2048 characters and 4096 characters. Any value less than or equal to 2048 will be treated as 2048, and any value greater that 2048 will be treated as 4096. If the RECORDS keyword is specified in the DATA BLOCK CONTAINS clause, then integer-2 is multiplied by the record size and the result is used to select either a 2048 or 4096 characters page.

(2) If DATA BLOCK or INDEX BLOCK is not specified 2048 is used.

(3) **Note:** U/FOS also supports page sizes of 512, 1024, and 8192 if the file is created with the ufos_create utility. These may be specified in these clauses, but will result in runtime errors if a program attempts to create a file using any of these three values.

D.7. DATA RECORDS Clause

D.7.1 Function

The DATA RECORDS clause serves as documentation for the names of data records within their associated file. The DATA RECORDS clause is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL. We suggest that you remove it from your source or change it to be a comment line.

D.7.2 General Format

```
DATA { RECORD IS  }
     { RECORDS ARE }   { data-name }...
```

D.7.3 Syntax Rules

(1) *Data-name* is the name of a data record and must have an 01 level-number record description, with the same name, associated with it.

D.7.4 General Rules

(1) The DATA RECORDS clause is used for documentation purposes only, although the compiler checks that the specified names do occur as record descriptors.

(2) The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.

(3) Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

D.8. EXTERNAL Clause

D.8.1 Function

The EXTERNAL clause specifies that a file connector is external. The file and constituent data records are available in a run unit to all programs that describe the file as external.

D.8.2 General Format

IS EXTERNAL

D.8.3 Syntax Rules

(1) If you define data items in the FD or SELECT statement of an EXTERNAL file, you must specify them as EXTERNAL. For example, INFOS STATUS (***VXCOBOL***), FILE STATUS, RECORD LENGTH, etc. must be EXTERNAL if the file is external. The compiler will flag these items with a warning if they are not EXTERNAL but the file is EXTERNAL.

D.8 .4 General Rules

(1) The file connector associated with thsi file description entry is an external file connector.

(2) The data contained in all record description entries subordinate to that file description entry are external and may be accessed by any runtime element in the run unit that describesthe same file and records as external, subject to the following rules.

(3) Any LINAGE-COUNTER data item associated with the file is external.

(4) An EXTERNAL file uses the declaratives of the program which it is currently running. To ensure that the same action is taken for all exceptions, a COPY file for the declaratives should be used in all programs that reference this file.

(5) An EXTERNAL file can only be opened once. An error will occur if you attempt to open the file again (either in the main program or a subprogram) without first explicitly closing the file.

(6) If record keys are declared for a file with the EXTERNAL clause, then the record keys must also be declared EXTERNAL if there are not implicitly external by being in the data record, and in the same order in each subprogram which references the file.

(7) At runtime, if any of the file's characteristics do not match those of a previously referenced external file, **ICOBOL** will generate an exception status 1296 "External item in called program does not match existing item" on the call of a subprogram that contains an external file.

D.9. FEEDBACK Clause (***VXCOBOL***)

D.9.1 Function

The FEEDBACK clause contains the location of records for an INFOS file.

D.9.2 General Format

FEEDBACK IS *identifier*

D.9.3 Syntax Rules

(1) *Identifier* is a 4-byte data item in Working-Storage that receives feedback information about the location of records in INFOS files.

(2) This clause is ignored for an indexed file.

D.9.4 General Rules

(1) If you specify FEEDBACK for a file, each time you read, write, or rewrite a record in the file, the FEEDBACK data item is updated with the location of the record you just accessed. A WRITE INVERTED uses the FEEDBACK data item to obtain the location of the record to which another key is already pointing.

(2) READ, REWRITE, and WRITE update the FEEDBACK data items if specified.

(3) FEEDBACK can not be used to READ a particular record.

D.10. INDEX NODE Clause (***VXCOBOL***)

D.10.1 Function

The INDEX NODE clause specifies the size, in characters, of an index node in an INFOS file.

D.10.2 General Format

INDEX NODE SIZE IS *integer* CHARACTERS

D.10.3 Syntax Rules

(1) *Integer* is a positive integer literal that specifies the number of characters in an index node.

(2) This clause is ignored for an indexed file.

D.10.4 General Rules

(1) The node size must be large enough to hold three keys. If you omit this option, the system calculates the size according to the maximum key length, the partial record length, and whether or not subindexing is allowed.

D.11. LABEL RECORD Clause

D.11.1 Function

The LABEL RECORD clause specifies whether labels are present. The LABEL RECORD clause is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

D.11.2 General Format

ANSI 74 and ANSI 85

```
d  LABEL { RECORD IS   } { STANDARD }
         { RECORDS ARE } { OMITTED  }
```

VXCOBOL

```
d  LABEL { RECORD IS   } { ASCII [ int-1 ]    }
         { RECORDS ARE } { NATIVE [ int-1 ]   }
                         { STANDARD [ int-1 ] }
                         { EBCDIC [ int-2 ]   }
                         { OMITTED            }
```

D.11.3 Syntax Rules

(1) *int-1* is a positive integer literal indicating the level number of the tape; it may be either 1 or 3.

(2) *int-2* is a positive integer literal indicating the level number of the tape; it may be either 1 or 2.

D.11.4 General Rules

(1) The LABEL RECORD clause is used for documentation purposes only.

(2) OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned.

(3) STANDARD specifies that labels exist for the file or the device to which the file is assigned and the labels conform to the label specifications.

(4) For ***VXCOBOL***, EBCDIC indicates IBM format labels. If *int-2* is not specified it is assumed to be 2. ASCII is equivalent to STANDARD. NATIVE refers to Data General Format. If *int-1* is not specified it is assumed to be 3.

D.12. LINAGE Clause

D.12.1 Function

The LINAGE clause provides a means for specifying the depth of a logical page in terms of number of lines. It also provides for specifying the size of the top and bottom margins on the logical page, and the line number, within the page body, at which the footing area begins.

D.12.2 General Format

```
LINAGE IS { data-name-1 }          LINES [ WITH FOOTING AT { data-name-2 } ]
          { integer-1   }                                  { integer-2   }

    [ LINES AT TOP { data-name-3 } ] [ LINES AT BOTTOM { data-name-4 } ]
                   { integer-3   }                     { integer-4   }
```

D.12.3 Syntax Rules

(1) *Data-name-1, data-name-2, data-name-3*, and *data-name-4* must reference elementary unsigned numeric data items.

(2) *Data-name-1, data-name-2, data-name-3*, and *data-name-4* may be qualified.

(3) *Integer-2* must not be greater than *integer-1*.

(4) *Integer-3* and *integer-4* may be zero.

D.12.4 General Rules

(0) The associated file must have been specified with PRINTER or PRINTER-1 in the ASSIGN Clause of the SELECT statement. If no device or DISK is specified and the LINAGE clause is present, the file will be treated as if PRINTER was specified in the ASSIGN clause.

(1) The LINAGE clause provides a means for specifying the size of a logical page in terms of number of lines. The logical page size is the sum of the values referenced by each phrase except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrases are not specified, the values of these items are zero. If the FOOTING phrase is not specified, no end-of-page condition independent of the page overflow condition exists.

There is not necessarily any relationship between the size of the logical page and the size of a physical page.

(2) *Integer-1* or the value of the data item referenced by *data-name-1* specifies the number of lines that can be written and/or spaced on the logical page. The value must be greater than zero. That part of the logical page in which these lines can be written and/or spaced is called the page body.

(3) *Integer-2* or the value of the data item referenced by *data-name-2* specifies the line number within the page body at which the footing begins. The value must be greater than zero and not greater than *integer-1* or the value of the data item referenced by *data-name-1*.

The footing area comprises the area of the page body between the line represented by *integer-2* or the value of the data item referenced by *data-name-2* and the line represented by *integer-1* or the value of the data item referenced by *data-name-1*, inclusive.

(4) *Integer-3* or the value of the data item referenced by *data-name-3* specifies the number of lines that comprise the top margin on the logical page. The value may be zero.

(5) *Integer-4* or the value of the data item referenced by *data-name-4* specifies the number of lines that comprise the bottom margin on the logical page. The value may be zero.

(6) *Integer-1, integer-3*, and *integer-4*, if specified, are used at the time the file is opened by the execution of an OPEN statement with the OUTPUT phrase, to specify the number of lines that comprise each of the indicated sections of a logical page. *Integer-2*, if specified, is used at that time to define the footing area. These values are used for all logical pages written for that file during a given execution of the program.

(7) The values of the data items referenced by *data-name-1, data-name-3*, and *data-name-4*, if specified, are used as follows:

a. The values of the data items, at the time an OPEN statement with the OUTPUT phrase is executed for the file, are used to specify the number of lines that are to comprise each of the indicated sections for the first logical page.

b. The values of the data items, at the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, are used to specify the number of lines that are to comprise each of the indicated sections for the next logical page. (See the WRITE Statement.)

(8) The value of the data item referenced by *data-name-2*, if specified, at the time an OPEN statement with the OUTPUT phrase is executed for the file, is used to define the footing area for the first logical page. At the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, it is used to define the footing area for the next logical page.

(9) A LINAGE-COUNTER is generated by the presence of a LINAGE clause. The value in the LINAGE-COUNTER at any given time represents the line number at which the device is positioned within the current page body. The rules governing the LINAGE-COUNTER are as follows:

a. A separate LINAGE-COUNTER is supplied for each file described in the File Section whose file description entry contains a LINAGE clause.

b. LINAGE-COUNTER may be referenced only in Procedure Division statements; however, only the input-output control system may change the value of LINAGE-COUNTER. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by file-name when necessary.

c. LINAGE-COUNTER is automatically modified, according to the following rules, during the execution of a WRITE statement to an associated file:

1) When the ADVANCING PAGE phrase of the WRITE statement is specified, the LINAGE-COUNTER is automatically reset to one. During the resetting of LINAGE-COUNTER to the value one, the value of LINAGE-COUNTER is implicitly incremented to exceed the value specified by *integer-1* or the data item referenced by *data-name-1*.

2) When the ADVANCING *identifier-2* or *integer-1* phrase of the WRITE statement is specified, the LINAGE-COUNTER is incremented by *integer-1* or the value of the data item referenced by *identifier-2*.

3) When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value one. (See the WRITE Statement.)

4) The value of LINAGE-COUNTER is automatically reset to one when the device is repositioned to the first line that can be written on for each of the succeeding logical pages. (See the WRITE Statement.)

d. The value of LINAGE-COUNTER is automatically set to one at the time an OPEN statement with the OUTPUT phrase is executed for the associated file. An OPEN with the EXTEND phrase leaves the value of LINAGE-COUNTER undefined.

(10) Each logical page is contiguous to the next with no additional spacing provided.

(11) If the file connector associated with this file description entry is an external file connector, all file description entries in the run unit which are associated with this file connector must have:

a. A LINAGE clause, if any file description entry has a LINAGE clause.

b. The same corresponding values for *integer-1, integer-2, integer-3*, and *integer-4*, if specified.

c. The same corresponding external data items referenced by *data-name-1, data-name-2, data-name-3*, and *data-name-4*.

D.13. MERIT Clause (***VXCOBOL***)

D.13.1 Function

The MERIT clause allows record distribution to be optimized in an INFOS file.

D.13.2 General Format

MERIT IS *identifier*

D.13.3 Syntax Rules

(1) *Identifier* is an integer data item that specifies a merit factor from 1 to 32. Two volumes can have the same merit factor.

D.13.4 General Rules

(1) If a merit factor is given for a record, INFOS places the record on the first volume that has both available space and a merit factor equal to or less than the record's merit factor. If the system cannot find a volume that satisfies these criteria, it places the record on the volume with the lowest merit factor that is higher than the one specified.

(2) This clause is ignored.

D.14. PARTIAL RECORD Clause (***VXCOBOL***)

D.14.1 Function

The PARTIAL RECORD clause allows a frequently used portion of the record to be accessed with a key.

D.14.2 General Format

PARTIAL RECORD IS *identifier*

D.14.3 Syntax Rules

(1) *Identifier* is an alphanumeric data item that receives the partial record data. It receives the partial record on every operation that accesses a data record (unless the partial record is suppressed.)

D.14.4 General Rules

(1) The size of the data item specified by *identifier* determines the length that the partial record can have. This length cannot be larger than the maximum size of the partial record set at index or subindex creation time (up to 255). When a data record for this file is accessed, the partial record is returned to *identifier*.

(2) With INFOS II, the partial record is stored in the index with the key. With U/FOS, the partial record is stored as a second data record.

D.15. RECORD Clause (***ANSI 74*** and ***ANSI 85***)

D.15.1 Function

The RECORD clause specifies the number of character positions in a fixed length record, or specifies the range of character positions in a variable length record. If the number of character positions does vary, the clause specifies the minimum and maximum number of character positions.

D.15.2 General Format

Format 1 (fixed-length):

RECORD CONTAINS *integer-1* CHARACTERS

Format 2 (variable-length):

RECORD IS VARYING IN SIZE [[FROM *integer-2*] [TO *integer-3*] CHARACTERS]
[DEPENDING ON *data-name-1*]

Format 3 (fixed or variable length):

RECORD CONTAINS *integer-4* TO *integer-5* CHARACTERS

D.15.3 Syntax Rules

Format 1:

(1) No record description entry for the file may specify a number of character positions greater than *integer-1*.

(2) For ***VXCOBOL***, this format is for documentation purposes only.

Format 2:

(3) This format is not supported under ***VXCOBOL***.

(4) No record description entry for the file may specify a number of character positions less than *integer-2* or greater than *integer-3*.

(5) *Integer-3* shall be greater than *integer-2*.

(6) *Data-name-1* shall describe an elementary unsigned integer in working storage or linkage section.

(7) *Integer-2* shall be greater than zero.

(8) This format may not be specified if the RECORDING MODE clause is specified.

Format 3:

(9) For ***VXCOBOL***, this format is for documentation purposes only.

(10) *Integer-4* shall be greater than zero.

(11) *Integer-5* shall be greater than *integer-4*.

D.15.4 General Rules

All Formats:

(1) Each integer in a RECORD clause specifies a record size in terms of alphanumeric character positions.

(2) The implicit or explicit RECORD clause specifies the size of the records in the record area. The size of records on physical storage media may be different due to control information required by the operating environment.

(3) The size of each data record is specified in terms of the number of alphanumeric character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of the record is determined by the sum of the number of alphanumeric character positions in all fixed length elementary items plus the sum of the maximum number of alphanumeric character positions in any variable-length data item subordinate to the record.

(4) If the RECORD clause is not specified, an implicit format 1 or format 2 RECORD clause is assumed to be specified. This implicit RECORD clause is defined with the following characteristics:

a. Format 1 is implied when RECORDING MODE clause is absent or FIXED. *Integer-1* shall be the record size of the largest record description entry in this file description entry.

b. Format 2 is implied when the RECORDING MODE IS VARIABLE. *Integer-2* shall be the record size of the smallest record description entry in this file description entry, and *integer-3* shall be the largest record description entry in this file description entry. The DEPENDING ON phrase is assumed to be omitted.

Format 1:

(5) Format 1 is used to specify fixed length records. *Integer-1* specifies the number of character positions contained in each record in the file.

Format 2:

(6) Format 2 is used to specify variable-length records. *Integer-2* specifies the minimum number of alphanumeric character positions to be contained in any record of the file. *Integer-3* specifies the maximum number of alphanumeric character positions in any record of the file.

(7) The number of alphanumeric character positions associated with a record description is determined by the sum of the number of alphanumeric character positions in all elementary data items excluding redefinitions and renamings, plus any implicit FILLER due to synchronization. If a table is specified:

a. The minimum number of table elements described in the record is used in the summation above to determine the minimum number of alphanumeric character positions associated with the record description.

b. The maximum number of table elements described in the record is used in the summation above to determine the maximum number of alphanumeric character positions associated with the record description.

(8) If *integer-2* is not specified, the minimum number of alphanumeric character positions to be contained in any record of the file is equal to the least number of alphanumeric character positions described for a record in that file.

(9) If *integer-3* is not specified, the maximum number of alphanumeric character positions to be contained in any record of the file is equal to the greatest number of alphanumeric character positions described for a record in that file.

(10) If *data-name-1* is specified, the number of alphanumeric character positions in the record shall be placed into the data item referenced by *data-name-1* before any RELEASE, REWRITE, or WRITE statement is executed for the file.

(11) If *data-name-1* is specified, the execution of a DELETE, RELEASE, REWRITE, START, or WRITE statement or the unsuccessful execution of a READ or RETURN statement does not alter the content of the data item referenced by *data-name-1*.

(12) During the execution of a RELEASE, REWRITE, or WRITE statement, the number of alphanumeric character positions in the record is determined by the following conditions:

a. If *data-name-1* is specified, by the content of the data item referenced by *data-name-1*.

b. If *data-name-1* is not specified and the record does not contain a variable-occurrence data item, by the number of alphanumeric character positions in the record.

c. If *data-name-1* is not specified and the record does contain a variable-occurrence data item, by the sum of the fixed portion and that portion of the table described by the number of occurrences at the time of execution of the output statement.

d. If the file had been specified with a RECORD DELIMITER IS DATA-SENSITIVE or RECORD DELIMITER IS literal, by the first occurrence of a delimiter character or as determined by rules a) - c) if that number of alphanumeric character positions is less.

(13) If the number of alphanumeric character positions in the record to be written is less than integer-2 or greater than integer-3, then if a RELEASE, REWRITE, or WRITE statement is being executed, exception 185 condition is set to exist, and the execution of the RELEASE, REWRITE, or WRITE statement is unsuccessful with i-o status 92 (**ANSI 74**) or i-o status 44 (**ANSI 85**).

(14) If *data-name-1* is specified, after the successful execution of a READ or RETURN statement for the file, the contents of the data item referenced by data-name-1 will indicate the number of alphanumeric character positions in the record just read.

(15) If the INTO phrase is specified in the READ or RETURN statement, the number of alphanumeric character positions in the current record that participate as the sending operands in the implicit MOVE statement is determined by the following conditions:

a. If *data-name-1* is specified, by the content of the data item referenced by *data-name-1*.

b. If *data-name-1* is not specified, by the value that would have been moved into the data item referenced by *data-name-1* had *data-name-1* been specified.

If the number of alphanumeric character positions determined as above is zero, the record is a zero-length item.

(16) INDEXED and RELATIVE files are varying length within a fixed allocation and must be ICISAM version 7 files. SEQUENTIAL files are written with a varying length and format based on the RECORD DELIMITER clause of the SELECT statement.

Format 3

(17) Format 3 of the RECORD clause produces fixed-length records if the RECORDING MODE clause is absent or FIXED. Format 3 produces variable-length records if the RECORDING MODE is VARIABLE.
.

(18) When format 3 of the RECORD clause is used, *integer-4* and *integer-5* refer to the minimum number of alphanumeric characters in the smallest size data record and the maximum number of alphanumeric characters in the

largest size data record, respectively. However, in this case, the size of each data record is completely defined in the record description entry.

(19) If the number of alphanumeric character positions in the logical record to be written is less than *integer-4* or greater than *integer-5*, then if a RELEASE, REWRITE, or WRITE statement is being executed, the exception 185 is set to exist and the execution of the RELEASE, REWRITE, or WRITE statement is unsuccessful with I-O status 92 (**ANSI 74**) or 44 (**ANSI 85**).

D.16. RECORDING MODE Clause (*ANSI 74* and *ANSI 85*)

D.16.1 Function

The RECORDING MODE clause specifies whether a sequential disk file is have a fixed length record or a variable length record based on the specified record. This clause is obsolete; variable sequential files may be obtained with the RECORD DELIMITER IS BINARY LENGTH and RECORD IS VARYING clauses. The RECORDING MODE clause is an extension to ANSI COBOL.

D.16.2 General Format

RECORDING MODE IS { VARIABLE / FIXED } .

D.16.3 Syntax Rules

(1) RECORDING MODE is only allowed for sequential disk files.

(2) This clause may not be specified with the RECORD IS VARYING clause.

D.16.4 General Rules

(1) If this clause is not specified, RECORDING MODE IS FIXED is assumed.

D.17. RECORDING MODE Clause (*VXCOBOL*)

D.17.1 Function

The RECORDING MODE clause specifies the record format used in the file. The RECORDING MODE clause is an extension to ANSI COBOL.

D.17.2 General Format

```
RECORDING MODE IS
{ FIXED
  UNDEFINED [ RECORD LENGTH IS identifier-1 ]
  VARIABLE [ RECORD LENGTH IS identifier-1 ]
  DATA-SENSITIVE [ DELIMITER IS literal-1 ] [ RECORD LENGTH IS identifier-1 ]
  DYNAMIC RECORD LENGTH IS identifier-1 }
```

D.17.3 Syntax Rules

(1) *Identifier-1* is an integer data item that either specifies or receives a number of characters.

(2) *Literal-1* is a numeric literal specifying a character that delimits the end of a record, replacing the default delimiter.

(3) RECORDING MODE IS FIXED is the only format allowed for relative files.

(4) RECORDING MODE IS VARIABLE is the only format allowed for INFOS files.

(5) RECORDING MODE IS FIXED and RECORDING MODE IS VARIABLE are allowed for indexed files.

D.17.4 General Rules

(1) If this clause is not specified, RECORDING MODE IS FIXED is assumed for sequential and relative. RECORD MODE IS VARIABLE is assumed for indexed and INFOS files.

(2) If FIXED is specified, all records have the same number of characters, the length of which is determined by the size of the file's record area.

(3) If VARIABLE is specified, the maximum length for records can be specified in the RECORD LENGTH clause. No two records in the file need to be the same length. However, they cannot exceed the maximum length and they must never be 1. If a RECORD LENGTH clause is not used, the number of characters in a record determines the maximum length for that record. For an index file, the record length must always be large enough to include the RECORD key and all the ALTERNATE keys.

(4) If DYNAMIC is specified, the value of the data item specified in the RECORD LENGTH clause is used as the length of the record. Therefore the RECORD LENGTH clause must be specified when using DYNAMIC.

(5) If DATA-SENSITIVE is specified, the length of the record is determined by the occurrence of a special character (*literal-1*). If a delimiter character is not specified, carriage-return, form feed, null, or newline is used. The RECORD LENGTH clause can also be used to set a maximum length for a data-sensitive record. The delimiter should be counted as part of the record. DATA-SENSITIVE is ignored for printer files unless you use the DELIMITER IS clause to specify the delimiters.

(6) If undefined is specified, the file is read only as a sequence of binary bytes rather than a sequence of records.

(7) If all cases, if RECORD LENGTH is omitted, a record cannot be more than the length of the file's record area. If RECORD LENGTH is specified, the number of characters read from a record is returned. On output in variable record format, identifier will specify the number of characters to write.

E. WORKING-STORAGE SECTION

The Working-Storage Section is located in the Data Division of a source program. The Working-Storage Section describes records and subordinate data items which are not part of data files.

The Working-Storage Section is composed of the section header, followed by record description entries and/or data description entries for noncontiguous data items.

The general format of the Working-Storage Section is shown below.

```
WORKING-STORAGE SECTION.
[ 77-level-description-entry ]
[ record-description-entry   ] ...
```

E.1. Noncontiguous Working Storage

Items and constants in working storage which bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data description entry which begins with the special level-number, 77.

The following data clauses are required in each data description entry:

1. level-number 77
2. data-name
3. the PICTURE clause, the USAGE IS INDEX clause, or the USAGE IS POINTER (**ANSI 74and ANSI 85**) clause

Other data description clauses are optional and can be used to complete the description of the item if necessary.

E.2. Working Storage Records

Data elements in working storage which bear a definite hierarchical relationship to one another must be grouped into records according to the rules for formation of record descriptions. Data elements in the Working-Storage Section which bear no hierarchical relationship to any other data item may be described as records which are single elementary items. All clauses which are used in record descriptions in the File Section can be used in record descriptions in the Working-Storage Section.

E.3. Record Description Structure

A record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by the data-name or FILLER clause, if specified, followed by a series of independent clauses as required. A record description may have a hierarchical structure and therefore the clauses used within an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description and the elements allowed in a record description entry are explained in Concept of Levels and in The Data Description Entry.

E.4. Initial Values

The initial value of any data item in the Working-Storage Section except an index data item is specified by associating the VALUE clause with the data item. The initial value of any index data item or any data item not associated with a VALUE clause is undefined.

E.5. Data Description Entry

E.5.1 Function

A data description entry specifies the characteristics of a particular item of data.

E.5.2 General Format

Format 1:

```
level-number { data-name-1 | FILLER }                    [ IS EXTERNAL ]
    [ BLANK WHEN ZERO ]
    [ { JUSTIFIED | JUST } RIGHT ]
    [ OCCURS { integer TIMES
             | integer TO integer TIMES DEPENDING ON identifier }
        [ { ASCENDING | DESCENDING } KEY IS {data-name}... ] ...
        [ INDEXED BY { index-name }... ] ]
    [ { PICTURE | PIC } IS character-string ]
    [ REDEFINES data-name-2 ]
    [ [ SIGN IS ] { LEADING | TRAILING }                  [ SEPARATE CHARACTER ] ]
d   [ { SYNCHRONIZED | SYNC } [ LEFT | RIGHT ]                                  ]
    [ usage-clause ]
    [ VALUE IS { literal | NULL | VALID | OVERFLOW } ] .
```

Where *usage-clause* is:

For ***ANSI 74*** and ***ANSI 85***:

```
[ USAGE IS] { BINARY
            | COMPUTATIONAL | COMP
            | COMPUTATIONAL-3 | COMP-3
            | COMPUTATIONAL-5 | COMP-5
            | DISPLAY
            | INDEX
            | PACKED-DECIMAL
            | POINTER }
```

For ***VXCOBOL***:

```
                    COMPUTATIONAL | COMP
[ USAGE IS ] {  COMPUTATIONAL-3 | COMP-3  }
                    DISPLAY
                    INDEX
```

For ***ISQL***: add the following selections:

```
                    { CHARACTER } [ VARYING ]  [ ( integer-1 ) ]
                    { CHAR      }
                    DATE
                    INDICATOR
                    INT
                    INTEGER

                    YEAR ( integer-2 )  [ TO MONTH ]
                    MONTH ( integer-2 )
                    DAY ( integer-2 ) [ TO { HOUR   } ]
                                           { MINUTE }
                                           { SECOND [ ( integer-3 ) ] }
                    HOUR ( integer-2 ) [ TO { MINUTE   } ]
                                           { SECOND [ ( integer-3 ) ] }
                    MINUTE ( integer-2 ) [ TO SECOND [ ( integer-3 ) ] ]
                    SECOND ( integer-2 [ integer-3 ] )

                    NUMERIC ( integer-4 [ integer-5 ] )
                    SMALLINT
                    TIME [ ( integer-3 ) ]
                    TIMESTAMP [ ( integer-3 ) ]
```

Format 2:

```
66 data-name-1 RENAMES data-name-2 [{ THROUGH }                data-name-3 ] .
                                    { THRU    }
```

Format 3:

```
88 condition-name { VALUE IS  } { literal-1 [ { THROUGH } literal-2 ] } ...
                  { VALUES ARE }             { THRU    }
```

E.5.3 Syntax Rules

(1) Level number in Format 1 may be any number from 01 through 49 or 77.

(2) In Format 1, the *data-name-1* or FILLER clause, if specified, must immediately follow the level number. The REDEFINES clause, if specified, must immediately follow the *data-name-1* or FILLER clause if either is specified; otherwise, it must immediately follow the *level-number*. The remaining clauses may be written in any order.

(3) The EXTERNAL clause and the REDEFINES clause must not be specified in the same data description entry.

(4) *Data-name-1* must be specified for an entry containing the EXTERNAL clause or for record descriptions associated with a file description entry which contains the EXTERNAL clause.

(5) The PICTURE clause must be specified for every elementary item except an index or pointer data item in which case use of this clause is prohibited.

(6) The words THRU and THROUGH are equivalent.

(7) (***ISQL***) The words CHAR and CHARACTER are equivalent.

(8) (***ISQL***) The words INT and INTEGER are equivalent.

(9) The EXTERNAL clause may be specified only in data description entries in the Working-Storage Section whose level-number is 01 or 77.

E.5.4 General Rules

(1) The clauses SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO must not be specified except for an elementary item.

(2) Format 3 is used for each *condition-name*. Each *condition-name* requires a separate entry with level-number 88. Format 3 contains the name of the condition and the value, values, or range of values associated with the *condition-name*. The *condition-name* entries for a particular conditional variable must immediately follow the entry describing the item with which the *condition-name* is associated. A *condition-name* can be associated with any data description entry which contains a level-number except the following:

a. Another condition-name.

b. A level 66 item.

c. A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

d. An index or pointer data item.

(3) Multiple level 01 entries subordinate to any given level indicator (i.e., FD or SD) represent implicit redefinitions of the same area.

E.6. BLANK WHEN ZERO Clause

E.6.1 Function

The BLANK WHEN ZERO clause permits the blanking of an item when its value is zero.

E.6.2 General Format

BLANK WHEN ZERO

E.6.3 Syntax Rules

(1) The BLANK WHEN ZERO clause can be specified only for an elementary item whose PICTURE is specified as numeric or numeric edited.

(2) The numeric or numeric edited data description entry to which the BLANK WHEN ZERO clause applies must be described, either implicitly or explicitly, as USAGE IS DISPLAY.

E.6.4 General Rules

(1) When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero.

(2) When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

E.7. Data-Name or FILLER Clause

E.7.1 Function

A data-name specifies the name of the data item being described. The keyword FILLER may be used to specify a data item which is not referenced explicitly.

E.7.2 General Format

```
[ data-name-1 ]
[ FILLER      ]
```

E.7.3 Syntax Rules

(1) In the File, Working-Storage, and Linkage Sections, *data-name-1* or the keyword FILLER, if either is specified, must be the first word following the level-number in each data description entry.

E.7.4 General Rules

(1) If this clause is omitted, the data item being described is treated as though FILLER had been specified.

(2) The keyword FILLER may be used to name a data item. Under no circumstances can a FILLER item be referred to explicitly. However, the keyword FILLER may be used to name a conditional variable because such use does not require explicit reference to the data item itself, but only to the value contained therein.

E.8. EXTERNAL Clause

E.8.1 Function

The EXTERNAL clause specifies that a data item is external. A data item is external if the storage associated with that object is associated with the run unit rather than with any particular program within the run unit. The constituent data items and group data items of an external data record are available to every program in the run unit which describes that record.

E.8.2 General Format

IS EXTERNAL

E.8.3 Syntax Rules

(1) The EXTERNAL clause may be specified only in data description entries in the Working-Storage Section whose level number is 01 or 77. For ***VXCOBOL***, the EXTERNAL clause may be specified on a data description in the FILE SECTION as long as it is also specified on the FD of the file.

(2) In the same program, the data-name specified as the subject of the entry whose level-number is 01 or 77 that includes the EXTERNAL clause must not be the same data-name specified for any other data description entry which includes the EXTERNAL clause.

(3) The EXTERNAL clause shall not be specified for a data item of class pointer.

E.8.4 General Rules

(1) The data contained in the data description entry named by the data-name clause is external and may be accessed and processed by any program in the run unit which describes and optionally, redefines it subject to the following general rules.

(2) Within a run unit, if two or more programs describe the same external data record or elementary item, each record-name or data-name of the associated data description entries must be the same and the data descriptions must define the same number of standard format characters. The items must be of the same type. However, a program which describes an external record may contain a data description entry including the REDEFINES clause which redefines the complete external record, and this complete redefinition need not occur identically in other programs in the run unit.

(3) If the VALUE clause is specified for a data description entry with the EXTERNAL clause or subordinate to a data description entry with an EXTERNAL clause, then every program describing that external item must specify an identical VALUE clause on its declaration of the external item.

(4) On the call of a subprogram that contains the declaration of an EXTERNAL item, if any of the external data item's characteristics fail to match those of a previously loaded external item of the same name, **ICOBOL** will generate an exception status 1296 "External item in called program does not match existing item" and the call will fail.

E.9. JUSTIFIED Clause

E.9.1 Function

The JUSTIFIED clause permits alternate positioning of data within a receiving data item, specifically right justification.

E.9.2 General Format

```
{ JUSTIFIED }
{ JUST      }        RIGHT
```

E.9.3 Syntax Rules

(1) The JUSTIFIED clause can be specified only at the elementary item level.

(2) JUST is an abbreviation for JUSTIFIED.

(3) The JUSTIFIED clause cannot be specified for any data item described as numeric or for which editing is specified. (i.e., it can only be used with an unedited alphabetic or alphanumeric data item.)

(4) The JUSTIFIED clause must not be specified for an index data item.

E.9.4 General Rules

(1) When the receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the left-most characters are truncated. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the right-most character position in the data item with space fill for the left-most character positions.

(2) When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply.

E.10. Level-Number

E.10.1 Function

The level-number indicates the position of a data item within the hierarchical structure of a logical record. In addition, it is used to identify entries for working storage items, linkage items, condition-names and the RENAMES clause.

E.10.2 General Format

level-number

E.10.3 Syntax Rules

(1) A *level-number* is required as the first element in each data description entry.

(2) Data description entries subordinate to a FD or SD entry must have level-numbers with the values 01 through 49, 66, or 88.

(3) Data description entries in the Working-Storage Section and Linkage Section must have level-numbers 01 through 49, 66, 77, or 88.

(4) Data description entries in the Screen Section must have level-numbers 01 through 49.

(5) A level number in the range 01 through 09 may be specified as 1 through 9.

E.10.4 General Rules

(1) The level-number 01 identifies the first entry in each record description.

(2) Special level-numbers have been assigned to certain entries where there is no real concept of hierarchy:

a. Level-number 77 is assigned to identify noncontiguous working storage data items, noncontiguous linkage data items, and can be used only as described by Format 1 of the data description entry.

b. Level-number 66 is assigned to identify RENAMES entries and can be used only as described by Format 2 of the data description entry.

c. Level-number 88 is assigned to entries which define condition-names associated with a conditional variable and can be used only as described by Format 3 of the data description entry.

(3) Multiple level 01 entries subordinate to any given level indicator (i.e., FD or SD) represent implicit redefinitions of the same area.

E.11. OCCURS Clause

E.11.1 Function

The OCCURS clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts.

E.11.2 General Format

```
OCCURS { integer-2 TIMES
       { integer-1 TO integer-2 TIMES DEPENDING ON data-name-1 }

    [ { ASCENDING  } KEY IS { data-name-2 } ... ] ...                    [ INDEXED BY { inde
      { DESCENDING }
```

E.11.3 Syntax Rules

(1) The OCCURS clause must not be specified in a data description entry that has a level-number of 01, 66, 77, 88, or which has a variable occurrence data-item subordinate to it. For ***VXCOBOL***, the occurs clause may occur on a data description entry that has a level number or 01.

(2) *Data-name-1* and *data-name-2* may be qualified.

(3) The first specification of *data-name-2* must be the name of either the entry containing the OCCURS clause or an entry subordinate to the entry containing the OCCURS clause. Subsequent specification of *data-name-2* must be subordinate to the entry containing the OCCURS clause.

(4) *Data-name-2* must be specified without the subscripting that is normally required.

(5) *Integer-2* must be greater than zero.

(6) If *integer-1* is given it must be greater than or equal to zero, and *integer-2* must be greater than *integer-1* but less than or equal to 16,777,216.

(7) *Data-name-1* must describe an integer and its picture must not include the character P. The data item described by *data-name-1* must not occupy a character position within the range of the first character position defined by the data description entry containing the OCCURS clause and the last character position defined by the record description entry containing that OCCURS clause.

(8) If the OCCURS clause is specified in a data description entry included in a record description entry containing the EXTERNAL clause, data-name if specified, must reference a data item possessing the external attribute which is described in the same Data Division.

(9) A data description entry that contains a DEPENDING ON may only be followed, within that record description, by data description entries which are subordinate to it.

(10) The data item identified by *data-name-2* must not contain an OCCURS clause except when *data-name-2* is the subject of the entry.

(11) There must not be an entry that contains an OCCURS clause between the descriptions of the data items identified by data-names in the KEY IS phrase and the subject of the entry.

(12) An INDEXED BY phrase is required if the subject of this entry, or an entry subordinate to this entry, is to be referenced by indexing. The *index-name-1* identified by this phrase is not defined elsewhere since its allocation and format are dependent on the hardware and, not being data, cannot be associated with any data hierarchy.

(13) *Index-name-1* must be a unique word within the program.

E.11.4 General Rules

(1) Except for the OCCURS clause itself, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.

(2) If the DEPENDING ON phrase is not given, the value of integer-2 represents the exact number of occurrences of the subject entry. If the DEPENDING ON phrase is given, the number of occurrences of the subject entry is defined to be the value of the data item referenced by data-name-1. In this case the subject of the entry has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum number of occurrences. This does not imply that the length of the subject entry is variable, but that the number of occurrences is variable.

At the time the subject entry is referenced or any data item subordinate or superordinate to the subject of entry is referenced, the value of the data item referenced by identifier must fall within the range of integer-1 through integer-2. The contents of the data items whose occurrence numbers exceed the value of the data item referenced by data-name-1 are undefined. **ICOBOL** will raise an "Index out of range" error in this case.

(3) When a group data item, having subordinate to it an entry that specifies a DEPENDING ON, is referenced, the part of the table area used in the operation is determined as follows:

a. If the data item referenced by *data-name-1* is outside the group, only that part of the table area that is specified by the value of the data item referenced by *data-name-1* at the start of the operation will be used.

b. If the data item referenced by *data-name-1* is included in the same group and the group data item is referenced as a sending item, only that part of the table area that is specified by the value of the data item referenced by *data-name-1* at the start of the operation will be used in the operation. If the group is a receiving item, the maximum length of the group will be used. (This last sentence is different than how AOS/VS COBOL behaves.)

(4) When the KEY IS phrase is specified, the repeated data must be arranged in ascending or descending order according to the values contained in *data-name-3*. The ascending or descending order is determined according to the rules for the comparison of operands. The data-names are listed in their descending order of significance.

(5) At most ten (10) KEY IS phrases may be specified.

(6) If the OCCURS WITH DEPENDING is specified in a record description entry and the associated file description or sort-merge description entry contains the VARYING phrase of the RECORD clause, the records are variable length. If the DEPENDING ON phrase of the RECORD clause is not specified, the content of the data item referenced by *data-name-1* of the OCCURS clause must be set to the number of occurrences to be written before the execution of any RELEASE, REWRITE, or WRITE statement.

E.12. PICTURE Clause

E.12.1 Function

The PICTURE clause describes the general characteristics and editing requirements of an elementary item.

E.12.2 General Format

```
{ PICTURE }
{ PIC     }        IS character-string
```

E.12.3 Syntax Rules

(1) The PICTURE clause can be specified only at the elementary item level.

(2) A *character-string* consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item.

(3) The lowercase letters corresponding to the uppercase letters representing the PICTURE symbols A, B, P, S, V, X, Z, CR, and DB are equivalent to their uppercase representations in a PICTURE character-string. All other lowercase letters are not equivalent to their corresponding uppercase representations.

(4) The maximum number of characters allowed in the character-string is 30.

(5) The PICTURE clause must be specified for every elementary item except an index data item. In that case the use of this clause is prohibited.

(6) PIC is an abbreviation for PICTURE.

(7) The asterisk, when used as the zero suppression symbol, and the clause BLANK WHEN ZERO may not appear in the same entry.

(8) In the Screen section, unless the SIGN IS phrase is specified, the S PICTURE character is ignored by the compiler to be consistent with older versions of Interactive COBOL.

(9) In the Screen section, the PICTURE symbols P, V, CR, and DB can only be used with output (FROM) fields.

E.12.4 General Rules

(1) There are five categories of data that can be described with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited.

(2) To define an item as **alphabetic**:

a. Its PICTURE character-string can contain only the symbol `A'; and

b. Its content, when represented in standard data format, must be one or more alphabetic characters.

(3) To define an item as **numeric**:

a. Its PICTURE character-string can contain only the symbols `9', `P', `S', and `V'. The number of digit positions that can be described by the PICTURE character-string must range from 1 to 18 inclusive; and

b. If unsigned, its content when represented in standard data format must be one or more numeric characters; if signed, the item may also contain a `+', `-', or other representation of an operational sign.

(4) To define an item as **alphanumeric**:

a. Its PICTURE character-string is restricted to certain combinations of the symbols `A', `X', `9', and the item is treated as if the character-string contained all `X's. A PICTURE character-string which contains all `A's or all `9's does not define an alphanumeric item, and;

b. Its content, when represented in standard data format, must be one or more characters in the computer's character set.

(5) To define an item as **alphanumeric edited**:

a. Its PICTURE character-string is restricted to certain combinations of the following symbols: `A', `X', `9', `B', `0', and `/'; and must contain at least one `A' or `X' and must contain at least one `B' or `0' (zero) or `/' (slant).

b. Its content when represented in standard data format must be two or more characters in the computer's character set.

(6) To define an item as **numeric edited**:

a. Its PICTURE character-string is restricted to certain combinations of the symbols `B', `/', `P', `V', `Z', `0', `9', `,', `.', `*', `+', `-', `CR', `DB', and the currency symbol. The allowable combinations are determined from the order of precedence of symbols and the editing rules; and

1) The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 18 inclusive; and

2) The character-string must contain at least one `0', `B', `/', `Z', `*', `+', `,', `.', `-', `CR', `DB', or the currency symbol.

b. The content of each of the character positions must be consistent with the corresponding PICTURE symbol.

(7) The size of an elementary item, where size means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols that represent character positions. An unsigned nonzero integer which is enclosed in parentheses following the symbols `A', `,', `X', `9', `P', `Z', `*,' `B', `/', `0', `+', `-', or the currency symbol indicates the number of consecutive occurrences of the symbol. The following symbols may appear only once in a given PICTURE: `S', `V', `.', `CR', and `DB'.

(8) The functions of the symbols, used to describe an elementary item are explain as follows:

A Each `A' in the character-string represents a character position which can contain only an alphabetic character and is counted in the size of the item.

B Each `B' in the character-string represents a character position into which the space character will be inserted and is counted in the size of the item.

P Each `P' in the character-string indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character `P' is not counted in the size of the data item. Scaling position characters are counted in

determining the maximum number of digit positions (18) in numeric edited items or numeric items. The scaling position character. `P' can appear only as a continuous string of `P's in the left-most or right-most digit positions within a PICTURE character-string; since the scaling position character `P' implies an assumed decimal point (to the left of `P's if `P's are left-most PICTURE symbols and to the right if `P's are right-most PICTURE symbols), the assumed decimal point symbol `V' is redundant as either the left-most or right-most character within such a PICTURE description. The symbol `P' and the insertion symbol ` .' (period) cannot both occur in the same PICTURE character-string.

In certain operations that reference a data item whose PICTURE character-string contains the symbol `P', the algebraic value of the data item is used rather than the actual character representation of the data item. This algebraic value assumes the decimal point in the prescribed location and zero in place of the digit position specified by the symbol `P'. The size of the value is the number of digit positions represented by the PICTURE character-string. These operations are any of the following:

a. Any operation requiring a numeric sending operand.

b. A MOVE statement where the sending operand is numeric and its PICTURE character-string contains the symbol `P'.

c. 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.

d. A comparison operation where both operands are numeric.

In all other operations the digit positions specified with the symbol `P' are ignored and are not counted in the size of the operand.

S The `S' is used in a character-string to indicate the presence, but neither the representation nor, necessarily, the position of an operational sign; it must be written as the left-most character in the PICTURE. The `S' is not counted in determining the size (in terms of standard data format characters) of the elementary item unless the entry is subject to a SIGN clause which specifies the optional SEPARATE CHARACTER phrase.

V The `V' is used in a character-string to indicate the location of the assumed decimal point and may only appear once in a character-string. The `V' does not represent a character position and therefore is not counted in the size of the elementary item. When the assumed decimal point is to the right of the right-most symbol in the string representing a digit position or scaling position, the `V' is redundant.

X Each `X' in the character-string is used to represent a character position which contains any allowable character from the computer's character set and is counted in the size of the item.

Z Each `Z' in a character-string may only be used to represent the left-most leading numeric character positions which will be replaced by a space character when the content of that character position is a leading zero. Each `Z' is counted in the size of the item.

9 Each `9' in the character-string represents a digit position which contains a numeric character and is counted in the size of the item.

0 Each `0' (zero) in the character-string represents a character position into which the character zero will be inserted. The `0' is counted in the size of the item.

/ Each `/' (slant) in the character-string represents a character position into which the slant character will be inserted. The `/' is counted in the size of the item.

, Each `,' (comma) in the character-string represents a character position into which the character `,' will be inserted. This character position is counted in the size of the item.

. When the symbol `.' (period) appears in the character-string it is an editing symbol which represents the decimal point for alignment purposes and, in addition, represents a character position into which the character `.' will be inserted. The character `.' is counted in the size of the item. For a given program the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in a PICTURE clause.

+ - CR DB These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data item.

* Each `*' (asterisk) in the character-string represents a leading numeric character position into which an asterisk will be placed when the content of that position is a leading zero. Each `*' is counted in the size of the item.

cs The currency symbol in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol in a character-string is represented by either the currency sign ($) or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in the size of the item.

E.12.5 Editing Rules

(1) There are two general methods of performing editing in the PICTURE clause, either by insertion or by suppression and replacement. There are four types of insertion editing available. They are:

a. Simple insertion
b. Special insertion
c. Fixed insertion
d. Floating insertion

There are two types of suppression and replacement editing:

a. Zero suppression and replacement with spaces
b. Zero suppression and replacement with asterisks

(2) The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. The following table specifies which type of editing may be performed upon a given category:

CATEGORY	TYPE OF EDITING
Alphabetic	None
Numeric	None
Alphanumeric	None
Alphanumeric edited	Simple insertion `0', `B', and `/'
Numeric edited	All, subject to rule 3 below

TABLE 4. PICTURE Editing

(3) Floating insertion editing and editing by zero suppression and replacement are mutually exclusive in a PICTURE clause. Only one type of replacement may be used with zero suppression in a PICTURE clause.

(4) Simple insertion editing. The `,' (comma), `B' (space), `0' (zero), and `/' (slant) are used as the insertion characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted. If the insertion character `,' (comma) is the last symbol in the PICTURE character-string, the PICTURE clause must be the last clause of the data description entry and must be immediately followed by the separator period. This results in the combination of `,.' appearing in the data description entry, or, if the DECIMAL POINT IS COMMA clause is used, in two consecutive periods.

(5) Special insertion editing. The `.' (period) is used as the insertion character. In addition to being an insertion character it also represents the decimal point for alignment purposes. The insertion character used for the actual

decimal point is counted in the size of the item. The use of the assumed decimal point, represented by the symbol `V' and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is disallowed. If the insertion character is the last symbol in the PICTURE character-string, the PICTURE clause must be the last clause of that data description entry and must be immediately followed by the separator period. This results in two consecutive periods appearing in the data description entry, or in the combination of `,.' if the DECIMAL-POINT IS COMMA clause is used. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string.

(6) Fixed insertion editing. The currency symbol and the editing sign control symbols `+', `-', `CR', and `DB' are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols `CR' or `DB' are used they represent two character positions in determining the size of the item and they must represent the right-most character positions that are counted in the size of the item. If these character positions contain the symbols `CR' or `DB', the uppercase letters are the insertion characters. The symbol `+' or `-' when used, must be either the left-most or right-most character position to be counted in the size of the item. The currency symbol must be the left-most character position to be counted in the size of the item except that it can be preceded by either a `+' or a `-' symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character-string. Editing sign control symbols produce the following results depending upon the value of the data item:

EDITING SYMBOL IN PICTURE CHARACTER-STRING	RESULT	
	DATA ITEM POSITIVE OR ZERO	DATA ITEM NEGATIVE
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

TABLE 5. Sign Control in Fixed PICTURE Editing

(7) Floating insertion editing. The currency symbol and editing sign control symbols `+' and `-' are the floating insertion characters and as such are mutually exclusive in a given PICTURE character-string.

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the floating insertion characters. This string of floating insertion characters may contain any of the simple insertion characters or have simple insertion characters immediately to the right of this string. These simple insertion characters are part of the floating string. When the floating insertion character is the currency symbol, this string of floating insertion characters may have the fixed insertion characters `CR' and `DB' immediately to the right of this string.

The left-most character of the floating insertion string represents the left-most limit of the floating symbols in the data item. The right-most character of the floating string represents the right-most limit of the floating symbols in the data item.

The second floating character from the left represents the left-most limit of the numeric data that can be stored in the data item. Nonzero numeric data may replace all the characters at or to the right of this limit.

In a PICTURE character-string, there are only two ways of representing floating insertion editing. One way is to represent any or all of the leading numeric character positions on the left of the decimal point by the insertion character. The other way is to represent all of the numeric character positions in the PICTURE character-string by the insertion character.

If the insertion character positions are only to the left of the decimal point in the PICTURE character-string, the result is that a single floating insertion character will be placed into the character position immediately preceding either the decimal point or the first nonzero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string. The character positions preceding the insertion character are replaced with spaces.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, at least one of the insertion characters must be to the left of the decimal point.

When the floating insertion character is the editing control symbol `+' or `-' the character inserted depends upon the value of the data item:

EDITING SYMBOL IN PICTURE CHARACTER-STRING	RESULT	
	DATA ITEM POSITIVE OR ZERO	DATA ITEM NEGATIVE
+	+	-
-	space	-

TABLE 6. Sign Control in Floating PICTURE Editing

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon the value of the data. If the value is zero the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of nonfloating insertion characters being edited into the receiving data item, plus one for the floating insertion character. If truncation does occur, the value of the data that is used for editing is the value after truncation.

(8) Zero suppression editing. The suppression of leading zeros in numeric character positions is indicated by the use of the alphabetic character `Z' or the character `*' (asterisk) as suppression symbols in a PICTURE character-string. These symbols are mutually exclusive in a given PICTURE character-string. Each suppression symbol is counted in determining the size of the item. If `Z' is used the replacement character will be the space and if the asterisk is used, the replacement character will be `*'.

Zero suppression and replacement is indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the associated character position in the data contains a leading zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

In a PICTURE character-string, there are only two ways of representing zero suppression. One way is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. The other way is to represent all of the numeric character positions in the PICTURE character-string by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

If all numeric character positions in the PICTURE character-string are represented by suppression symbols and the value of the data is not zero the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero and the suppression symbol is `Z', the entire data item, including any editing characters, is spaces. If the value is zero and the suppression symbol is `*' the entire data item, including any insertion editing symbols except the actual decimal point, will be `*'. In this case, the actual decimal point will appear in the data item.

(9) The symbols `+', `-', `*', `Z', and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character-string.

E.12.6 Precedence Rules

The following table shows the order of precedence when using characters as symbols in a character-string. An `X' at an intersection indicates that the symbol(s) at the top of the column may precede (but not necessarily immediately), in a given character-string, the symbol(s) at the left of the row. Two arguments appearing together indicate that the symbols are mutually exclusive. The currency symbol is indicated by the symbol `cs'.

At least one of the symbols `A', `X', `Z', `9', or `*', or at least two occurrences of one of the symbols `+', `-', or `cs' must be present in a PICTURE character-string.

Nonfloating insertion symbols `+' and `-', floating insertion symbols `Z', `*', `+', `-', and `cs', and other symbol `P' appear twice in the PICTURE character precedence chart in the table. The left-most column and upper-most row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the chart represents its use to the right of the decimal point position.

First Symbol		Non-floating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
Second Symbol		B	0	/	,	.	+ -	+ -	CR DB	cs	Z *	Z *	+ -	+ -	cs	cs	9	A X	S	V	P	P
Non-floating Insertion Symbols	B	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	0	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	/	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	,	x	x	x	x	x	x			x	x	x	x	x	x	x	x			x		x
	.	x	x	x	x		x			x	x		x		x		x					
	+ -																					
	+ -	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	CR DB	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	cs						x															
Floating Insertion Symbols	Z *	x	x	x	x		x			x	x											
	Z *	x	x	x	x	x	x			x	x	x								x		x
	+ -	x	x	x	x					x			x									
	+ -	x	x	x	x	x				x			x	x						x		x
	cs	x	x	x	x		x								x							
	cs	x	x	x	x	x	x								x	x				x		x
Other Symbols	9	x	x	x	x	x	x			x	x		x		x		x	x	x	x		x
	A X	x	x	x													x	x				
	S																					
	V	x	x	x	x		x			x	x		x		x		x		x		x	
	P	x	x	x	x		x			x	x		x		x		x		x		x	
	P						x			x										x	x	

TABLE 7. PICTURE Precedence Rules

Note: When two of the same symbols appear twice in the chart, the left-most column and upper-most row symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the chart represents its use to the right of the decimal point position.

E.13. REDEFINES Clause

E.13.1 Function

The REDEFINES clause allows the same computer storage area to be described by different data description entries.

E.13.2 General Format

level-number [**data-name-1** / **FILLER**] **[REDEFINES *data-name-2*]**

> **Note:** Level-number, data-name-1, and FILLER are shown in the above format to improve clarity. Level-number, data-name-1, and FILLER are not part of the REDEFINES clause.

E.13.3 Syntax Rules

(1) The REDEFINES clause, when specified, must immediately follow the subject of the entry.

(2) The level-numbers of *data-name-2* and the subject of the entry must be identical, but must not be 66 or 88.

(3) This clause must not be used in level 01 entries in the File Section.

(4) The data description entry for *data-name-2* cannot contain an OCCURS clause. However, *data-name-2* may be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to *data-name-2* in the REDEFINES clause may not be subscripted. Neither the original definition nor the redefinition can include a variable occurrence item.

(5) *Data-name-2* must not be qualified even if it is not unique; no ambiguity of reference exists in this case because of the required placement of the REDEFINES clause within the source program.

(6) Multiple redefinitions of the same character positions are permitted. Multiple redefinitions of the same character positions must all use the data-name of the entry that originally defined the area.

(7) The entries giving the new description of the character positions must not contain any VALUE clauses, except in condition-name entries.

(8) No entry having a level-number numerically lower than the level-number of *data-name-2* and the subject of the entry may occur between the data description entries of *data-name-2* and the subject of the entry.

(9) The entries giving the new descriptions of the character positions must follow the entries defining the area of *data-name-2*, without intervening entries that define new character positions.

(10) *Data-name-2* may be subordinate to an entry which contains a REDEFINES clause.

(11) If the data item referenced by *data-name-2* is either declared to be an external data record or is specified with a level-number other than 01, the number of character positions it contains must be equal to the number of character positions in the data item referenced by the subject of this entry. If the data-name referenced by *data-name-2* is specified with a level-number of 01 and is not declared to be an external data record, there is no such constraint.

E.13.4 General Rules

(1) Storage allocation starts at *data-name-2* and continues over a storage area sufficient to contain the number of character positions in the data item referenced by the *data-name-1* or FILLER clause.

(2) When the same character position is defined by more than one data description entry, the data-name associated with any of those data description entries can be used to reference that character position.

E.14. RENAMES Clause

E.14.1 Function

The RENAMES clause permits alternative, possibly overlapping, groupings of elementary items.

E.14.2 General Format

66 *data-name-1* RENAMES *data-name-2* [{ THROUGH / THRU } *data-name-3*] .

Note:	Level-number 66 and data-name-1 are shown in the above format to improve clarity. Level-number 66 and data-name-1 are not part of the RENAMES clause.

E.14.3 Syntax Rules

(1) Any number of RENAMES entries may be written for a logical record.

(2) All RENAMES entries referring to data items within a given logical record must immediately follow the last data description entry of the associated record description entry.

(3) *Data-name-1* cannot be used as a qualifier, and can only be qualified by the names of the associated level 01, FD, or SD entries. Neither *data-name-2* nor *data-name-3* may have an OCCURS clause in its data description entry nor be subordinate to an item that has an OCCURS clause in its data description entry.

(4) *Data-name-2* and *data-name-3* must be names of elementary items or groups of elementary items in the same logical record, and cannot be the same data-name. A 66 level entry cannot rename another 66 level entry nor can it at rename a 77, 88, or 01 level entry.

(5) *Data-name-2* and *data-name-3* may be qualified.

(6) None of the items within the range, including *data-name-2* and *data-name-3*, if specified, can be variable occurrence data items.

(7) The words THROUGH and THRU are equivalent.

(8) The beginning of the area described by *data-name-3* must not be to the left of the beginning of the area described by *data-name-2*. The end of the area described by *data-name-3* must be to the right of the end of the area described by *data-name-2*. *Data-name-3*, therefore, cannot be subordinate to *data-name-2*.

E.14.4 General Rules

(1) When *data-name-3* is specified, *data-name-1* is a group item which includes all elementary items starting with *data-name-2* (if *data-name-2* is an elementary item) or the first elementary item in *data-name-2* (if *data-name-2* is a group item), and concluding with *data-name-3* (if *data-name-3* is an elementary item) or the last elementary item in *data-name-3* (if *data-name-3* is a group item).

(2) When *data-name-3* is not specified, all of the data attributes of *data-name-2* become the data attributes for *data-name-1*.

E.15. SIGN Clause

E.15.1 Function

The SIGN clause specifies the position and the mode of representation of the operational sign when it is necessary to describe these properties explicitly.

E.15.2 General Format

```
[ SIGN IS ] { LEADING  }              [ SEPARATE CHARACTER ]
            { TRAILING }
```

E.15.3 Syntax Rules

(1) The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character `S'.

(2) The numeric data description entries to which the SIGN clause applies must be described, implicitly or explicitly, as USAGE IS DISPLAY.

(3) If the CODE-SET clause is specified in a file description entry, any signed numeric data description entries associated with that file description entry must be described with the SIGN IS SEPARATE clause.

E.15.4 General Rules

(1) The optional SIGN clause, if present, specifies the position and the mode of representation of the operational sign for the numeric data description entry to which it applies, or for each numeric data description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character `S'; the `S' indicates the presence of, but neither the representation nor, necessarily, the position of the operational sign.

(2) If a SIGN clause is specified in a group item subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate group item takes precedence for that subordinate group item.

(3) If a SIGN clause is specified in an elementary numeric data description entry subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate elementary numeric data description entry takes precedence for that elementary numeric data item.

(4) A numeric data description entry whose PICTURE contains the character `S', but to which no optional SIGN clause applies, has an operational sign, but neither the representation, nor, necessarily, the position of the operational sign is specified by the character `S'. For items whose USAGE IS DISPLAY, the default operational sign is the same as SIGN IS TRAILING. For items whose USAGE IS COMPUTATION, the operational sign is inherent in the binary representation of the value. General rules 5 through 7 do not apply to such default signed numeric data items.

(5) If the optional SEPARATE CHARACTER phrase is not present, then:

a. The operational sign will be presumed to be associated with the leading (or, respectively, trailing) digit position of the elementary numeric data item, more commonly called over punched.

b. The letter `S' in a PICTURE character-string is not counted in determining the size of the item (in terms of standard data format characters).

c. The table below defines the valid sign(s) for data items.

Digit	Positive	Negative
0	{ <173>	} <175>
1	A <101>	J <112>
2	B <102>	K <113>
3	C <103>	L <114>
4	D <104>	M <115>
5	E <105>	N <116>
6	F <106>	O <117>
7	G <107>	P <120>
8	H <110>	Q <121>
9	I <111>	R <122>

TABLE 8. SIGN Overpunch Characters

(6) If the optional SEPARATE CHARACTER phrase is present, then:

a. The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.

b. The letter `S' in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).

c. The operational signs for positive and negative are the standard data format characters `+' and `-' respectively.

(7) Every numeric data description entry whose PICTURE contains the character `S' is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

E.16. SYNCHRONIZED Clause

E.16.1 Function

The SYNCHRONIZED clause specifies the alignment of an elementary item on the natural boundaries of the computer memory.

E.16.2 General Format

E.16.3 Syntax Rules

(1) This clause may only appear with an elementary item.

(2) SYNC is an abbreviation for SYNCHRONIZED.

E.16.4 General Rules

(1) The SYNCHRONIZED clause is used for documentation only. All data items within a record are aligned on the next available byte in storage.

(2) All 01 and 77 level items are aligned on an even byte boundary. This default alignment may be overridden with the -B compiler switch.

E.17. USAGE Clause

E.17.1 Function

The USAGE clause specifies the format of a data item in the computer storage.

E.17.2 General Format (***ANSI 74*** and ***ANSI 85***)

```
[ USAGE IS ] { BINARY                          }
             { COMPUTATIONAL | COMP            }
             { COMPUTATIONAL-3 | COMP-3        }
             { COMPUTATIONAL-5 | COMP-5        }
             { DISPLAY                         }
             { INDEX                           }
             { PACKED-DECIMAL                  }
             { POINTER                         }
```

E.17.3 General Format (***VXCOBOL***)

```
[ USAGE IS ] { COMPUTATIONAL | COMP            }
             { COMPUTATIONAL-3 | COMP-3        }
             { DISPLAY                         }
             { INDEX                           }
```

E.17.4 Syntax Rules

(1) A USAGE clause specifying BINARY, COMPUTATION-5, PACKED-DECIMAL, or POINTER may not be used with the ***VXCOBOL*** dialect.

(2) The USAGE clause may be written in any data description entry with a level-number other than 66 or 88.

(3) If the USAGE clause is written in the data description entry for a group item, it may also be written in the data description entry for any subordinate elementary item or group item, but the same usage must be specified in both entries.

(4) An elementary data item whose declaration contains, or an elementary data item subordinate to a group item whose declaration contains, a USAGE clause specifying BINARY, COMPUTATIONAL, COMPUTATIONAL-3, COMPUTATIONAL-5, or PACKED-DECIMAL, must be declared with a PICTURE character-string that describes a numeric item, i.e., a PICTURE character-string that contains only the symbols `P', `S', `V', and `9'.

(5) COMP is an abbreviation for COMPUTATIONAL.

(6) COMP-3 is an abbreviation for COMPUTATIONAL-3.

(7) COMP-5 is an abbreviation for COMPUTATIONAL-5.

(8) An index data item can be referenced explicitly only in a SET statement, a relation condition, the USING phrase of a Procedure Division header, or the USING phrase of a CALL statement.

(9) The BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, SYNCHRONIZED, and VALUE clauses must not be specified for data items whose usage is INDEX.

(10) An elementary data item described with a USAGE IS INDEX or USAGE IS POINTER clause must not be a conditional variable.

(11) An elementary data item described with a USAGE IS POINTER must contain no other data description clauses other than VALUE IS NULL.

E.17.5 General Rules

(1) If the USAGE clause is written at a group level, it applies to each elementary item in the group.

(2) The USAGE clause specifies the manner in which a data item is represented in the storage of a computer. It may affect the use of the data item, and the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.

(3) The USAGE IS BINARY or COMPUTATIONAL clause specifies a twos-complement big-endian binary representation of the numeric item in the storage of the computer. The table below lists the bytes required to store BINARY and COMPUTATIONAL items.

VXCOBOL	ANSI 74 and ANSI 85		
Number of Decimal Digits	Number of Decimal Digits		Bytes Required
	Unsigned	Signed	
1-2	1-2	1-2	1
3-4	3-4	3-4	2
5-6	5-7	5-6	3
7-9	8-9	7-9	4
10-11	10-12	10-11	5
12-14	13-14	12-14	6
15-16	15-16	15-16	7
17-18	17-18	17-18	8

TABLE 9. BINARY & COMPUTATIONAL Storage Allocation

(4) The USAGE IS DISPLAY clause (whether specified explicitly or implicitly) specifies that a standard data format is used to represent a data item in the storage of the computer, and that the data item is aligned on a character boundary. The data is stored as ASCII characters in bytes.

(5) If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, the usage is implicitly DISPLAY.

(6) The USAGE IS COMPUTATIONAL-3 and PACKED-DECIMAL clauses specify that a radix of 10 (packed decimal) is used to represent a numeric item in the storage of the computer. Furthermore, this clause specifies that each digit position must occupy the minimum possible configuration in computer storage. COMPUTATIONAL-3 and PACKED-DECIMAL items are stored most significant digit first as a string of 4-bit half-bytes (nibbles). Each nibble except the rightmost, contains a hexadecimal digit of 0 through 9; the remaining nibble contains a hexadecimal C if the data is positive or unsigned or D if the data is negative. The sign nibble is always present as the last nibble. Because there must be an even number of nibbles (i.e., you cannot store a half-byte) and a sign nibble is always stored, the number of digits stored is always rounded up to an odd number. Thus a PIC 99 is stored the same as a PIC 999.

(7) The usage is COMPUTATIONAL-5 clause specifies a twos-complement binary representation of the numeric item in the storage of the computer. The format of a COMPUTATIONAL-5 item differs from that of a COMPUTATIONAL item in that in is stored in an order that is natural to the host computer. On "*big-endian*" machines, data is stored with high-order bytes at the lowest addresses and successively lower-order bytes at successively higher addresses. On "*little-endian*" machines, data is stored in the reverse order, i.e., the lower the address the lower the significance of the byte. For example, a computational item with a four byte hexadecimal

value of 12 34 56 78 would be stored as 12 34 56 78 on a "*big-endian*" machine and as 78 56 34 12 on a "*little-endian*" machine. Most RISC processors are "*big-endian*" and most Intel processors are "*little-endian*".

NOTE: Data stored in a COMPUTATIONAL-5 field may not be transportable to a different machine since different machines have different byte orderings.

The number of bytes required to store COMPUTATIONAL-5 items is described in the following table. Storage does not differ between signed and unsigned items.

Number of Decimal Digits	Bytes Required
1-2	1
3-4	2
5-9	4
10-18	8

TABLE 10. COMPUTATIONAL-5 Storage Allocation

(8) The USAGE IS INDEX clause specifies that a data item is an index data item and contains a value which must correspond to an occurrence number of a table element. INDEX items are represented internally as 4-byte unsigned items.

(9) The USAGE IS POINTER clause specifies a data-item in which the address of a data item can be stored. A pointer item requires 4 bytes. The format of the item is machine dependent. USAGE IS POINTER data items have their values assigned by the SET or INITIALIZE statements and may appear in relational conditions for equality and inequality.

(10) When a MOVE statement or an input-output statement that references a group item that contains an index data item or a pointer data item is executed, no conversion of the data item takes place.

(11) The ON SIZE condition is processed as follows for the various usages:

a. For a BINARY item, the number of digits used in the check is based on the picture specified.

b. For a COMPUTATIONAL item, the check is based on the picture specified except for **ANSI 74** where it is based on storage size rather than the picture. (The -G p and -G b compiler switches allow COMPUTATIONAL items to be size checked based on picture and storage respectively thus allowing the default behavior to be overridden.)

c. For a PACKED-DECIMAL item, the number of digits used in the check is based on the picture specified.

d. For a COMPUTATIONAL-3 item, the check is based on storage size rather than the picture; i.e., it uses the rounded-up digit count as explained in rule 8.

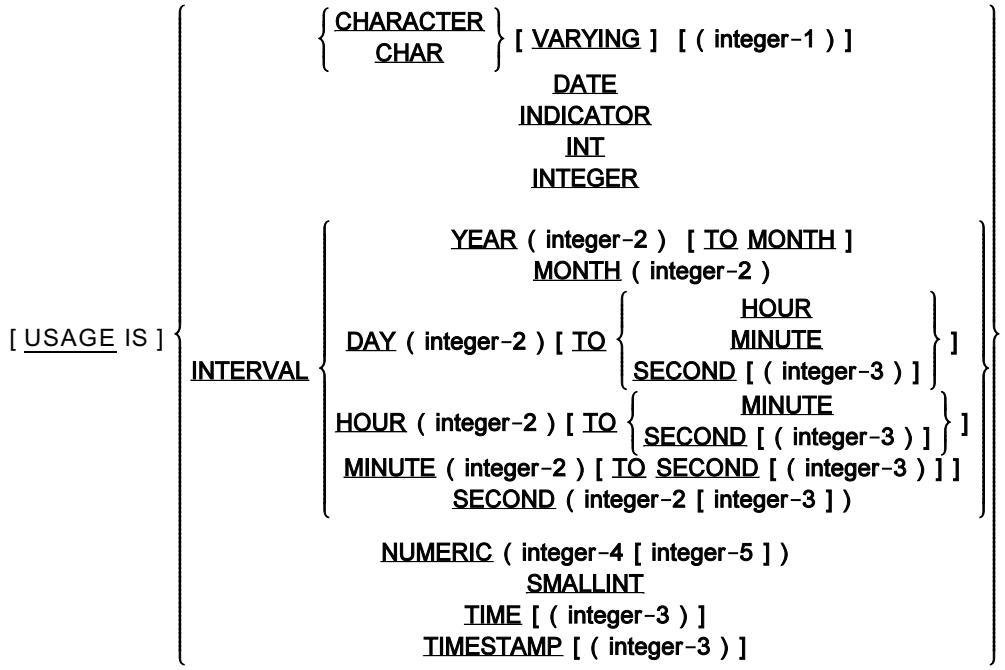
e. For a COMPUTATIONAL-5 item, the check is based on storage size rather than picture.

E.18. USAGE Clause (***ISQL***)

E.18.1 Function

This USAGE clause specifies the format of a data item in the computer storage when used with the ***ISQL*** feature set..

E.18.2 General Format



E.18.3 Syntax Rules

(1) A USAGE clause specifying CHARACTER, DATE, INDICATOR, INTEGER, INTERVAL, NUMERIC, SMALLINT, TIME, or TIMESTAMP is available only when the ***ISQL*** feature-set is enabled and appear as USAGE options that are in addition to certain dialect-specific options.

(2) The USAGE clause may be written in any data description entry with a level-number other than 66 or 88.

(3) A USAGE clause specifying CHARACTER, DATE, INDICATOR, INTEGER, INTERVAL, NUMERIC, SMALLINT, TIME, or TIMESTAMP must not be specified at the group level.

(4) If the USAGE clause is written in the data description entry for a group item, it may also be written in the data description entry for any subordinate elementary item or group item, but the same usage must be specified in both entries.

(5) CHAR is an abbreviation for CHARACTER.

(6) INT is an abbreviation for INTEGER.

(7) The BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, and SYNCHRONIZED clauses must not be specified for data items whose usage is CHARACTER, DATE, INDICATOR, INTEGER, INTERVAL, NUMERIC, SMALLINT, TIME, or TIMESTAMP.

(8) The value of *integer-1* must be greater than zero and less than or equal to 65535. If *integer-1* is omitted it is assumed to have a value of one.

(9) The value of *integer-2* must be greater than zero and less than the values specified in the general rules below.

(10) The value of *integer-3* must be greater than zero and less than or equal to six.

(11) The value of *integer-4* must be greater than zero and less than or equal to 18.

(12) The value of *integer-5* must be greater than or equal to zero and less than or equal to the value of *integer-4*. If *integer-5* is omitted it is assumed to have a value of zero.

E.18.4 General Rules

(1) If the USAGE clause is written at a group level, it applies to each elementary item in the group.

(2) The USAGE clause specifies the manner in which a data item is represented in the storage of a computer. It may affect the use of the data item, and the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.

(3) The USAGE IS CHARACTER clause specifies a data-item which can store an SQL character string value. The value of *integer-1* specifies the number of characters that can be stored in the item. If *integer-1* is omitted, it is assumed to have a value of one. The maximum value for *integer-1* is 65535.

If the VARYING phrase is specified, the data item may vary in length from zero characters to the number of characters specified by *integer-1*. The format of an elementary item in storage has a two-byte binary current-length field (equivalent to a 2-byte unsigned COMPUTATIONAL field) followed by the alphanumeric data field. The format is equivalent to the following data redefinition:

```
01  TOP-LEVEL.
    02  VARYING-ITEM CHARACTER VARYING (20).
    02  VARYING-RED REDEFINES VARYING-ITEM.
        03  RED-LENGTH PIC 9(4) COMPUTATIONAL.
        03  RED-ITEM PIC X(20).
```

When the data item is referenced at execution, the value of the length field is implicitly referenced to determine the effective length of the data field. Only the data positions encompassed by the current length are referenced.

(4) The USAGE IS DATE clause specifies a data item which can store an SQL date value. The value is stored as a sequence of 8 ASCII decimal digits (yyyymmdd), with the leftmost four-digits specifying the year field, the next two-digits specifying the month field, and the final two digits specifying the day of the month field. The values or the various fields must meet the rules for valid month and day values in the Gregorian calendar. The size of a DATE data item is 8 bytes.

(5) The USAGE IS INTEGER clause specifies a data item which can store an SQL integer value. It has the same storage format and runtime behavior as a signed 4-byte COMPUTATIONAL-5 data item.

(6) The USAGE IS INDICATOR clause specifies a data item which can store the value that indicates whether an item has no value (is NULL), has a valid value (is VALID), or has a truncated value (is OVERFLOW). The value can be set by using the SET statement or by specifying the data-item in the INDICATOR clause in the parameter list of an SQL statement.

(7) The USAGE IS INTERVAL clause specifies a data item which can store an SQL interval value. The value is stored as a sign, containing an ASCII '+' or '-' character, followed by a sequence of ASCII decimal digits

expressing the interval value in units of the rightmost field. Thus "10:08" MINUTE TO SECOND is stored as +0608. The number of digits in the leftmost field can be set by specifying *integer-2*. The maximum and default values for *integer-2* depend on the type of the leftmost field specifier and are specified in the following table. The size is 1 byte plus the sum of the sizes of the individual fields in the range, except for the DAY TO SECOND interval, which is one less (just the sum of the sizes of the individual fields.) This yields a minimum size of 2 bytes (sign plus a single field of precision 1) and a maximum size of 19 bytes.

Field	Maximum Precision as Leftmost Field	Default Precision
YEAR	4	4
MONTH	6	2
DAY	7	2
HOUR	8	2
MINUTE	10	2
SECOND	12	2

TABLE 11. INTERVAL Field Maximum Precision (***ISQL***)

(8) The USAGE IS SMALLINT clause specifies a data item which can store an SQL small integer value. It has the same storage format and runtime behavior as a signed 2-byte COMPUTATIONAL-5 data item.

(9) The USAGE IS TIME clause specifies a data item which can store an SQL time value. The value is stored as a sequence of 6 (hhmmss) plus *integer-3* ASCII decimal digits, with the leftmost two-digits specifying the hours field, the next two-digits specifying the minutes field, the next two-digits specifying the seconds field, and the final *integer-3* digits specifying the fractional seconds field. If *integer-3* is omitted, it is assumed to be zero. The value of *integer-3* must be less than or equal to 6. The values for the various fields must meet the rules for time keeping using a 24 hour clock, i.e., 00-23 for hours and 00-59 for minutes and seconds. Thus size of a TIME item is 6 plus *integer-3* bytes, with a maximum of 12 bytes.

(10) The USAGE IS TIMESTAMP clause specifies a data item which can store an SQL timestamp value. The value is stored as a DATE data item directly followed by a TIME data item with *integer-3* fraction digits. The size of a TIMESTAMP data item is 14 plus *integer-3* bytes.

(11) When a MOVE statement or an input-output statement that references a group item that contains an indicator data item is executed, no conversion of the data item takes place.

(12) The ON SIZE condition is processed as follows for the various usages:

a. For an INTEGER or SMALLINT item, the check is based on storage size rather than picture.

b. For a NUMERIC item, the check is based on the declared precision.

(13) Uninitialized DATE and TIME items can cause exceptions if used before a valid value is stored.

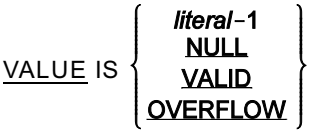
E.19. VALUE Clause

E.19.1 Function

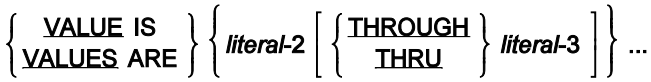
The VALUE clause defines the initial value of Working-Storage data items and the values associated with condition-names.

E.19.2 General Format

Format 1:



Format 2:



E.19.3 Syntax Rules

(1) The words THROUGH and THRU are equivalent.

(2) A signed numeric literal must have associated with it a signed numeric PICTURE *character-string* or a usage that represents a signed numeric item.

(3) All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause, and must not have a value which would require truncation of nonzero digits. Items whose USAGE enables size checking by storage must have a value which will fit in the storage allocated.

(4) Nonnumeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.

(5) The word NULL may only be specified for an item with usage POINTER or (***ISQL***) INDICATOR.

(6) (***ISQL***) The words VALID and OVERFLOW may only be specified for an item of usage INDICATOR.

(7) (***ISQL***) If the class of the item is date-time or interval, the literals in the VALUE clause must be of the same category and must not have a value which would require the truncation of nonzero digits.

(8) *Literal-1* may not be specified for an item with usage POINTER or (***ISQL***) usage INDICATOR.

E.19.4 General Rules

(1) The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item.

(2) If the category of the item is numeric, all literals in the VALUE clause must be numeric. If the literal defines the value of a working storage item, the literal is aligned in the data item according to the standard alignment rules.

(3) If the category of the item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric. Editing characters in the PICTURE clause are included in determining the size of the data item but have no effect on initialization of the data item. Therefore, the VALUE for an edited item must be specified in an edited form.

(4) Initialization is not affected by any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.

(5) (***ISQL***) If the category of the item is date, time, or timestamp the literals in the VALUE clause must be of the same category. If the literal defines the value of a working-storage item, the literal may also be a simple nonnumeric literal whose content matches the content of a literal of the same category as the item.

(6) (***ISQL***) If the category of the item is year-to-month or day-to-time, the literals in the VALUE clause must be of the same category and have the same range of field specifiers. If the literal defines the value of a working-storage item, the literal may also be a simple nonnumeric literal whose content matches the content of a literal of the same category and with the same range of field specifiers as the item.

E.19.5 Condition-Name Rules

(1) In a condition-name entry, the VALUE clause is required. The VALUE clause and the *condition-name* itself are the only two clauses permitted in the entry. The characteristics of a *condition-name* are implicitly those of its conditional variable.

(2) Format 2 can be used only in conjunction with condition-names. Whenever the THRU phrase is used, *literal-2* must be less than *literal-3*.

(3) A condition-name entry may not be used if the conditional variable is defined with usage POINTER, usage INDEX or (***ISQL***) usage INDICATOR.

E.19.6 Data Description Entries Other Than Condition-Names

(1) Rules governing the use of the VALUE clause differ with the respective sections of the Data Division:

a. In the File Section, the VALUE clause may be used only in condition-name entries; therefore, the initial value of the data item in the File Section is undefined.

b. In the Linkage Section, the VALUE clause may only be used in condition-name entries.

c. In the Working-Storage Section, the VALUE clause must be used in condition-name entries. VALUE clauses in the Working-Storage Section of a program take effect only when the program is placed into its initial state. If the VALUE clause is used in the description of the data item, the data item is initialized to the defined value. If the VALUE clause is not associated with a data item, the initial value of that data item is undefined.

d. In the Screen Section a figurative constant cannot be used.

(2) The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

(3) If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the individual elementary or group items contained within this group. The VALUE clause cannot be stated at the subordinate levels within this group.

(4) The VALUE clause must not be specified for a group item containing items subordinate to it with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

(5) A Format 1 VALUE clause specified in a data description entry that contains an OCCURS clause or in a entry that is subordinate to an OCCURS clause causes every occurrence of the associated data item to be assigned the specified value.

(6) If a VALUE clause is specified in a data description entry of a data item which is associated with a variable occurrence data item, the initialization of the data item behaves as if the value of the data item referenced by the DEPENDING ON phrase in the OCCURS clause specified for that variable occurrence data item is set to the maximum number of occurrences as specified by the OCCURS clause. A data item is associated with a variable occurrence data item in any of the following cases:

a. It is a group data item containing a variable occurrence data item.

b. It is a variable occurrence data item.

c. It is subordinate to a variable occurrence data item.

If a VALUE clause is associated with the data item referenced by a DEPENDING ON phrase, that value is considered to be placed in the data item after the variable occurrence data item has been initialized.

(7) (***ISQL***) If the VALUE clause is specified in a data description entry that contains the VARYING phrase, the current length of the data item is also initialized to the length of the literal item specified in the VALUE clause. If the literal item is a figurative constant, the length is the length of a single occurrence of the constant.

F. VIRTUAL-STORAGE SECTION (*VXCOBOL*)

The Virtual-Storage Section is located in the Data Division of a source program. The Virtual-Storage Section is treated as an extension of the Working-Storage Section.

The general format of the Virtual-Storage Section is the same as that shown for the Working-Storage Section.

All rules that apply to the Working-Storage Section apply equally to the Virtual-Storage Section.

G. LINKAGE SECTION

The Linkage Section is located in the Data Division of a source program. The Linkage Section appears in the called program and describes data items that are to be referred to by the calling program and the called program.

The Linkage Section in a program is meaningful if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase or if the program was passed data from another program with a CALL PROGRAM statement that contained a USING phrase or the program was started with data passed in when the runtime system was initially started.

The Linkage Section is used for describing data that is available through the calling program but is to be referred to in both the calling and the called program. The mechanism by which a correspondence is established between the data items described in the Linkage Section of a called program and data items described in the calling program is described elsewhere in these specifications. In the case of index-names, no such correspondence is established and index-names in the called and calling programs always refer to separate indices.

The structure of the Linkage Section is the same as that previously described for the Working-Storage Section, beginning with a section header, followed by noncontiguous data items and/or record description entries.

The general format of the Linkage Section is shown below.

```
LINKAGE SECTION.
[ 77-level-description-entry ]
[ record-description-entry   ] ...
```

If a data item in the Linkage Section is accessed in a program which is not a called program, the effect is undefined.

G.1. Noncontiguous Linkage Storage

Items in the Linkage Section that bear no hierarchical relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data description entry which begins with the special level-number 77.

The following data clauses are required in each data description entry:

1. level-number 77
2. data-name
3. the PICTURE clause or a USAGE clause that precludes the use of a PICTURE clause.

Other data description clauses are optional and can be used to complete the description of the item if necessary.

G.2. Linkage Records

Data elements in the Linkage Section which bear a definite hierarchical relationship to one another must be grouped into records according to the rules for formation of record descriptions. Data elements in the Linkage Section which bear no hierarchical relationship to any other data item may be described as records which are single elementary items.

G.3. Initial Values

The VALUE clause must not be specified in the Linkage Section except in condition-name entries (level-number 88).

H. SCREEN SECTION

The Screen Section is located in the Data Division of a source program. The Screen Section defines the attributes of screens to be used in interactive screen I/O. Screen section entries are referenced in the procedure Division with the ACCEPT and DISPLAY verbs. The Screen Section is an extension to ANSI COBOL.

The Screen Section is composed of the section header, followed by screen description entries.

The general format of the Screen Section is shown below.

SCREEN SECTION.
[*screen-description-entry*]...

H.1. Screen Description

A screen description consists of a set of screen description entries which describe the characteristics of a particular screen. Each screen description entry consists of a level-number followed by the screen-name, if specified, followed by a series of independent clauses as required. A screen description may have a hierarchical structure and therefore the clauses used within an entry may vary considerably, depending upon whether or not it is followed by subordinate entries.

In its simplest form, the screen description consists of a single, named 01 level item that can be a screen-literal or screen-data format entry. In its more complex form, the screen description consists of a named 01 level item that is a screen-group format item. The screen description consists of the 01 level item and all items subordinate to it. It can have the same type of hierarchical structure as a record description.

The screen description entry and the allowable elements are explained in the next section.

H.2. Screen Description Entry

H.2.1 Function

A screen description entry specifies the characteristics of a particular item in a screen.

Screen data description entries can be screen-literal, screen-data, or screen-group format items. Screen-literal format is used to display constant information, such as prompts. Screen-data format is used to perform input/output operations and transfer data between the screen-data and data items in the File, Working-Storage, and Linkage sections. Screen-group format is used to organize multiple screen-literal and screen-data items into logical groups for input/output operations as well as to specify attributes that apply to several screen-data items.

Screen description entries consist of a level number, an optional screen-name, and optional clauses that specify the position of a field, along with various attributes.

H.2.2 General Format

Screen-Literal Format: (***ANSI 74*** and ***ANSI 85***)

level-number [{ *screen-name* / FILLER }]

[{ BACKGROUND-COLOR / BACKGROUND } IS { *integer* / *color-name* / *identifier* }]

[{ FOREGROUND-COLOR / FOREGROUND } IS { *integer* / *color-name* / *identifier* }]

[{ COLUMN / COL } [NUMBER IS [PLUS / + / MINUS / -] { *integer* / *identifier* }]]

[LINE [NUMBER IS [PLUS / + / MINUS / -] { *integer* / *identifier* }]]

[BLANK { LINE / REMAINDER / SCREEN }]

[ERASE [{ LINE / SCREEN / EOL / EOS / END OF LINE / END OF SCREEN }]]

[{ BELL / BEEP }]

[BLINK]

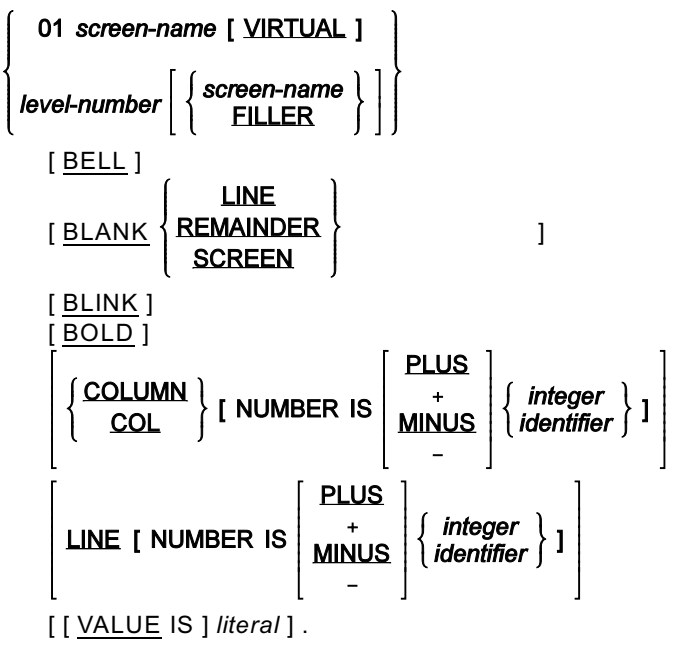
[{ BOLD / BRIGHT / [NO] HIGHLIGHT / DIM / LOWLIGHT }]

[{ REVERSE-VIDEO / REVERSE / REVERSED }]

[{ UNDERLINED / UNDERLINE }]

[[VALUE IS] *literal*] .

Screen-Literal Format: (*VXCOBOL*)



Screen-Data Format: (*ANSI 74* and *ANSI 85*)

```
level-number [ { screen-name } ]
             { FILLER      }

  [ { BACKGROUND-COLOR } IS { integer    } ]
    { BACKGROUND       }    { color-name }
                            { identifier }

  [ { FOREGROUND-COLOR } IS { integer    } ]
    { FOREGROUND       }    { color-name }
                            { identifier }

  [ { COLUMN } [ NUMBER IS [ PLUS  ] { integer    } ] ]
    { COL    }             [ +     ] { identifier } ]
                             MINUS
                             -

  [ LINE [ NUMBER IS [ PLUS  ] { integer    } ] ]
                     [ +     ] { identifier } ]
                      MINUS
                      -

  [ BLANK { LINE      } ]
          { REMAINDER }
          { SCREEN    }

  [ ERASE [ { LINE          } ] ]
          [ { SCREEN        } ]
          [ { EOL           } ]
          [ { EOS           } ]
          [ { END OF LINE   } ]
          [ { END OF SCREEN } ]

  [ BELL  ]
  [ BEEP  ]

  [ BLINK ]

  [ { BOLD              } ]
  [ { BRIGHT            } ]
  [ { [NO] HIGHLIGHT    } ]
  [ { DIM               } ]
  [ { LOWLIGHT          } ]

  [ { UNDERLINED } ]
  [ { UNDERLINE  } ]

  [ OCCURS integer TIMES ]

  [ AUTO ]

  [ BLANK WHEN ZERO ]

  [ CONVERTING { UP   } ]
               { DOWN }

  [ FULL ]

  [ { JUSTIFIED } RIGHT ]
  [ { JUST      } ]

  { PICTURE } IS character-string
  { PIC    }

  [ REQUIRED ]
```

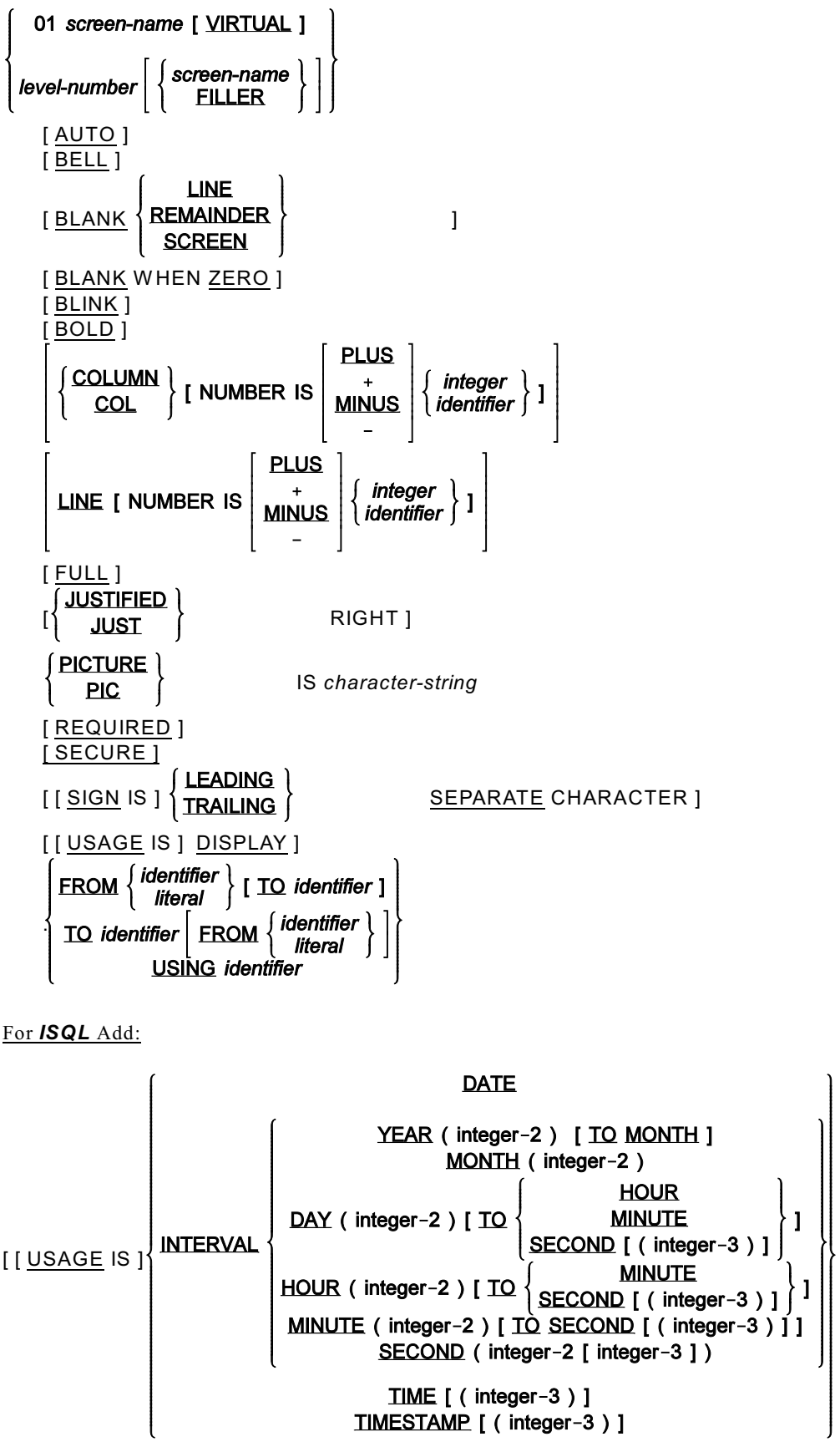
```
[ SECURE [ { WITH ECHO } ]                                  ]
           [ { NO ECHO  } ]

[ [ SIGN IS ] { LEADING }                  SEPARATE CHARACTER ]
              { TRAILING }

d     [ [ USAGE IS ] DISPLAY ]
      { FROM { arith-exp } [ TO identifier ]
      {      { identifier }
      {      { literal    }
      { TO identifier [ FROM { arith-exp } ]
      {                      { identifier }
      {                      { literal    }
      {      USING identifier                }
```

For ***ISQL*** Add:

```
                                  DATE
                                                                                  }
                                                                                  }
                                  YEAR ( integer-2 )  [ TO MONTH ]                }
                                  MONTH ( integer-2 )                             }
                                                                                  }
                                                                                  }
                                  DAY ( integer-2 ) [ TO { HOUR   } ]             }
                                                         { MINUTE   }             }
                                                         { SECOND [ ( integer-3 ) ] }
                                                                                  }
                                  HOUR ( integer-2 ) [ TO { MINUTE   } ]          }
                                                         { SECOND [ ( integer-3 ) ] }
                                  MINUTE ( integer-2 ) [ TO SECOND [ ( integer-3 ) ] ]
                                  SECOND ( integer-2 [ integer-3 ] )              }
                                                                                  }
                                  TIME [ ( integer-3 ) ]                          }
                                  TIMESTAMP [ ( integer-3 ) ]                     }
```

```
[ [ USAGE IS ] { INTERVAL { YEAR ( integer-2 )  [ TO MONTH ]                    } ]
                          { MONTH ( integer-2 )                                 }
                          { DAY ( integer-2 ) [ TO { HOUR   } ]                 }
                          {                        { MINUTE   }                 }
                          {                        { SECOND [ ( integer-3 ) ] }
                          { HOUR ( integer-2 ) [ TO { MINUTE   } ]              }
                          {                        { SECOND [ ( integer-3 ) ] }
                          { MINUTE ( integer-2 ) [ TO SECOND [ ( integer-3 ) ] ]
                          { SECOND ( integer-2 [ integer-3 ] )                  }
                          }
                          { TIME [ ( integer-3 ) ]                              }
                          { TIMESTAMP [ ( integer-3 ) ]                         }
```

Screen-Data Format: (***VXCOBOL***)



For ***ISQL*** Add:

Screen-Group Format: (ANSI 74 and ANSI 85)

```
level-number [ { screen-name } ]
               { FILLER      }

  [ { BACKGROUND-COLOR } IS { integer    } ]
    { BACKGROUND       }    { color-name }
                            { identifier }

  [ { FOREGROUND-COLOR } IS { integer    } ]
    { FOREGROUND       }    { color-name }
                            { identifier }

  [ { COLUMN } [ NUMBER IS [ PLUS  ] { integer    } ] ]
    { COL    }             [ +    ]  { identifier } ]
                             MINUS
                             -

  [ LINE [ NUMBER IS [ PLUS  ] { integer    } ] ]
                     [ +    ]  { identifier } ]
                      MINUS
                      -

  [ BLANK SCREEN ]

  [ { BELL } ]
    { BEEP }

  [ { BOLD      }
    { BRIGHT    }
  [ { [NO] HIGHLIGHT } ]
    { DIM       }
    { LOWLIGHT  }

  [ { REVERSE      } ]
  [ { REVERSED     } ]
    { REVERSE-VIDEO }

  [ { UNDERLINED } ]
    { UNDERLINE  }

  [ OCCURS integer TIMES ]
  [ AUTO ]
  [ FULL ]
  [ REQUIRED ]
  [ SECURE [ { WITH ECHO } ] ]
           [ { NO ECHO   } ]

d [ [ USAGE IS ] DISPLAY ]
  [ [ SIGN IS ] { LEADING  } SEPARATE CHARACTER ] .
                { TRAILING }

  { screen-data-item    }...
  { screen-literal-item }
```

Screen-Group Format: (*VXCOBOL*)

```
{ 01 screen-name [ VIRTUAL ] }
{ level-number [ { screen-name } ] }
{                { FILLER      } }

  [ AUTO ]
  [ { COLUMN } [ NUMBER IS [ PLUS ] { integer } ] ]
  [ { COL    }             [ +    ] { identifier } ]
  [                        [ MINUS ]                ]
  [                        [ -     ]                ]
  [ LINE [ NUMBER IS [ PLUS ] { integer } ] ]
  [                 [ +    ] { identifier } ]
  [                 [ MINUS ]                ]
  [                 [ -     ]                ]
  [ BLANK SCREEN ]
  [ BELL ]
  [ BOLD ]
  [ FULL ]
  [ REQUIRED ]
  [ SECURE ]
  { screen-data-item } ...
  { screen-literal-item }
```

H.2.3 Syntax Rules

(1) *Level-number* may be any number from 01 through 49.

(2) *Screen-name* is required for level 01.

(3) In all formats, if *screen-name* is present it must immediately follow the level number.

(4) The literal in the VALUE clause must be a nonnumeric literal and it cannot be a figurative constant.

(5) Unnamed items cannot be referenced individually, but only indirectly by referencing a containing named screen-group item.

(6) A screen-literal format item cannot specify a PICTURE clause.

(7) A screen-data format item cannot specify a VALUE clause.

(8) A screen-data format item must include a PICTURE clause as well as one of the following combinations of the TO, FROM, and USING clauses:

a. a FROM clause,
b. a TO clause,
c. a FROM clause and a TO clause, or
d. a USING clause.

(9) The JUSTIFIED and BLANK WHEN ZERO clauses may only be specified for a screen-data format item and are subject to the same PICTURE compatibility restrictions as apply to a data item in Working Storage.

(10) If more than one clause is specified for an entry, the clauses may occur in any order. Since at execution time a specific order is followed, it is useful to follow this same order in the source program.

The order of execution for a DISPLAY statement is as follows:

For ***ANSI 74*** and ***ANSI 85***:

BACKGROUND-COLOR & FOREGROUND-COLOR
BLANK SCREEN
COLUMN and LINE positioning
BLANK LINE/ERASE EOL, ERASE EOS, ERASE LINE
BELL
display literal or data with appropriate attributes

For ***VXCOBOL***:

BLANK SCREEN
COLUMN and LINE positioning
BLANK LINE
BELL
display literal or data with appropriate attributes

The order of execution for an ACCEPT statement is as follows:

For ***ANSI 74*** and ***ANSI 85***:

BACKGROUND-COLOR & FOREGROUND-COLOR
COLUMN and LINE positioning
accept data with appropriate attributes

For ***VXCOBOL***:

COLUMN and LINE positioning
accept data with appropriate attributes

(11) USAGE IS DISPLAY is for documentation purposes only as USAGE IS DISPLAY is the default.

Additional Syntax Rule for *VXCOBOL*:

(12) The VIRTUAL clause is used for documentation only, but may only be specified for an 01 level entry.

H.2.4 General Rules

(1) The PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses have the same meaning for screen-data items as for data items in Working Storage. The other clauses are described more fully in the sections that follow.

(2) A screen-data item with a FROM clause and no TO clause is described as an output field.

(3) A screen-data item with a TO clause and no FROM clause is described as an input field.

(4) A screen-data item with both a FROM clause and a TO clause is described as an input-output field.

(5) A screen-data item with a USING clause is described as an update field.

(6) The relationship of the level numbers in a screen description are used to differentiate between the screen-group format items and the elementary format items, which are screen-literal and screen-data.

(7) For the two elementary format items, the presence of a PICTURE clause is used to differentiate between a screen-data format item and a screen-literal format item.

Notes:

1. The default appearance for literal and output fields is DIM.
2. The default appearance for input, input-output, and update fields is BOLD.

H.3. AUTO, FULL, REQUIRED Clauses

H.3.1 Function

The clauses AUTO, FULL, and REQUIRED affect the behavior of data entry to input, input-output, and update fields during the execution of an ACCEPT statement.

H.3.2 General Format

AUTO
FULL
REQUIRED

H.3.3 Syntax Rules

(1) These clauses can only be used with input, input-output, or update screen-data items or with screen-group items.

H.3.4 General Rules

(1) If one of these clauses is written at a screen-group level, it applies to each elementary input, input-output, and update item in the screen-group.

(2) These clauses have no effect during the execution of a DISPLAY statement.

(3) The AUTO clause causes data entry for the field to automatically terminate when data is entered into the last character position in the field. If this field is the last field in the screen the ACCEPT is terminated as if a normal terminator (any key with an ESCAPE KEY value of 00) had been entered.

(4) The FULL clause requires that a character or space must be entered in every position of a field, if any character is entered. USING fields are initially always full.

(5) The REQUIRED clause requires that there must be at least one non-blank character in the data entry field before data entry for the field can be terminated.

> **Note:** A non-blank update field will always satisfy this requirement.

H.4. BACKGROUND-COLOR, FOREGROUND-COLOR Clauses (***ANSI 74*** and ***ANSI 85***)

H.4.1 Function

The BACKGROUND-COLOR and FOREGROUND-COLOR clauses set the background and foreground color for a screen item.

H.4.2 General Format

$$\left\{\begin{array}{c}\underline{\text{BACKGROUND-COLOR}}\\ \underline{\text{BACKGROUND}}\end{array}\right\} \text{IS} \left\{\begin{array}{c}\textit{integer}\\ \textit{color-name}\\ \textit{identifier}\end{array}\right\}$$

$$\left\{\begin{array}{c}\underline{\text{FOREGROUND-COLOR}}\\ \underline{\text{FOREGROUND}}\end{array}\right\} \text{IS} \left\{\begin{array}{c}\textit{integer}\\ \textit{color-name}\\ \textit{identifier}\end{array}\right\}$$

H.4.3 Syntax Rules

(1) BACKGROUND-COLOR and BACKGROUND are synonyms.

(2) FOREGROUND-COLOR and FOREGROUND are synonyms.

H.4.4 General Rules

(1) The BACKGROUND-COLOR clause determines the background color for a screen item.

(2) The FOREGROUND-COLOR clause determines the foreground color for a screen item.

(3) These clauses are effective only with color screens.

(4) The color is specified by entering an integer from 0 to 7, the appropriate color-name, or and integer whose value ranges from 0 to 7. The color-names with their integer values are shown in the chart below.

Color	BLACK	BLUE	GREEN	CYAN	RED	MAGENTA	BROWN	WHITE
Integer	0	1	2	3	4	5	6	7

TABLE 12. BACKGROUND-COLOR and FOREGROUND-COLOR

(5) If the value of *identifier* falls outside of the range 0 through 7, then the associated BACKGROUND-COLOR or FOREGROUND-COLOR clause is ignored.

(6) When used at the screen-group level, these clauses apply to all subordinate screen items. If no colors are specified, the terminal uses its default background and foreground colors.

H.5. BELL Clause

H.5.1 Function

The BELL clause sounds the tone on the user's display device.

H.5.2 General Format

```
{ BELL }
{ BEEP }
```

H.5.3 Syntax Rules

(1) The words BELL and BEEP are synonyms.

H.5.4 General Rules

(1) The BELL clause is effective only during the execution of a DISPLAY statement.

(2) If BELL is specified on a screen-group format item, the tone is only sounded when the screen-group item is processed, not when each item subordinate to the screen-group item is processed.

H.6. BLANK Clause

H.6.1 Function

The BLANK clause is used to erase part or all of the user's display device during the execution of a DISPLAY statement.

H.6.2 General Format

```
BLANK { LINE      }
      { REMAINDER }
      { SCREEN    }
```

H.6.3 General Rules

(1) The BLANK LINE, BLANK SCREEN, and BLANK REMAINDER clauses are effective only during the execution of a DISPLAY statement.

(2) BLANK SCREEN erases the entire screen and positions the cursor to line 1 column 1.

(3) BLANK SCREEN is processed before any LINE and COLUMN positioning clauses because it has an implicit positioning of the cursor.

(4) BLANK LINE erases the current line from the cursor position to the end of the line without changing the cursor position.

(5) BLANK REMAINDER erases the screen starting at the cursor position to the end of the screen. The cursor is not affected.

(6) BLANK LINE and BLANK REMAINDER are processed after the LINE and COLUMN positioning clauses and before any screen-literal or screen-data items so that they can be used to clear data previously displayed on the screen before displaying new data at the same position.

NOTE: If the compiler's ISO screen behavior option (-G e) is specified, the BLANK LINE clause will erase the entire line starting in column 1 rather than starting at the specified or implied column position.

H.7. BLINK, BOLD/BRIGHT/HIGHLIGHT/DIM/LOWLIGHT, REVERSE/REVERSED/REVERSED-VIDEO, UNDERLINE/UNDERLINED Clauses

H.7.1 Function

These clauses are used to control the appearance of data that is displayed on the user's display device.

H.7.2 General Format (***ANSI 74*** and ***ANSI 85***)

```
BLINK
{ BOLD                  }
{ BRIGHT                }
{ [NO] HIGHLIGHT        }
{ DIM                   }
{ LOWLIGHT              }
{ REVERSE-VIDEO         }
{ REVERSE               }
{ REVERSED              }
{ UNDERLINED            }
{ UNDERLINE             }
```

H.7.3 General Format (***VXCOBOL***)

```
BLINK
BOLD
```

H.7.4 Syntax Rules

(1) These clauses can be specified for a screen-literal format item, a screen-data format item, or a screen-group format item.

H.7.5 General Rules

(1) These clauses apply to both ACCEPT and DISPLAY.

(2) The BLINK clause causes the field to blink.

(3) The BOLD, BRIGHT, or HIGHLIGHT clauses cause the field to be displayed at high intensity.

(4) The DIM, LOWLIGHT, or NO HIGHLIGHT clauses cause the field to be displayed at low intensity.

(5) The REVERSE, REVERSED, or REVERSE-VIDEO clauses cause the field to be displayed in reverse-video.

(6) The UNDERLINE or UNDERLINED clauses cause a field to be displayed underlined.

(7) The clauses can be combined to provide combined effects, such as bold and underlined. However, not all video display devices are capable of displaying all of the combinations.

Notes:

1. The default appearance for literal and output fields is DIM.
2. The default appearance for input, input-output, and update fields is BOLD.

H.8. CONVERTING Clause

H.8.1 Function

The CONVERTING clause is used to insure that accepted data is in a consistent case.

H.8.2 General Format

```
CONVERTING { UP   }
           { DOWN }
```

H.8.3 Syntax Rules

(1) The CONVERTING clause may only be specified in a screen description which includes either the TO or USING clauses.

H.8.4 General Rules

(1) The CONVERTING clause is effective only during the execution of an ACCEPT statement.

(2) If CONVERTING UP is specified character data entered during an ACCEPT is echoed to the screen and stored in uppercase. In particular characters between 'a' and 'z' inclusive are converted to the corresponding character between 'A' and 'Z'.

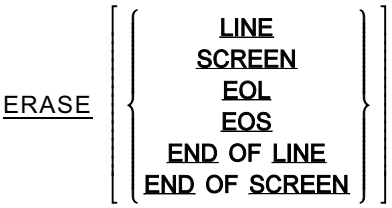
(3) If CONVERTING DOWN is specified character data entered during an ACCEPT is echoed to the screen and stored in lowercase. In particular characters between 'A' and 'Z' inclusive are converted to the corresponding character between 'a' and 'z.

H.9. ERASE Clause

H.9.1 Function

The ERASE clause is used to erase part or all of the user's display device during the execution of a DISPLAY statement.

H.9.2 General Format



H.9.3 Syntax Rules

(1) The word EOL is equivalent to the phrase END OF LINE.

(2) The word EOS is equivalent to the phrase END OF SCREEN.

H.9.4 General Rules

(1) The ERASE clause is effective only during the execution of a DISPLAY statement.

(2) ERASE SCREEN and ERASE with no additional modifiers erases the entire screen and positions the cursor to line 1 column 1.

(3) ERASE LINE erases the current line from column 1 to the end of the line without changing the cursor position.

(4) ERASE EOL and ERASE END OF LINE erase the screen starting at the cursor position to the end of the line. The cursor is not affected.

(5) ERASE EOS and ERASE END OF SCREEN erase the screen starting at the cursor position and continuing to the end of the screen. The cursor position is not changed.

(6) ERASE and ERASE SCREEN are processed before any LINE and COLUMN positioning clauses because they have an implicit positioning of the cursor. All other ERASE clauses are processed after the LINE and COLUMN positioning clauses and before any screen-literal or screen-data items so that they can be used to clear data previously displayed on the screen before displaying new data at the same position.

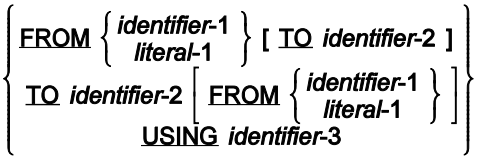
NOTE: If the compiler's ISO screen behavior option (-G e) is specified, the ERASE LINE clause will erase the line beginning at the cursor position and continuing to the end of the line. Similarly, the ERASE SCREEN clause will erase from the cursor position to the end of the screen.

H.10. FROM, TO, USING Clauses

H.10.1 Function

These clauses are used to determine the types of input-output operations (ACCEPT and DISPLAY) that can be performed on an item, as well as the associated data items or values.

H.10.2 General Format



H.10.3 General Rules

(1) A FROM clause with no TO clause defines the field as an output field.

(2) A TO clause with no FROM clause defines the field as an input field.

(3) A FROM clause and a TO clause together define a field as an input-output field.

(4) A USING clause defines the field as an update field.

(5) The default attribute for output fields is DIM, and for input, input-output, or update fields is BOLD. This may be overridden by using the appropriate attribute.

(6) When a DISPLAY statement is executed, an input field is displayed as underscore characters. The number of underscores displayed corresponds to the number of characters in the picture string.

(7) The item specified by *literal-1* or *identifier-1* (for an output or input-output field) or by *identifier-3* (for an update field) must be compatible with the screen-data according to the rules for the MOVE statement, where *literal-1*, *identifier-1*, or *identifier-3* is the sending item, and the screen-data is the receiving item.

(8) The item specified by *identifier-2* (for an input or input-output field) or *identifier-3* (for an update field) must be compatible with the screen-data according to the rules for the MOVE statement, where screen-data is the sending item and *identifier-2* or *identifier-3* is the receiving item, with the exception that the combination of a numeric edited sending item and numeric receiving item is allowed.

(9) If the subject of the TO, FROM or USING entry is subject to an OCCURS clause, *identifier-1*, *identifier-2*, or *identifier-3* shall be specified without the subscripting normally required.

(10) For more on how each clause works see ACCEPT screen, page 283; or DISPLAY screen, page 338.

H.11. LINE and COLUMN Clauses

H.11.1 Function

The LINE and COLUMN clauses specify the vertical and horizontal location of the cursor on the user's display device (and thus the location of the erase, input, or output operation being specified.)

H.11.2 General Format

```
LINE [ NUMBER IS [ PLUS  ] { integer-1    } ]
                 [ +     ] { identifier-1 }
                 [ MINUS ]
                 [ -     ]

{ COLUMN } [ NUMBER IS [ PLUS  ] { integer-2    } ]
{ COL    }             [ +     ] { identifier-2 }
                       [ MINUS ]
                       [ -     ]
```

H.11.3 Syntax Rules

(1) *Integer-1* and *integer-2* must be unsigned and non-zero.

(2) *Identifier-1* and *identifier-2* must represent an unsigned elementary numeric data item.

(3) The word COL is an abbreviation for the word COLUMN.

(4) PLUS and + are synonyms.

(5) MINUS and - are synonyms.

(6) Neither the PLUS nor MINUS phrase shall be specified for the first elementary item in a screen record.

(7) If generating for code revision 1, *identifier-1* and *identifier-2* may not be specified if either the PLUS or MINUS clauses are present.

H.11.4 General Rules

(1) The screen description entries in a screen description are processed beginning with the 01 level item and proceeding through all screen description entries subordinate to the 01 level item in the order in which they appear in the source program.

(2) As the screen description entries are processed, the compiler maintains a current cursor position for the screen description. The current cursor position determines the placement of fields on the user's display screen when an ACCEPT or DISPLAY statement is executed. This current cursor position is composed of two components, the current cursor line and the current cursor column, with the upper left corner of the display being line 1, column 1.

(3) The current cursor position cannot be greater than line 128 or column 128 at the beginning of a BLANK LINE operation or as the starting character position for a screen-literal or screen-data item.

(4) The current cursor position after a screen-literal or screen-data cannot be greater than line 128 or column 256.

(5) At execution time, if the current cursor position exceeds the size of the display device, the component of the position that exceeds the display size (line or column or both), is re-computed to be the remainder of the original

value divided by the display size, e.g., a cursor position of line 20 column 132 on a 24 line, 80 column display is re-computed as line 20, column 52.

(6) Each screen description is assumed to start (at the 01 level) with a current cursor position of line 1, column 1.

(7) If no LINE or COLUMN clause is specified, the current cursor position is not modified before the processing of any input, output, input-output, or update field that might be present, except by the BLANK SCREEN or ERASE SCREEN clause (which sets the current cursor position to line 1, column 1.)

(8) In the rules that follow, the components of the current cursor position are often treated independently.

(9) Line and/or column positions may be specified in one of three ways: Absolute positioning; Relative positioning; and Variable positioning.

(10) Absolute line positioning is defined by a LINE clause with *integer-1* and without either PLUS or MINUS. The value of *integer-1* becomes the new current cursor line value. It may not exceed the value 128 and should not exceed the usual number of lines in the display device. If a COLUMN phrase is also specified, it is handled independently. If one is not specified, it is assumed to be the same as specifying COLUMN 1.

(11) Absolute column positioning is defined by a COLUMN clause with *integer-2* and without either PLUS or MINUS. The value of *integer-2* becomes the current cursor column value. It may not exceed the value 128 and should not exceed the usual number of columns in the display device.

(12) Relative line positioning is defined by using the LINE clause with either the PLUS or MINUS phrase and *integer-1* or *identifier-1*. If PLUS is specified, the current cursor line is incremented by the value of *integer-1* or the contents of *identifier-1*. If MINUS is specified, the current cursor line is decremented by the value of *integer-1* or the contents of *identifier-1*. The resulting value must not exceed 128 or be less than 1. If a COLUMN phrase is also specified, it is handled independently. If one is not specified, it is assumed to be the same as specifying COLUMN 1.

(13) Relative column positioning is defined by using the COLUMN clause with either the PLUS or MINUS phrase and *integer-2* or *identifier-2*. If PLUS is specified, the current cursor column is incremented by the value of *integer-2* or the contents of *identifier-2*. If MINUS is specified, the current cursor column is decremented by the value of *integer-2* or the contents of *identifier-2*. The resulting value must not exceed 128 or be less than 1.

(14) Variable line positioning is defined by using the LINE clause with *identifier-1* and without either PLUS or MINUS. The actual line value is not known until execution time. If a COLUMN phrase is also specified, it is handled independently. If one is not specified, it is assumed to be the same as specifying COLUMN 1.

(15) Variable column positioning is defined by using the COLUMN clause with *identifier-2* and without either PLUS or MINUS. The actual column value is not known until execution time.

(16) If generating for code revision 1, relative line positioning cannot be specified for an entry that follows an entry with variable line positioning unless there is an intervening entry with absolute line positioning. If generating for code revision 2 or greater, the this restriction does not apply.

(17) If generating for code revision 1, relative column positioning cannot be specified for an entry that follows an entry with variable column positioning unless there is an intervening entry with absolute column positioning. The absolute column positioning may be derived from the COLUMN 1 clause that is implied in certain cases. If generating for code revision 2 or greater, this restriction does not apply.

(18) If the screen description entry also contains a VALUE clause or is an input, output, input-output, or update field, the value of the current cursor position is associated with the literal or field item. The current cursor column is also updated to be positioned at the first column after the literal or field item (i.e., it is incremented by the length of the literal or field item) unless the current cursor column is currently undefined because of variable column positioning. The updated current column position cannot exceed 256.

(19) The effects of LINE and COLUMN clauses in combination with each other is defined in the following table:

LINE Clause	COLUMN Clause	Field Position
No clause	No clause	No change
	COL	Same line, column plus 1
	COL n	Same line, column n
	COL PLUS n	Same line, column plus n
	COL id	Same line, column id
LINE	No clause	Line plus 1, column 1
	COL	Line plus 1, column plus 1
	COL n	Line plus 1, column n
	COL PLUS n	Line plus 1, column plus n
	COL id	Line plus 1, column id
LINE m	No clause	Line m, column 1
	COL	Line m, column plus 1
	COL n	Line m, column n
	COL PLUS n	Line m, column plus n
	COL id	Line m, column id
LINE PLUS m	No clause	Line plus m, column 1
	COL	Line plus m, column plus 1
	COL n	Line plus m, column n
	COL PLUS n	Line plus m, column plus n
	COL id	Line plus m, column id
LINE id	No clause	Line id, column 1
	COL	Line id, column plus 1
	COL n	Line id, column n
	COL PLUS n	Line id, column plus n
	COL id	Line id, column id

TABLE 13. LINE and COLUMN relationship

H.12. OCCURS Clause

H.12.1 Function

The OCCURS clause is similar to the OCCURS clause defined in the Working-Storage Section. It is eliminates the need for separate entries for repeated screen items and supplies information needed for the application of subscripts.

H.12.2 General Format

OCCURS *integer* TIMES

H.12.3 Syntax Rules

(1) The maximum number of dimensions for a table described in a screen description entry is two.

(2) If a screen description entry includes the OCCURS clause, then if it or any item subordinate to it has a description that includes the TO, FROM, or USING clause, that screen description entry shall be part of a table with the same number of dimensions and number of occurrences in each dimension as the identifier representing the receiving or sending operand. The identifier representing the receiving or sending operand shall not be subordinate to an OCCURS clause with the DEPENDING phrase.

(3) If a screen description entry that includes the OCCURS clause also contains the COLUMN clause, then the COLUMN clause shall include the PLUS or MINUS phrase, unless the screen description entry also includes a LINE clause with a PLUS or MINUS phrase.

(4) If a screen description entry that includes the OCCURS clause also contains the LINE clause, then the LINE clause shall include the PLUS or MINUS phrase, unless the screen description entry also includes a COLUMN clause with a PLUS or MINUS phrase.

H.12.4 General Rules

(1) During a DISPLAY screen or an ACCEPT screen statement that references a screen item whose description includes the OCCURS clause and whose description or whose subordinate's description includes a FROM, TO, or USING clause, the data values for corresponding table elements are moved from the data table element to the screen table element or from the screen table element to the data table element.

(2) If the description of a screen item includes the OCCURS clause, the positioning within the screen record of each occurrence of that screen item is as follows:

a. If the description of that screen item contains a COLUMN clause, each occurrence behaves as though it had the same column clause specified.

b. If that screen item is a group item with a subordinate screen item whose description contains a COLUMN clause with the PLUS or MINUS phrase and that group screen item is subordinate to a screen item whose description contains a LINE clause, each occurrence behaves as though it had the same subordinate entries with the same COLUMN clause specified.

c. If the description of that screen item contains a LINE clause with the PLUS or MINUS phrase, each occurrence behaves as though it had the same LINE clause specified.

d. If that screen item is a group item with a subordinate screen item whose description contains a LINE clause with the PLUS or MINUS phrase, each occurrence behaves as though it had the same subordinate entries with the same LINE clause specified.

H.13. PICTURE Clause

H.13.1 Function

The PICTURE clause is similar to the PICTURE clause defined in the Working-Storage Section. It is used to determine the size of a screen-data, the format of the data when it is presented to the user through the execution of a DISPLAY statement, and the data validation rules to apply to the data when it is entered by the user in response to the execution of an ACCEPT statement.

H.13.2 General Format

```
{ PICTURE }
{ PIC     }    IS character-string
```

H.13.3 Syntax Rules

(1) The picture characters P, V, CR, and DB can be used only with output (FROM) fields.

H.13.4 General Rules

(1) The rules for compatibility between the screen-data PICTURE and the literal or data items specified in the TO, FROM, or USING clauses are specified under the section for TO, FROM, and USING.

(2) Unless the SIGN IS clause is also specified, the S PICTURE character is ignored by the compiler to be consistent with older versions of Interactive COBOL.

H.14. SECURE Clause

H.14.1 Function

This clause affects the behavior of data entry to input fields while in an ACCEPT and how the entry is displayed during an ACCEPT.

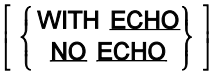
ANSI 74 and ***ANSI 85***:

The SECURE or SECURE ECHO clause causes asterisks to be echoed on the display during data entry or data display. The SECURE NO ECHO clause prevents characters from echoing on the display during data entry or data display.

VXCOBOL:

The SECURE clause prevents characters from echoing on the display during data entry or data display.

H.14.2 General Format (***ANSI 74*** and ***ANSI 85***)



H.14.3 General Format (***VXCOBOL***)

SECURE

H.14.4 Syntax Rules

(1) This clause can only be used with input, input-output, or update screen-data items or with screen-group format items.

H.14.5 General Rules

(1) The SECURE clause is effective only during the execution of an ACCEPT statement.

(2) If the SECURE clause is specified for a screen-group item, the clause applies to each elementary input, input-output, and update item subordinate to the screen-group item.

ANSI 74 and ***ANSI 85***:

(3) During the execution of an ACCEPT statement for a screen item that contains SECURE or SECURE ECHO, any characters entered by the user will be echoed as asterisks.

(4) During the execution of an ACCEPT statement for a screen item that contains SECURE NO ECHO, any characters entered by the user will not be echoed, and the cursor will not move as the characters are entered.

VXCOBOL:

(5) During the execution of an ACCEPT statement, any characters entered by the user will not be echoed. Additionally, the cursor will not move as the characters are entered.

H.15. SIGN Clause

H.15.1 Function

The SIGN clause specifies the position and the mode of representation of the operational sign when it is necessary to describe these properties explicitly.

H.15.2 General Format

```
[ SIGN IS ] { LEADING  }          SEPARATE CHARACTER
            { TRAILING }
```

H.15.3 Syntax Rules

(1) The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character `S'.

(2) The numeric data description entries to which the SIGN clause applies must be described, implicitly or explicitly, as USAGE IS DISPLAY.

H.15.4 General Rules

(1) The optional SIGN clause, if present, specifies the position and the mode of representation of the operational sign for the numeric data description entry to which it applies, or for each numeric data description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character `S'; the `S' indicates the presence of, but neither the representation nor, necessarily, the position of the operational sign.

(2) If a SIGN clause is specified in a group item subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate group item takes precedence for that subordinate group item.

(3) If a SIGN clause is specified in an elementary numeric data description entry subordinate to a group item for which a SIGN clause is specified, the SIGN clause specified in the subordinate elementary numeric data description entry takes precedence for that elementary numeric data item.

(4) a. The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.

b. The letter `S' in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).

c. The operational signs for positive and negative are the standard data format characters `+' and `-' respectively.

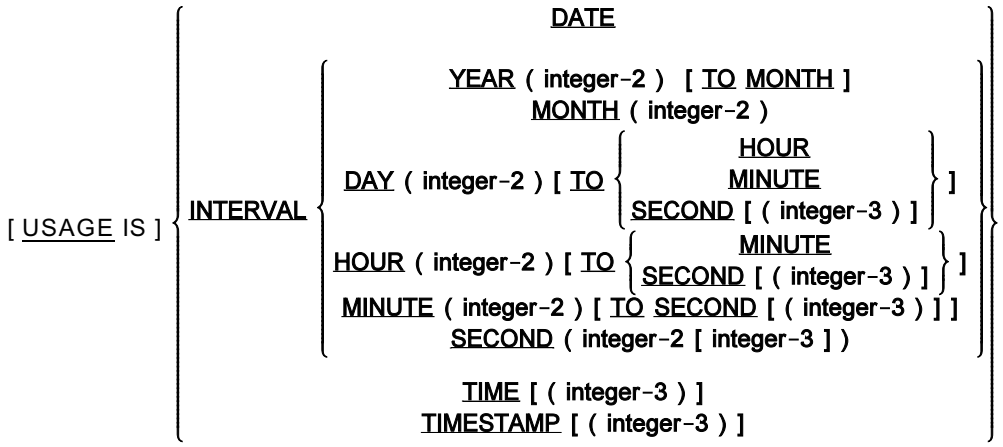
(5) Every numeric data description entry whose PICTURE contains the character `S' is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

H.16. USAGE Clause (***ISQL***)

H.16.1 Function

The USAGE clause specifies the special formatting of the data for the corresponding usage in the TO, FROM, or USING data items or literals.

H.16.2 General Format



H.16.3 Syntax Rules

(1) The USAGE clause specifying DATE, INTERVAL, TIME, or TIMESTAMP is available only when the ***ISQL*** feature-set is enabled and appear as USAGE options that are in addition to certain dialect-specific options.

(2) A USAGE clause specifying DATE, INTERVAL, TIME, or TIMESTAMP must not be specified at the group level.

(3) The BLANK WHEN ZERO, JUSTIFIED, PICTURE, and SIGN clauses must not be specified for screen items whose usage is DATE, INTERVAL, TIME, or TIMESTAMP.

(4) The value of *integer-2* must be greater than zero and less than the values specified in the general rules below.

(5) The value of *integer-3* must be greater than zero and less than or equal to six.

E.17.5 General Rules

(1) If the USAGE clause is not specified, the usage is implicitly DISPLAY.

(2) The USAGE IS DATE clause specifies a screen item that can accept and display an SQL date value. Upon output, the date value will be formatted with intervening hyphens in the same manner as a date literal. Upon input, the field will appear as three separate fields separated by intervening hyphens. The system will automatically skip over the hyphens. The entered value will be tested to be a valid date and an appropriate message will be displayed to the user if it is not. A screen field of usage DATE occupies 10 characters. (yyyy-mm-dd)

(3) The USAGE IS INTERVAL clause specifies a screen item that can accept or display an SQL interval value. Upon output, the interval value will be formatted in the same manner as the corresponding interval literal. The number of digits in the leftmost field can be set by specifying *integer-3*. The maximum and default values for

integer-3 depend on the type of the leftmost field specifier and are specified in the following table. The size of the screen field is 1 byte plus the sum of the sizes of the individual fields in the range, plus the number of fields minus one for the intervening formatting characters. Upon input, the field will appear as separate fields (one for each of the interval fields) separated by the appropriate punctuation. The system will automatically skip over the punctuation. The entered value will be tested to be a valid interval and an appropriate message will be displayed to the user if it is not. The field has a minimum size of 2 characters (sign plus a single field of precision 1) and a maximum size of 24 characters.

For example, DAY (7) TO SECOND (6) may have a value that displays as:

+1234567 12:34:56.123456

Field	Maximum Precision as Leftmost Field	Default Precision
YEAR	4	4
MONTH	6	2
DAY	7	2
HOUR	8	2
MINUTE	10	2
SECOND	12	2

TABLE 14. INTERVAL Field Maximum Precision (***ISQL***)

(4) The USAGE IS TIME clause specifies a screen item that can accept and display an SQL time value. If *integer-3* is omitted, it is assumed to be zero. The value of *integer-3* must be less than or equal to 6 and specifies the number of fractional seconds field. Upon output, the data is formatted in the same manner as a time literal with the intervening colons and an optional decimal point. Upon input, the field will appear as three (or four) separate fields separated by the colons and an optional decimal point. The system will automatically skip over the colons (and decimal point) as data is entered. The entered value will be tested to be a valid time and an appropriate message will be displayed to the user if it is not. A screen field of usage TIME occupies from 8 to 15 characters on the screen. I.E., from hh:mm:ss to hh:mm:ss.nnnnnn.

(5) The USAGE IS TIMESTAMP clause specifies a screen item that can accept and display an SQL timestamp value. The screen item is a composite of a date screen field and a time screen field with an intervening space. A TIMESTAMP screen item occupies from 19 to 26 characters, depending on the number of fractional second digits.

H.17 VALUE Clause

H.17.1 Function

The VALUE clause specifies literal information to be displayed.

H.17.2 General Format

[VALUE IS] *literal-1*

H.17.3 Syntax Rules

(1) *Literal-1* must be a nonnumeric-literal.

(2) *Literal-1* must not be a figurative constant.

(3) The words VALUE IS are not required.

H.17.4 General Rules

(1) During the execution of a DISPLAY statement, the contents of *literal-1* are displayed on the user's display device at the current cursor position (see LINE and COLUMN clauses).

(2) Literals are displayed DIM unless the BOLD/BRIGHT/HIGHLIGHT attribute was specified.

VI. PROCEDURE DIVISION

A. General Description

The Procedure Division contains procedures to be executed by the object program. The Procedure Division is optional in a COBOL source program.

A.1. DECLARATIVES

Declarative sections must be grouped at the beginning of the Procedure Division preceded by the keyword DECLARATIVES and followed by the keywords END DECLARATIVES.

A.2. Procedures

A procedure is composed of a paragraph, or a group of successive paragraphs, or a section, or a group of successive sections within the Procedure Division. If one paragraph is in a section, all paragraphs must be in sections. A procedure-name is a word used to refer to a paragraph or section in the source program in which it occurs. It consists of a paragraph-name (which may be qualified) or a section-name.

A section consists of a section header followed by zero, one, or more successive paragraphs. A section ends immediately before the next section or at the end of the Procedure Division or, in the declaratives portion of the Procedure Division, at the keywords END DECLARATIVES.

A paragraph consists of a paragraph-name followed by a period and a space and by zero, one, or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name or at the end of the Procedure Division or, in the declaratives portion of the Procedure Division, at the keywords END DECLARATIVES. A sentence consists of one or more statements and is terminated by the separator period.

A statement is a syntactically valid combination of words, literals, and separators beginning with a COBOL verb.

The term `identifier' is defined as the word or words necessary to make unique reference to a data item.

A.3. Execution

Execution begins with the first statement of the Procedure Division, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules indicate some other order.

The general formats of the Procedure Division are shown below.

Format 1:

PROCEDURE DIVISION [USING { *data-name-1* }...] .

[DECLARATIVES.

{ *section-name* SECTION [*segment-number*] .

USE *statement*.

[*paragraph-name*.

[*sentence*]...]... }...

END DECLARATIVES.]

{ *section-name* SECTION [*segment-number*] .

[*paragraph-name*.

[*sentence*]...]... }...

Format 2:

PROCEDURE DIVISION [USING { *data-name-1* }...] .

{ *paragraph-name*.

[*sentence*]... }...

B. Concepts

B.1. Arithmetic Expressions

B.1.1 Definition of an Arithmetic Expression

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, the figurative constant ZERO (ZEROS, ZEROES), such identifiers, figurative constants, and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator. The permissible combinations of identifiers, numeric literals, arithmetic operators, and parentheses are given in the table, Combination of Symbols in Arithmetic Expressions, below.

Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

B.1.2 Arithmetic Operators

There are five binary arithmetic operators and two unary arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters that must be preceded by a space and followed by a space.

Binary Arithmetic Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary Arithmetic Operator	Meaning
+	The effect of multiplication by the numeric literal +1
-	The effect of multiplication by the numeric literal -1

B.1.3 Formation and Evaluation Rules

(1) Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first, and, within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

1st - Unary plus and minus
2nd - Exponentiation
3rd - Multiplication and division
4th - Addition and subtraction

(2) Parentheses are used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, or to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation from the normal precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

(3) The ways in which identifiers, literals, operators, and parentheses may be combined in an arithmetic expression are summarized in the table below, where:

a. The letter `P' indicates a permissible pair of symbols.

b. The character `-' indicates an invalid pair.

FIRST SYMBOL	SECOND SYMBOL				
	Identifier or Literal	+ - * / **	Unary + or -	(	)
Identifier or Literal	-	P	-	-	P
+ - * / **	P	-	P	P	-
Unary + or -	P	-	-	P	-
(	P	-	P	P	-
)	-	-	-	-	P

TABLE 15. Combination of Symbols in Arithmetic Expressions

(4) An arithmetic expression may only begin with the symbol `(', `+', `-', an identifier, or a literal and may only end with a `)', an identifier, or a literal. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right

parenthesis. If the first operator in an arithmetic expression is a unary operator, it must be immediately preceded by a left parenthesis if that arithmetic expression immediately follows an identifier or another arithmetic expression.

(5) The following rules apply to evaluation of exponentiation in an arithmetic expression:

a. If the value of an expression to be raised to a power is zero, the exponent must have a value greater than zero. Otherwise, the size error condition exists.

b. If the evaluation yields both a positive and a negative real number, the value returned as the result is the positive number.

c. If no real number exists as the result of the evaluation, the size error condition exists.

(6) Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items.

(7) (***ISQL***) The following table summarizes the valid arithmetic operations involving items of class date-time and interval.

First Operand	Operator(s)	Second Operand	Result
date-time	-	date-time	interval
date-time	+ -	interval	date-time
interval	+	date-time	date-time
interval	+ -	interval	interval
interval	+ - * /	number	interval
number	+ *	interval	interval

(8) (***ISQL***) The following rules apply to arithmetic operations involving items of class date-time and interval:

a. If both operands are of class date-time, they must both have the same category.

b. If both operands are of class interval, they must both have the same category. The result is of the same category with a span of fields that encompasses the span of fields of both operands. For example. Adding a DAY TO HOUR interval to an HOUR TO MINUTE interval will yield a DAY TO MINUTE interval as the result.

c. If one operand is class date-time and the other operand is class interval, the category of the interval operand must be defined such that it contains date-time fields that are also contained in the date-time operand. The category of the date-time result is of the same as the category of the date-time operand.

d. The difference of two timestamp operands is a day-time interval; of two date operands is a year-month interval; and of two time operands is a day-time interval.

e. The computation of an interval combined with a number is accomplished by first converting the interval into an equivalent interval value consisting of just the lowest order field, performing the arithmetic on that value, discarding any fraction that cannot be contained in the field, and then converting back to the original interval (normalized). For example, INTERVAL "4:30.25" MINUTE TO SECOND(2) / 2 is handled by converting to the equivalent INTERVAL "270.25" SECOND (3, 2), dividing by 2 to yield INTERVAL "135.12" SECOND (3,2), discarding the .005 second fraction, and then converting back to the original format INTERVAL "2:15.12" MINUTE TO SECOND (2).

B.2. Conditional Expressions

Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition. A conditional expression has a truth value

represented by either true or false. Conditional expressions are specified in the EVALUATE, IF, PERFORM, and SEARCH statements. There are two categories of conditions associated with conditional expressions: simple conditions and complex conditions. Each may be enclosed within any number of paired parentheses, in which case its category is not changed.

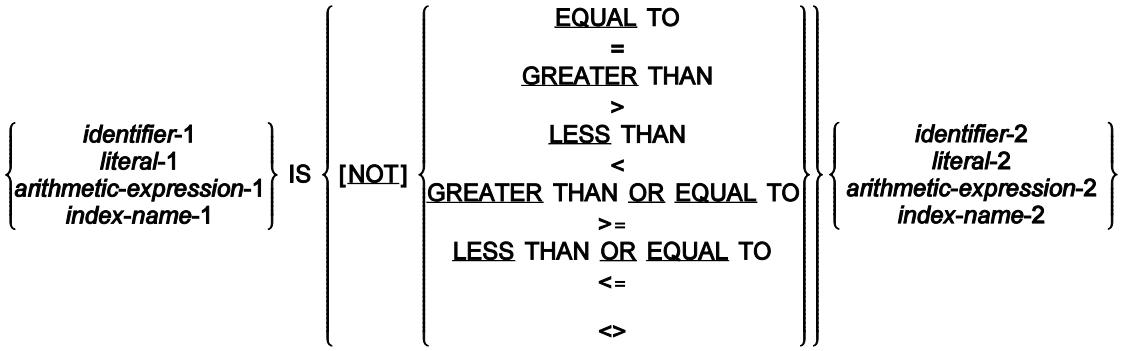
B.2.1 Simple Conditions

The simple conditions are the relation, class, condition-name, switch-status, and sign conditions. A simple condition has a truth value of true or false. The inclusion in parentheses of simple conditions does not change the simple condition truth value.

B.2.1.1 Relation Condition

A relation condition causes a comparison of two operands, each of which may be the data item referenced by an identifier, a literal, the value resulting from an arithmetic-expression, or an index-name. A relation condition has a truth value of true if the relation exists between the operands. Comparison of two numeric operands is permitted regardless of the formats specified in their respective USAGE clauses. However, for all other comparisons, the operands must have the same usage. If either of the operands is a group item, the nonnumeric comparison rules apply. Comparisons involving POINTER items have their own explicit rules. See section B.2.1.1.4 on page 235, 236.

The format for a relation condition is as follows:



The first operand (*identifier-1*, *literal-1*, *arithmetic-expression-1*, or *index-name-1*) is called the subject of the condition; the second operand (*identifier-2*, *literal-2*, *arithmetic-expression-2*, or *index-name-2*) is called the object of the condition. The relation condition must contain at least one reference to a variable.

The relational operators specify the type of comparison to be made in a relation condition. A space must precede and follow each reserved word comprising the relational operator. When used, NOT and the next keyword or relation character are one relational operator that defines the comparison to be executed for truth value. The following relational operators are equivalent:

IS NOT GREATER THAN is equivalent to IS LESS THAN OR EQUAL TO;

IS NOT LESS THAN is equivalent to IS GREATER THAN OR EQUAL TO.

IS <> is equivalent to IS NOT =.

IS NOT <> is equivalent to IS =.

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than OR not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than OR not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to OR not equal to
IS [NOT] GREATER THAN OR EQUAL TO IS [NOT] >=	Greater than or equal to OR not greater than or equal to
IS [NOT] LESS THAN OR EQUAL TO IS [NOT] <=	Less than or equal to OR not less than or equal to
IS <>	Not equal to

TABLE 16. Relational Operators

B.2.1.1.1 Comparison of Numeric Operands

For operands whose class is numeric, a comparison is made with respect to the algebraic value of the operands. The length of the literal or arithmetic-expression operands, in terms of the number of digits represented, is not significant. Zero is considered a unique value regardless of the sign.

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparison.

B.2.1.1.2 Comparison of Nonnumeric Operands

For nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters. If one of the operands is specified as numeric, it must be an integer data item or an integer literal and:

(1) If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were moved to an elementary alphanumeric data item of the same size as the numeric data item (in terms of standard data format characters), and the content of this alphanumeric data item were then compared to the nonnumeric operand.

(2) If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size as the numeric data item (in terms of standard data format characters), and the content of this item were then compared to the nonnumeric operand.

(3) A non-integer numeric operand cannot be compared to a nonnumeric operand.

The size of an operand is the total number of standard data format characters in the operand. Numeric and nonnumeric operands may be compared only when their usage is the same.

When comparing two nonnumeric operands there are two cases to consider: operands of equal size and operands of unequal size.

(1) Operands of equal size. If the operands are of equal size, comparison effectively proceeds by comparing characters in corresponding character positions starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of corresponding characters are equal.

The first encountered pair of unequal characters is compared to determine their relative position in the collating sequence. The operand that contains the character that is positioned higher in the collating sequence is considered to be the greater operand.

(2) Operands of unequal size. If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by sufficient spaces to make the operands of equal size.

B.2.1.1.3 Comparisons Involving Index-Names and/or Index Data Items

Relation tests may be made only between:

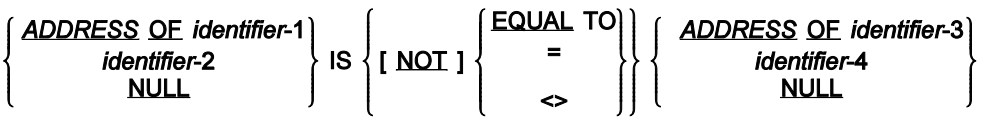
(1) Two index-names. The result is the same as if the corresponding occurrence numbers were compared.

(2) An index-name and a data item (other than an index data item) or literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

(3) An index data item and an index-name or another index data item. The actual values are compared without conversion.

B.2.1.1.4 Comparisons Involving USAGE POINTER Data Items (***ANSI 74*** and ***ANSI 85***)

Two data items that are explicitly or implicitly defined as USAGE POINTER can be compared. Pointer comparisons can include only relational operators which test for equality or inequality. The general format of such comparisons is:



Syntax Rules:

(1) *Identifier-1* and *identifier-3* can refer to any data items defined in the Data Division.

(2) *Identifier-2* and *identifier-4* must be defined as USAGE IS POINTER.

General Rules:

(1) If ADDRESS OF clause is specified, the address if the named identifier is referenced, not the contents of the identifier.

(2) The operands are equal if the two address are identical. Otherwise, they are unequal.

(3) This type of condition is allowed in the IF, PERFORM and Format 1 SEARCH statement. It is not allowed in a Format 2 SEARCH statement (SEARCH ALL) since there is no implied ordering to pointer data items.

B.2.1.1.5 Comparisons Involving Date-Time Items (***ISQL***)

General Rules:

(1) Two items of class date-time may be compared provided that they have the same category.

(2) Comparisons are performed in accordance with chronological ordering.

B.2.1.1.6 Comparisons Involving Interval Items (***ISQL***)

General Rules:

(1) Two items of class interval may be compared provided that they have the same category.

(2) An item of class interval and a numeric item may be compared provided that the interval item consists of only a single date-time field. The interval items is treated as a signed integer item for the purpose of the comparison.

(3) Comparisons are performed in accordance with the sign and magnitude.

(4) When the set of fields of the two intervals match, the comparison is straightforward and proceeds field by field from left to right.

(5) When the set of fields of the two intervals does not match, the comparison extends either operand as necessary with additional fields of value zero such that the set of fields matches. The extended values are 'normalized' by beginning with the rightmost field and normalizing it to its usual range and carrying any overflow to the next field to the left. Comparison then proceeds as in rule 4.

For example:

Taking the comparison (INTERVAL "1:45" HOUR TO MINUTE < INTERVAL "105:23" MINUTE TO SECOND) and applying the rules of extension above, we would have (INTERVAL "1:45:00" HOUR TO SECOND < INTERVAL "0:105:23" HOUR TO SECOND) and normalizing the second operand yields (INTERVAL "1:45:00" HOUR TO SECOND < INTERVAL "1:45:23" HOUR TO SECOND), which evaluates to TRUE.

B.2.1.2 Class Condition

The class condition determines whether an operand is numeric, alphabetic, or contains only the characters in the set of characters specified by the CLASS clause as defined in the SPECIAL-NAMES paragraph of the Environment Division.

B.2.1.2.1 General Format

ANSI 74 and ANSI 85

identifier IS [NOT] { NUMERIC / ALPHABETIC / ALPHABETIC-LOWER / ALPHABETIC-UPPER / *class-name*-1 }

VXCOBOL

identifier IS [NOT] { NUMERIC / ALPHABETIC }

B.2.1.2.1 Syntax Rules

(1) If the NUMERIC phrase is specified, the usage of the operand being tested must be described as DISPLAY, except for ***VXCOBOL*** where the NUMERIC test will allow any numeric item.

(2) If the NUMERIC phrase is specified, the operand being tested must not be an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational sign(s).

(3) If the NUMERIC phrase is **not** specified, the usage of the operand being tested must be described as DISPLAY.

(4) If the ALPHABETIC, ALPHABETIC-UPPER, ALPHABETIC-LOWER, or *class-name-1* phrase is specified, the operand being tested must not be an item whose data description describes the item as numeric.

B.2.1.2.2 General Rules

(1) When used, NOT and the next keyword specify one class condition that defines the class test to be executed for truth value and which has the opposite truth value from the test without the NOT. So, e.g., NOT NUMERIC is a truth test for determining that an operand is nonnumeric. The remaining rules are expressed in terms of the truth of the condition expressed without the NOT.

(2) When the operand being tested is a zero-length item, the result of the test is always false.

(3) If the NUMERIC phrase is specified, the following rules apply:

a. If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the content is numeric and an operational sign is not present. If the data description of the item does indicate the presence of an operational sign, the item being tested is determined to be numeric only if the content is numeric and a valid operational sign is present. Valid operational signs for data items described with the SIGN IS SEPARATE clause are the standard data format characters + and -; see The USAGE clause on page 189, 192, 226 for more information.

b. (***VXCOBOL***) The result is true for a usage display operand if it consists entirely of the characters 0, 1, 2, ... 9 and space with or without the operation sign; however, if the '-G a' compiler switch is used, space is not allowed. The result is true for non-display identifiers if their content is in agreement with the data description.

c. (***ANSI 74*** and ***ANSI 85***) The result is true if the operand consists entirely of the characters 0, 1, 2, 3, ... , 9, with or without an operational sign.

(4) If the ALPHABETIC phrase is specified, the following rules apply:

a. (***ANSI 74*** and ***VXCOBOL***) The result is true if the operand consists entirely of the uppercase letters A, B, C, ... , Z, or space, or any combination of the uppercase letters and spaces.

b. (***ANSI 85***) The result is true if the operand consists entirely of the uppercase letters A, B, C, ... , Z, space, or the lowercase letters a, b, c, ... , z, or any combination of the uppercase and lowercase letters and spaces.

(5) If the ALPHABETIC-LOWER phrase is specified, the result is true if the operand consists entirely of the lowercase letters a, b, c, ... , z, or space, or any combination of the lowercase letters and spaces.

(6) If the ALPHABETIC-UPPER phrase is specified, the result is true if the operand consists entirely of the uppercase letters A, B, C, ... , Z, or space, or any combination of the uppercase letters and spaces.

(7) If the *class-name-1* phrase is specified, the result is true if the operand consists entirely of the characters listed in the definition of *class-name-1* in the SPECIAL-NAMES paragraph.

B.2.1.3 Condition-Name Condition (Conditional Variable)

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with *condition-name-1*. The general format for the condition-name condition is as follows:

condition-name-1

If *condition-name-1* is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range, including the end values.

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

The result of the test is true if one of the values corresponding to *condition-name-1* equals the value of its associated conditional variable.

B.2.1.4 Switch-Status Condition

A switch-status condition determines the on or off status of an external switch. The switch-name and the on or off value associated with the condition must be named in the SPECIAL-NAMES paragraph of the Environment Division. (See The SPECIAL-NAMES paragraph on page 74 for the description of switch conditions.) The general format for the switch-status condition is as follows:

condition-name-1

The result of the test is true if the switch is set to the specified position corresponding to *condition-name-1*.

B.2.1.5 Sign Condition

The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than, or equal to zero. The general format for a sign condition is as follows:

arithmetic-expression-1 IS [NOT] { POSTIVE / NEGATIVE / ZERO }

When used, NOT and the next keyword specify one sign condition that defines the algebraic test to be executed for truth value; e.g., NOT ZERO is a truth test for a nonzero (positive or negative) value. An operand is positive, if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero. *Arithmetic-expression-1* must contain at least one reference to a variable.

B.2.1.6 (***ISQL***) Indicator Condition

The indicator condition determines the status of an indicator value. The general format for an indicator condition is as follows:

identifier IS [NOT] { NULL / VALID / OVERFLOW }

When used, NOT and the next keyword specify one indicator condition that defines the test to be executed for truth value. Since there are only three valid values, the NOT test is a test for either one of the values other than the one specified, e.g., NOT VALID is a truth test for either NULL or OVERFLOW. An indicator value of NULL means

that the database item was a NULL item and the corresponding data item was not set. An indicator value of VALID means that the database item was fetched and stored in the corresponding data item. An indicator value of OVERFLOW means that the database item was fetched, but it had to be truncated in order to be stored in the data item.

B.2.2 Complex Conditions

A complex condition is formed by combining simple conditions and/or complex conditions with logical connectors (logical operators `AND' and `OR') or by negating these conditions with logical negation (the logical operator `NOT')'. The truth value of a complex condition, whether parenthesized or not, is that truth value which results from the interaction of the stated logical operators on its constituent conditions.

The logical operators and their meanings are:

Logical Operator	Meaning
AND	Logical conjunction; the truth value is true if both of the conjoined conditions are true; false if both of the conjoined conditions is false.
OR	Logical inclusive OR; the truth value is true if one or both of the included conditions is true; false if both included conditions are false.
NOT	Logical negation or reversal of truth value; the truth value is true if the condition is false; false if the condition is true.

The logical operators must be preceded by a space and followed by a space.

B.2.2.1 Negated Conditions

A condition is negated by use of the logical operator `NOT' which reverses the truth value of the condition to which it is applied. Thus, the truth value of a negated condition is true if and only if the truth value of the condition being negated is false; the truth value of a negated condition is false if and only if the truth value of the condition being negated is true. Including a negated condition in parentheses does not change its truth value.

The general format for a negated condition is:

NOT *condition-1*

B.2.2.2 Combined Conditions

A combined condition results from connecting conditions with one of the logical operators `AND' or `OR'. The general format of a combined condition is:

condition { **AND** / **OR** } *condition* }...

B.2.2.3 Precedence of Logical Operators and the Use of Parentheses

In the absence of the relevant parentheses in a complex condition, the precedence (i.e., binding power) of the logical operators determines the conditions to which the specified logical operators apply and implies the equivalent parentheses. The order of precedence is `NOT', `AND', `OR'. Thus, specifying `*condition-1* OR NOT *condition-2* AND *condition-3*' implies and is equivalent to specifying `*condition-1* OR ((NOT *condition-2*) AND *condition-3*)'.

Where parentheses are used in a complex condition, they determine the binding of conditions to logical operators. Parentheses can, therefore, be used to depart from the normal precedence of logical operators as specified above. Thus, the example complex condition above can be given a different meaning by specifying it as `(*condition-1* OR (NOT *condition-2*)) AND *condition-3*'.

The following table indicates the ways in which conditions and logical operators may be combined and parenthesized. There must be a one-to-one correspondence between left and right parentheses such that each left parenthesis is to the left of its corresponding right parenthesis.

Given the following element:	In a conditional expression:		In a left-to-right sequence of elements:	
	May element be first?	May element be last?	Element, when not first, may be immediately followed by only:	Element, when not last, may be immediately followed by only:
simple-condition	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR or AND	No	No	simple-condition,)	simple-condition, NOT, (
NOT	Yes	No	OR, AND, (	simple-condition, (
(	Yes	No	OR, NOT, AND, (	simple-condition, NOT, (
)	No	Yes	simple-condition,)	OR, AND,)

TABLE 17. Combinations of Conditions, Logical Operators, and Parentheses

Thus, the element pair `OR NOT' is permissible, while the pair `NOT OR' is not permissible; the pair `NOT (' is permissible, while the pair `NOT NOT' is not permissible.

B.2.3 Abbreviated Combined Relation Conditions

When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence such that a succeeding relation condition contains a subject or subject and relational operator that is common with the preceding relation condition, and no parentheses are used within such a consecutive sequence, any relation condition except the first may be abbreviated by:

(1) The omission or the subject of the relation condition, or

(2) The omission of the subject and relational operator of the relation condition.

The format for an abbreviated combined relation condition is:

relation-condition { { **AND** / **OR** } [NOT] [*relational-operator*] *object* }...

Within a sequence of relation conditions both of the above forms of abbreviation may be used. The effect of using such abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator. The result of such implied insertion must comply with the rules of TABLE 17. This insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.

The interpretation applied to the use of the word NOT in an abbreviated combined relation condition is as follows:

(1) If the word immediately following NOT is GREATER, >, LESS, <, EQUAL, =, then the NOT participates as part of the relational operator; otherwise,

(2) The NOT is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

Some examples of abbreviated combined and negated combined relation conditions and expanded equivalents follow.

Abbreviated Combined Relation Condition	Expanded Equivalent
a > b AND NOT < c OR d	((a > b) AND (a NOT < c)) OR (a NOT < d)
a NOT EQUAL b OR c	(a NOT EQUAL b) OR (a NOT EQUAL c)
NOT a = b OR c	(NOT (a = b)) OR (a = c)
NOT (a GREATER b OR < c)	NOT ((a GREATER b) OR (a < c))
NOT (a NOT > b AND c AND NOT d)	NOT (((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d)))

EXAMPLE 14. Abbreviated combined and negated combined relation conditions

B.2.4 Order of Evaluation of Conditions

Parentheses, both explicit and implicit, denote a level of inclusiveness within a complex condition. Two or more conditions connected by only the logical operator `AND' or only the logical operator `OR' at the same level of inclusiveness establish a hierarchical level within a complex condition. Thus, an entire complex condition may be considered to be a nested structure of hierarchical levels with the entire complex condition itself being the most inclusive hierarchical level. Within this context, the evaluation of the conditions within an entire complex condition begins at the left of the entire complex condition and proceeds according to the following rule recursively applied where necessary:

(1) The constituent connected conditions within a hierarchical level are evaluated in order from left to right, and evaluation of that hierarchical level proceeds until all the constituent connected conditions within that hierarchical level have been evaluated.

Negated conditions are evaluated when it is necessary to evaluate the complex condition that they represent.

Application of the above rules is shown in the 4 figures that follow.

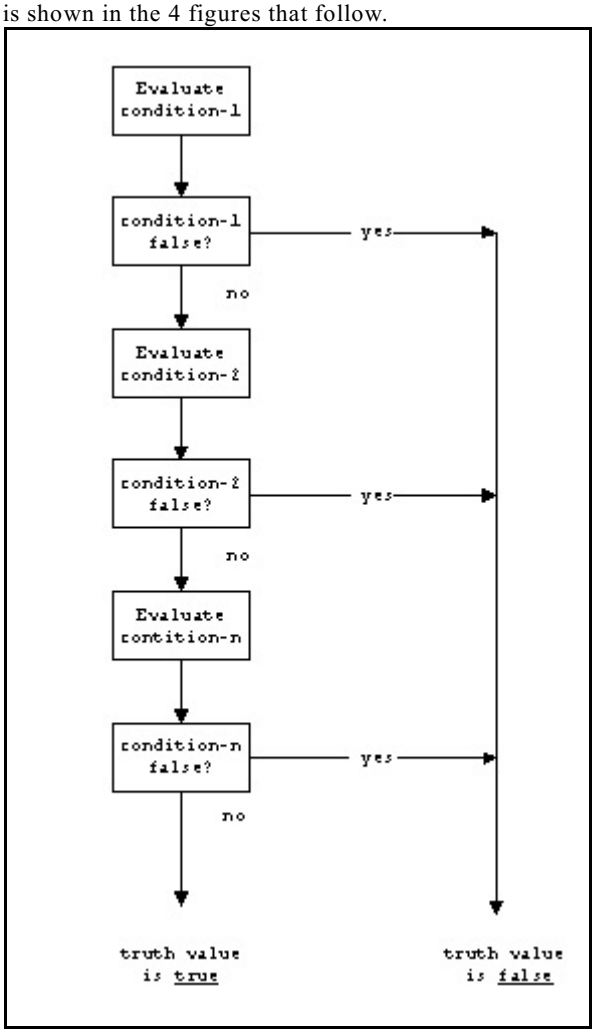


FIGURE 1. Evaluation of *condition-1* AND *condition-2* AND ... *condition-n*

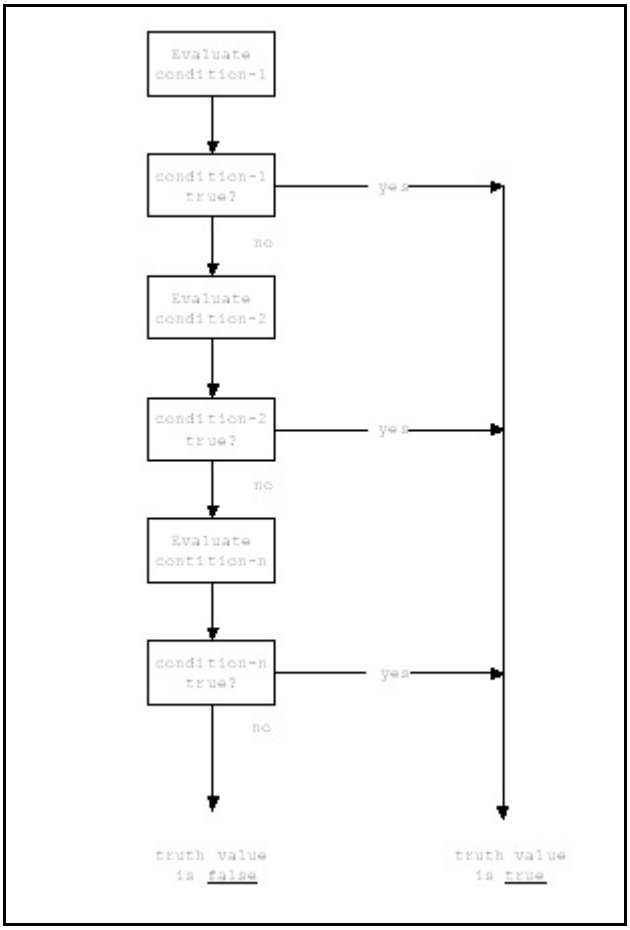


FIGURE 2. Evaluation of *condition-1* OR *condition-2* OR ... *condition-n*

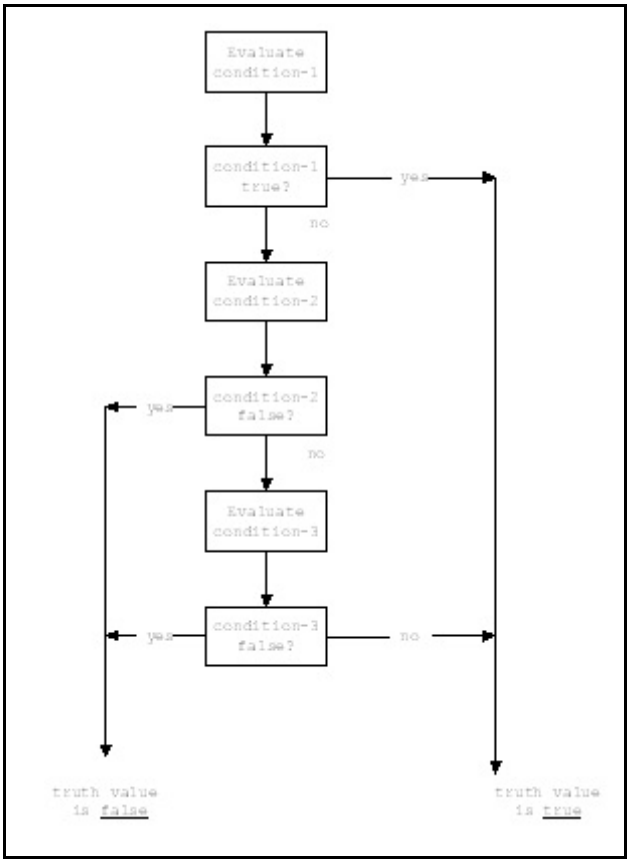


FIGURE 3. Evaluation of *condition-1* OR *condition-2* AND *condition-3*

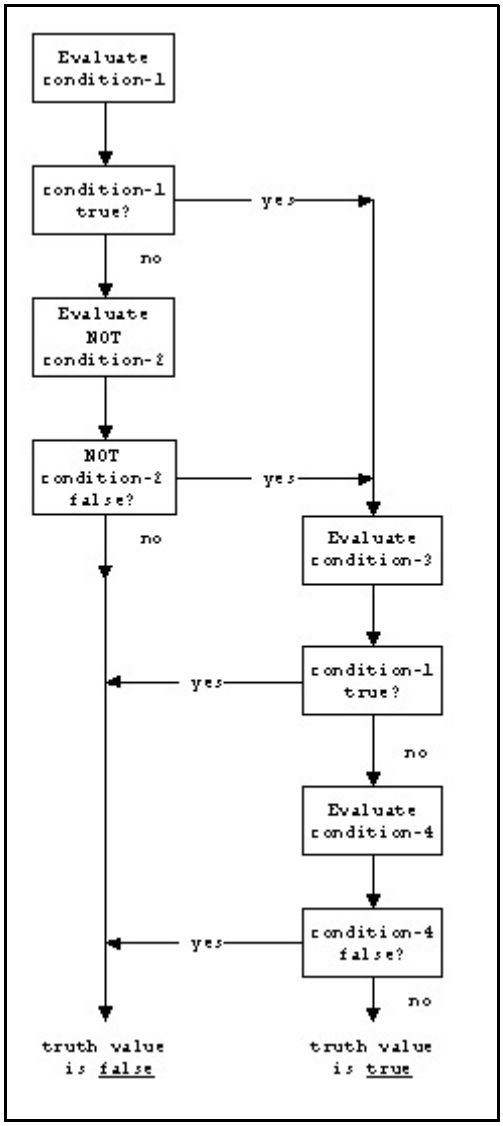


FIGURE 4. Evaluation of (*condition-1* OR NOT *condition-2*) AND *condition-3* AND *condition-4*

B.3. Common Options and Rules for Statements

Paragraph B and its subordinate paragraphs provide a description of the common options and conditions that pertain to or appear in several different statements.

B.3.1 ROUNDED Phrase

If, after decimal point alignment, the number of places in the fractions of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant identifier, truncation is relative to the size provided for the resultant identifier. When rounding is requested, the absolute value of the resultant identifier is increased by one in the low-order position whenever the most significant digit of the excess is greater than or equal to five.

When the low-order integer positions in a resultant identifier are represented by the character P in the PICTURE for that resultant identifier, rounding or truncation occurs relative to the right-most integer position for which storage is allocated.

B.3.2 ON SIZE ERROR Phrase

The size error condition occurs under the following circumstances:

(1) Violation of the rules for evaluation of exponentiation always terminates the arithmetic operation and always causes a size error condition.

(2) Division by zero always terminates the arithmetic operation and always causes a size error condition.

(3) If, after radix point alignment, the absolute value of a result exceeds the largest value that can be contained in the associated resultant identifier, a size error condition exists. If the ROUNDED phrase is specified, rounding takes place before checking for size error.

(4) (***ISQL***) If, after adding or subtracting an interval from a date-time value, the resulting date-time value is not a valid date-time value, the size error condition exists. For example, DATE "2001-01-30" + INTERVAL "1" MONTH yields DATE "2001-02-30", which is not a valid date.

If the ON SIZE ERROR phrase is specified and a size error condition exists after the execution of the arithmetic operations specified by an arithmetic statement, the values of the affected resultant identifiers remain unchanged from the values they had before execution of the arithmetic statement. The values of resultant identifiers for which no size error condition exists are the same as they would have been if the size error condition had not resulted for any of the resultant identifiers. After completion of the arithmetic operations, control is transferred to the imperative-statement specified in the ON SIZE ERROR phrase and execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the ON SIZE ERROR phrase, control is transferred to the end of the arithmetic statement and the NOT ON SIZE ERROR phrase, if specified, is ignored.

If the ON SIZE ERROR phrase is not specified and a size error condition exists after the execution of the arithmetic operations specified by an arithmetic statement, the values of the affected resultant identifiers are undefined. The values of resultant identifiers for which no size error condition exists are the same as they would have been if the size error condition had not resulted for any of the resultant identifiers. After completion of the arithmetic operations, control is transferred to the end of the arithmetic statement and the NOT ON SIZE ERROR phrase, if specified, is ignored.

If the size error condition does not exist after the execution of the arithmetic operations specified by an arithmetic statement, the ON SIZE ERROR phrase, if specified, is ignored and control is transferred to the end of the arithmetic statement or to the imperative-statement specified in the NOT ON SIZE ERROR phrase, if it is specified. In the latter case, execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the NOT ON SIZE ERROR phrase, control is transferred to the end of the arithmetic statement

For the ADD or SUBTRACT statement with the CORRESPONDING phrase, if any of the individual operations produces a size error condition, *imperative-statement-1* in the ON SIZE ERROR phrase is not executed until all of the individual additions or subtractions are completed.

B.3.3 CORRESPONDING Phrase

For the purpose of this discussion, D1 and D2 must each be identifiers that refer to group items. A pair of data items, one from D1 and one from D2 correspond if the following conditions exist:

(1) A data item in D1 and a data item in D2 are not designated by the keyword FILLER and have the same data-name and the same qualifiers up to, but not including, D1 and D2.

(2) At least one of the data items is an elementary data item and the resulting move is legal according to the move rules in the case of a MOVE statement with the CORRESPONDING phrase; and both of the data items are elementary numeric data items in the case of the ADD statement with the CORRESPONDING phrase or the SUBTRACT statement with the CORRESPONDING phrase.

(3) The description of D1 and D2 must not contain level-number 66, 77, or 88, the USAGE IS INDEX clause, or (for ***ANSI 74*** and ***ANSI 85***) the USAGE IS POINTER clause.

(4) A data item that is subordinate to D1 or D2 and contains a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause is ignored, as well as those data items subordinate to the data item that contains the REDEFINES, OCCURS, USAGE IS INDEX clause, or (for ***ANSI 74*** and ***ANSI 85***) the USAGE IS POINTER clause.

(5) The name of each data item which satisfies the above conditions must be unique after application of the implied qualifiers.

The following examples demonstrate the MOVE CORRESPONDING and ADD CORRESPONDING statements.

```
       FD PATIENT-FILE.
       01 PATIENT-RECORD.
           03 PATIENT-KEY.
              05 PATIENT-NO                 PIC 9(6).
              05 PATIENT-EMPLOYER           PIC X(30).
           03 PATIENT-NAME                  PIC X(20).
           03 PATIENT-INSURANCE-CO          PIC X(15).
           03 PATIENT-INS-GROUP-NO          PIC 9(3).
           03 TODAYS-CHARGES                PIC 9(4)V99.
           03 PATIENT-BALANCE.
              05 0-30                       PIC 9(4)V99.
              05 31-60                      PIC 9(4)V99.
              05 OVER-60                    PIC 9(4)V99.
       01 BILL-DETAIL-LINE.
           03 PATIENT-NAME                  PIC X(20).
           03 FILLER                        PIC X(5) VALUE SPACE.
           03 TODAYS-CHARGES.               PIC 9(4)V99.
           03 FILLER                        PIC X(5) VALUE SPACE.
           03 PREVIOUS-BALANCE              PIC 9(4)V99.
           03 TOTAL-BALANCE                 PIC 9(6)V99.
       01 ACCTS-REC-TOTALS
           03 SUPPLIER-BALANCE              PIC 9(8)V99.
           03 PATIENT-BALANCE.
              05 0-30                       PIC 9(4)V99.
              05 31-60                      PIC 9(4)V99.
              05 OVER-60                    PIC 9(4)V99.
*****************************************************************************
*** The following MOVE statement is the equivalent to:
***
***     MOVE PATIENT-NAME OF PATIENT-RECORD
***             TO PATIENT-NAME OF BILL-DETAIL-LINE.
***     MOVE TODAYS-CHARGES OF PATIENT-RECORD
***             TO TODAYS-CHARGES OF BILL-DETAIL-LINE.
*****************************************************************************
       MOVE CORR PATIENT-RECORD TO BILL-DETAIL-LINE.
*****************************************************************************
*** The following ADD statement is equivalent to:
***
***     ADD 0-30 OF PATIENT-BALANCE OF PATIENT-RECORD
***             TO 0-30 OF PATIENT-BALANCE OF ACCTS-REC-TOTALS.
***     ADD 31-60 OF PATIENT-BALANCE OF PATIENT-RECORD
***             TO 31-60 OF PATIENT-BALANCE OF ACCTS-REC-TOTALS.
***     ADD OVER-60 OF PATIENT-BALANCE OF PATIENT-RECORD
***             TO OVER-60 OF PATIENT-BALANCE OF ACCTS-REC-TOTALS.
*****************************************************************************
       ADD  CORR PATIENT-BALANCE OF PATIENT-RECORD TO PATIENT-BALANCE
              OF ACCTS-REC-TOTALS.
```

EXAMPLE 15. MOVE CORRESPONDING and ADD CORRESPONDING

The following code demonstrates the MOVE CORRESPONDING statement.

```
WORKING-STORAGE SECTION.
01 SYSTEM-DATE                              PIC 9(8) VALUE ZERO.
01 SYSTEM-DATE-R REDEFINES SYSTEM-DATE.
      03 SYSTEM-YEAR                        PIC 9(4).
      03 SYSTEM-MONTH                       PIC 9(2).
      03 SYSTEM-DAY                         PIC 9(2).
01 CURRENT-DATE.
      03 SYSTEM-MONTH                       PIC 9(2).
      03 SYSTEM-DAY                         PIC 9(2).
      03 SYSTEM-YEAR                        PIC 9(4).
01 CURRENT-DATE-R REDEFINES CURRENT-DATE    PIC 9(8).

        ACCEPT SYSTEM-DATE FROM DATE YYYYMMDD.
        MOVE CORRESPONDING SYSTEM-DATE-R TO CURRENT-DATE.
```

EXAMPLE 16. MOVE CORRESPONDING

B.3.4 Arithmetic Statements

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. They have several common features.

(1) The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation.

(2) The maximum size of each operand is 18 decimal digits. The composite of operands, which is a hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points, must not contain more than 18 decimal digits.

B.3.5 Overlapping Operands

When a sending and a receiving data item in any statement share a part or all of their storage areas, yet are not defined by the same data description entry, the result of the execution of such a statement is undefined. For statements in which the sending and receiving data items are defined by the same data description entry, the results of the execution of the statement may or may not be defined depending on the general rules associated with the applicable statement. If there are no specific rules addressing such overlapping operands, the results are undefined.

In the case of reference modification, the unique data item produced by reference modification is not considered to be the same data description entry as any other data description entry. Therefore, if an overlapping situation exists, the results of the operation are undefined.

B.3.6 Multiple Results in Arithmetic Statements

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements may have multiple results. Such statements behave as though they had been written in the following way:

(1) A statement whose execution accesses all data items that are part of the initial evaluation of the statement, performs any necessary arithmetic or combining of these data items and stores the result of this operation in a temporary location. See the individual statements for the rules indicating which items are part of the initial evaluation.

(2) A sequence of statements whose execution transfers or combines the value in this temporary location with each single resulting data item. These statements are considered to be written in the same left-to-right sequence that the multiple results are specified.

The result of the statement

```
ADD a, b, c, TO c, d(c), e
```

is equivalent to

```
ADD a, b, c GIVING temp
ADD temp TO c
ADD temp TO d(c)
ADD temp TO e
```

and the result of the statement

```
MULTIPLY a(i) BY i, a(i)
```

is equivalent to

```
MOVE a(i) TO temp
MULTIPLY temp BY i
MULTIPLY temp BY a(i)
```

in both cases, `temp' is an intermediate result item provided by the compiler.

B.3.7 Incompatible Data

Except for the class condition, when the content of a data item is referenced in the Procedure Division and the content of that data item is not compatible with the class specified for that data item by its PICTURE clause, then the result of such a reference is undefined.

B.4. Statements and Sentences

There are four types of statements: imperative statements, conditional statements, compiler directing statements, and delimited scope statements.

There are three types of sentences: imperative sentences, conditional sentences, and compiler directing sentences.

B.4.1 Conditional Statements and Sentences

B.4.1.1 Definition of Conditional Statement

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

A conditional statement is one of the following:

(1) An EVALUATE, IF, SEARCH, or RETURN statement.

(2) A READ statement that specifies the AT END, NOT AT END, INVALID KEY, or NOT INVALID KEY phrase.

(3) A WRITE statement that specifies the INVALID KEY, NOT INVALID KEY, END-OF-PAGE, or NOT END-OF-PAGE phrase.

(4) A DEFINE SUB-INDEX, DELETE, EXPUNGE SUB-INDEX, LINK SUB-INDEX, RETRIEVE, REWRITE, START, or UNDELETE statement that specifies the INVALID KEY or NOT INVALID KEY phrase.

(5) An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the ON SIZE ERROR or NOT ON SIZE ERROR phrase.

(6) A STRING or UNSTRING statement that specifies the ON OVERFLOW or NOT ON OVERFLOW phrase.

(7) A CALL statement that specifies the ON OVERFLOW, ON EXCEPTION, or NOT ON EXCEPTION phrase.

(8) A CALL PROGRAM statement that specifies the ON EXCEPTION or NOT ON EXCEPTION phrase.

(9) An ACCEPT statement that specifies ON ESCAPE or NOT ON ESCAPE.

(10) (***ISQL***) A COMMIT, CONNECT, DEALLOCATE, DISCONNET, EXECUTE, EXECUTE IMMEDIATE, FETCH, PREPARE, ROLLBACK, or SET CONNECTION statement that specifies ON SQLERROR or NOT ON SQLERROR.

(11) (***ISQL***) A GET DIAGNOSTICS statement that specifies the ON EXCEPTION or NOT ON EXCEPTION phrase.

B.4.1.1.1 Definition of Conditional Phrase

A conditional phrase specifies the action to be taken upon determination of the truth value of a condition resulting from the execution of a conditional statement.

A conditional phrase is one of the following:

(1) an AT END or NOT AT END phrase when specified within a READ statement.

(2) an INVALID KEY or NOT INVALID KEY phrase when specified within a DELETE, READ, REWRITE, START, UNDELETE, or WRITE statement.

(3) a SIZE ERROR or NOT SIZE ERROR phrase when specified within an ADD, COMPUTE, DIVIDE, MULTIPLY, or SUBTRACT statement.

(4) an ON OVERFLOW or NOT ON OVERFLOW phrase when specified within a STRING or UNSTRING statement.

(5) an ON OVERFLOW, ON EXCEPTION, NOT ON OVERFLOW, or NOT ON EXCEPTION phrase when specified within a CALL statement.

(6) an ON EXCEPTION or NOT ON EXCEPTION phrase when specified within a CALL PROGRAM statement.

(7) an END-OF-PAGE or NOT END-OF-PAGE phrase when specified with a WRITE statement.

(8) an ON ESCAPE or NOT ON ESCAPE phrase when specified with an ACCEPT statement.

(9) (***ISQL***) an ON SQLERROR or NOT ON SQLERROR phrase when specified with a COMMIT, CONNECT, DEALLOCATE, DISCONNECT, EXECUTE, EXECUTE IMMEDIATE, FETCH, PREPARE, ROLLBACK, SET CONNECTION statement.

(10) (***ISQL***) an ON EXCEPTION or NOT ON EXCEPTION phrase when specified within a GET DIAGNOSTICS statement.

B.4.1.2 Definition of Conditional Sentence

A conditional sentence is a conditional statement, optionally preceded by an imperative statement, terminated by the separator period.

B.4.2 Compiler Directing Statements and Sentences

B.4.2.1 Definition of Compiler Directing Statement

A compiler directing statement consists of a compiler directing verb and its operands. The compiler directing verbs are COPY and USE. A compiler directing statement causes the compiler to take a specific action during compilation.

B.4.2.2 Definition of Compiler Directing Sentence

A compiler directing sentence is a single compiler directing statement terminated by the separator period.

B.4.3 Imperative Statements and Sentences

B.4.3.1 Definition of Imperative Statement

An imperative statement begins with an imperative verb and specifies an unconditional action to be taken by the object program or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement may consist of a sequence of imperative statements, each possibly separated from the next by a separator. The imperative verbs are:

ACCEPT [7]	DISCONNECT [9]	LINK SUB-INDEX [2]	SET (**ISQL**) [9]
ADD [1]	DISPLAY	MERGE	SORT
CALL [5]	DIVIDE [1]	MOVE	START [2]
CALL PROGRAM [6]	EXECUTE [9]	MULTIPLY [1]	STOP
CANCEL	EXIT	OPEN	STRING [3]
CLOSE	EXPUNGE	PERFORM	SUBTRACT [1]
COMPUTE [1]	EXPUNGE SUB-INDEX [2]	PREPARE [9]	UNDELETE [2]
CONNECT [9]	FETCH [9]	READ [4]	UNSTRING [3]
CONTINUE	GET [6]	RELEASE	WRITE [8]
DEALLOCATE [9]	GO TO	RETRIEVE [2]	
DEFINE SUB-INDEX	INITIALIZE	REWRITE [2]	
DELETE [2]	INSPECT	SET	

[1] without the optional ON SIZE ERROR and NOT ON SIZE ERROR phrases
[2] without the optional INVALID KEY and NOT INVALID KEY phrases
[3] without the optional ON OVERFLOW and NOT ON OVERFLOW phrases
[4] without the optional AT END, NOT AT END, INVALID KEY, and NOT INVALID KEY phrases
[5] without the optional ON OVERFLOW, ON EXCEPTION, and NOT ON EXCEPTION phrases
[6] without the optional ON EXCEPTION and NOT ON EXCEPTION phrases
[7] without the optional ON ESCAPE and NOT ON ESCAPE phrases
[8] without the optional INVALID KEY, NOT INVALID KEY, END-OF-PAGE, and NOT END-OF-PAGE phrases
[9] without the optional ON SQLERROR and NOT ON SQLERROR phrases

Whenever `imperative-statement' appears in the general format of statements, `imperative-statement' refers to that sequence of consecutive imperative statements that must be ended by a period or by any phrase associated with a statement containing that `imperative-statement'.

(***ISQL***) The COMMIT, CONNECT, DEALLOCATE, DISCONNECT, EXECUTE, EXECUTE IMMEDIATE, FETCH, GET DIAGNOSTICS, PREPARE, ROLLBACK, and SET CONNECTION statements are only available when the ***ISQL*** feature-set is enabled.

B.4.3.2 Definition of Imperative Sentence

An imperative sentence is an imperative statement terminated by the separator period.

B.5. Scope of Statements

A delimited scope statement is any statement which includes its explicit scope terminator. (See section B.6.5 on page 255.)

When statements are nested within other statements, a separator period which terminates the sentence also implicitly terminates all nested statements.

Whenever any statement is contained within another statement, the next phrase of the containing statement following the contained statement terminates the scope of any unterminated contained statement.

When statements are nested within other statements which allow optional conditional phrases, any optional conditional phrase encountered is considered to be the next phrase of the nearest preceding unterminated statement with which that phrase is permitted to be associated according to the general format and the syntax rules for that statement, but with which no such phrase has already been associated. An unterminated statement is one which has not been previously terminated either explicitly or implicitly.

B.6. Explicit and Implicit Specifications

There are four types of explicit and implicit specifications that occur in COBOL source programs:

(1) Explicit and implicit Procedure Division references

(2) Explicit and implicit transfers of control

(3) Explicit and implicit attributes

(4) Explicit and implicit scope terminators

B.6.1 Explicit and Implicit Procedure Division References

A COBOL source program can reference data items either explicitly or implicitly in Procedure Division statements. An explicit reference occurs when the name of the referenced item is written in a Procedure Division statement or when the name of the referenced item is copied into the Procedure Division by the processing of a COPY statement. An implicit reference occurs when the item is referenced by a Procedure Division statement without the name of the referenced item being written in the source statement. An implicit reference also occurs, during the execution of a PERFORM statement, when the index or data item referenced by the index-name or identifier specified in the VARYING, AFTER, or UNTIL phrase is initialized, modified, or evaluated by the control mechanism associated with that PERFORM statement. Such an implicit reference occurs if and only if the data item contributes to the execution of the statement.

B.6.2 Explicit and Implicit Transfers of Control

The mechanism that controls program flow transfers control from statement to statement in the sequence in which they were written in the source program unless an explicit transfer of control overrides this sequence or there is no next executable statement to which control can be passed. The transfer of control from statement to statement occurs without the writing of an explicit Procedure Division statement, and, therefore, is an implicit transfer of control.

COBOL provides both explicit and implicit means of altering the implicit control transfer mechanism.

In addition to the implicit transfer of control between consecutive statements, implicit transfer of control also occurs when the normal flow is altered without the execution of a procedure branching statement. COBOL provides the following types of implicit control flow alterations which override the statement-to-statement transfers of control:

(1) If a paragraph is being executed under control of another COBOL statement (for example, PERFORM, USE, SORT, and MERGE) and the paragraph is the last paragraph in the range of the controlling statement, then an implied transfer of control occurs from the last statement in the paragraph to the control mechanism of the last executed controlling statement. Further, if a paragraph is being executed under the control of a PERFORM statement which paragraph is being executed under the control of a PERFORM statement which causes iterative execution, and that paragraph is the first paragraph in the range of that PERFORM statement, an implicit transfer of control occurs between the control mechanism associated with that PERFORM statement and the first statement in that paragraph for each iterative execution of the paragraph.

(2) When a SORT or MERGE statement is executed, an implicit transfer of control occurs to any associated input or output procedures.

(3) When any COBOL statement is executed which results in the execution of a declarative section, an implicit transfer of control to the declarative section occurs. Another implicit transfer of control occurs after execution of the declarative section, as described in paragraph 1 above.

An explicit transfer of control consists of an alteration of the implicit control transfer mechanism by the execution of a procedure branching or conditional statement. An explicit transfer of control can be caused only by the execution of a procedure branching or conditional statement. The procedure branching statement EXIT PROGRAM causes an explicit transfer of control only when the statement is executed in a called program.

In this document, the term `next executable statement' is used to refer to the next COBOL statement to which control is transferred according to the rules above and the rules associated with each language element.

There is no next executable statement when the program contains no Procedure Division or does contain the following:

(1) The last statement in a declarative section when the paragraph in which it appears is not being executed under the control of some other COBOL statement.

(2) The last statement in a declarative section when the statement is in the range of an active PERFORM statement executed in a different section and this last statement of the declarative section is not also the last statement of the procedure that is the exit of the active PERFORM statement.

(3) The last statement in a program when the paragraph in which it appears is not being executed under the control of some other COBOL statement in that program.

(4) A STOP RUN statement or EXIT PROGRAM statement that transfers control outside the COBOL program.

When there is no next executable statement and control is not transferred outside the COBOL program, the program flow of control is undefined unless the program execution is in the nondeclarative procedures portion of a program under control of a CALL statement, in which case an implicit EXIT PROGRAM statement is executed.

B.6.3 Explicit and Implicit Attributes

Attributes may be implicitly or explicitly specified. Any attribute which has been explicitly specified is called an explicit attribute. If an attribute has not been specified explicitly, then the attribute takes on the default specification. Such an attribute is known as an implicit attribute.

For example, the usage of a data item need not be specified, in which case a data item's usage is DISPLAY.

B.6.4 Scope Terminators

Scope terminators serve to delimit the scope of certain Procedure Division. Scope terminators are of two types: explicit and implicit.

B.6.5 Explicit Scope Terminators

The explicit scope terminators are the following:

END-ACCEPT	END-DISCONNECT	END-MULTIPLY	END-SET
END-ADD	END-DIVIDE	END-PERFORM	END-START
END-CALL	END-EXECUTE	END-PREPARE	END-STRING
END-COMMIT	END-EVALUATE	END-READ	END-SUBTRACT
END-COMPUTE	END-EXPUNGE	END-RETRIEVE	END-UNDELETE
END-CONNECT	END-FETCH	END-RETURN	END-UNSTRING
END-DEALLOCATE	END-GET	END-REWRITE	END-WRITE
END-DEFINE	END-IF	END-ROLLBACK	
END-DELETE	END-LINK	END-SEARCH	

B.6.6 Implicit Scope Terminators

The implicit scope terminators are the following:

(1) At the end of any sentence, the separator period which terminates the scope of all previous statements not yet terminated.

(2) Within any statement containing another statement, the next phrase of the containing statement following the contained statement terminates the scope of any unterminated contained statement. Examples of such phrases are ELSE, NOT AT END, etc.

C. File Concepts

A file is a collection of records which may be placed into or retrieved from a storage medium. The user not only chooses the file organization, but also chooses the file processing method and sequence. Although the file organization and processing method are restricted for sequential media, no such restrictions exist for mass storage media.

When describing the capabilities of COBOL programs to manipulate files, the following conventions are used. The term `file-name' means the user-defined word used in the COBOL source program to reference a file. The terms `file referenced by file-name' and `file' mean the physical file regardless of the file-name used in the COBOL program. The term `file connector' means the entity containing information concerning the file. All accesses to physical files occur through file connectors. In various implementations, the file connector is referred to as a file information table, a file control block, etc.

C.1. File Attributes

A file has several attributes which apply to the file at the time it is created and cannot be changed throughout the lifetime of the file. The primary attribute is the organization of the file, which describes its logical structure. Other fixed attributes of the file provided by the COBOL program are primary record key, alternate record keys, code set, the minimum and maximum logical record size, the record type (fixed or variable), the collating sequence of the keys for indexed files, the blocking factor, the padding character, and the record delimiter.

For ***ANSI 74*** and ***ANSI 85***, there are three organizations: sequential, relative, and indexed. For ***VXCOBOL***, there are four organizations: sequential, relative, indexed, and INFOS.

C.1.1 Sequential Organization

Sequential files are organized so that each record, except the last, has a unique successor record; each record, except the first, has a unique predecessor record. The successor relationships are established by the order of execution of WRITE statements when the file is created. Once established, successor relationships do not change except in the case where records are added to the end of a file.

A sequentially organized mass storage file has the same logical structure as a file on any sequential medium; however, a sequential mass storage file may be updated in place. When this technique is used, new records cannot be added to the file and each replaced record must be the same size as the original record.

C.1.2 Relative Organization

A file with relative organization is a mass storage file from which any record may be stored or retrieved by providing the value of its relative record number.

Conceptually, a file with relative organization comprises a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number. Each logical record in a relative file is identified by the relative record number of its storage area. For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in any of the first through the ninth record areas.

In order to achieve more efficient access to records in a relative file, the number of character positions reserved on the medium to store a particular logical record may be different from the number of character positions in the description of that record in the program.

C.1.3 Indexed Organization

A file with indexed organization is a mass storage file from which any record may be accessed by giving the value of a specified key in that record. For each key data item defined for the records of a file, an index is maintained. Each such index represents the set of values from the corresponding key data item in each record. Each index, therefore, is a mechanism which can provide access to any record in the file.

Each indexed file has a primary index which represents the primary record key of each record in the file. Each record is inserted in the file, changed, or deleted from the file based solely upon the value of its primary record key. The primary record key of each record in the file must be unique, and it must not be changed when updating a record. The primary record key is declared in the RECORD KEY clause of the file control entry for the file.

Alternate record keys provide alternative means of retrieval for the records of a file. Such keys are named in the ALTERNATE RECORD KEY clauses of the file control entry. The value of a particular alternate record key in each record need not be unique. When these values may not be unique, the DUPLICATES phrase is specified in the ALTERNATE RECORD KEY clause.

C.1.4 INFOS Organization (***VXCOBOL***)

A file with INFOS organization is a mass storage file from which any record may be accessed by giving the value of a specified key or keys. For each key data item defined for the records of a file, a subindex is maintained. Each such index represents the set of values from the corresponding key data item. One or more keys can be associated with each record in the file.

A key may or may not be contained within the record. A key may or may not be associated with a record.

A file with INFOS organization can have several modes of indexing:

(1) simple indexing (one index per file)

(2) alternate indexing (multiple paths to a record)

(3) multiple indexing

(4) multilevel indexing

ICOBOL requires the U/FOS data management software from Transoft, Inc. to provide INFOS support.

C.2. Logical Records

A logical record is the unit of data which is retrieved from or stored into a file. There are two types of records: fixed length and variable length. When a file is created, it is declared to contain either fixed length or variable length records. In any case, the content of the record does not reflect any information the implementor may add to the record on the physical storage medium (such as record length headers), nor does the length of the record used by the COBOL programmer reflect these additions.

C.2.1 Fixed Length Records

Fixed length records must contain the same number of character positions for all the records in the file. All input-output operations on the file can only process this one record size.

For ***ANSI 74*** and ***ANSI 85***, fixed length records may be explicitly selected by specifying a Format 1 RECORD clause in the file description entry for the file regardless of the individual record descriptions.

For ***VXCOBOL***, fixed length records may be explicitly selected by specifying RECORDING MODE IS FIXED clause in the file description entry for the file regardless of the individual record descriptions.

C.2.2 Variable Length Records (***ANSI 74*** and ***ANSI 85***)

Variable length records may contain differing numbers of character positions among the records on the file. To define variable length records explicitly, the VARYING phrase may be specified in the RECORD clause in the file description entry or the sort-merge file description entry for the file. The length of a record is affected by the data-item referenced in the DEPENDING phrase of the RECORD clause or the DEPENDING phrase of an OCCURS clause or by the length of the record description entry for this file. They may also be obtained with the RECORDING MODE IS VARIABLE clause, however this is obsolete and applies to sequential files only.

C.2.3 Variable Length Records (***VXCOBOL***)

Variable length records may contain differing numbers of character positions among the records on the file. Variable length records may be explicitly selected by selecting the RECORDING MODE IS VARIABLE clause in the file regardless of the individual record descriptions.

C.3. File Processing

A file can be processed by performing operations upon individual records or upon the file as a unit, or (for INFOS files when using the ***VXCOBOL*** dialect) by performing operations upon individual keys. Unusual conditions that occur during processing are communicated back to the program.

C.4. Record Operations

The ACCESS MODE clause of the file description entry specifies the manner in which the object program operates upon records within a file. The access mode may be sequential, random, or dynamic.

For files that are organized as relative, indexed, or INFOS, any of the three access modes can be used to access the file regardless of the access mode used to create the file. A file with sequential organization may only be accessed in sequential mode.

When a file is accessed in random mode, input-output statements are used to access the records in a programmer-specified order. With the indexed organization, the programmer specifies the desired record by placing the value of one of its record keys in a record key or an alternate record key data item.

With dynamic access mode, the programmer may change at will from sequential accessing to random accessing, using appropriate forms of input-output statements.

C.4.1 Sequential Access Mode

A file can be accessed sequentially irrespective of the file organization.

For *sequential organization*, the order of sequential access is the order in which the records were originally written. The START statement may be used to establish a starting point for a series of subsequent retrievals.

For *relative organization*, the order of sequential access is ascending or descending based on the value of the relative record numbers. Only records which currently exist in the file are made available. The START statement may be used to establish a starting point for a series of subsequent sequential retrievals.

For *indexed organization or INFOS*, the order of sequential access is ascending or descending based on the value of the key of reference according to the collating sequence associated with the native character set. Any of the keys associated with the file may be established as the key of reference during the processing of the file. The order of retrieval from a set of records which have duplicate key of reference values is the original order of arrival of those records into the set. The START statement may be used to establish a starting point within an indexed file for a series of subsequent sequential retrievals.

For ***VXCOBOL***, each individual I/O operation may be used to establish a starting point within an INFOS file for subsequent sequential retrievals.

C.4.2 Random Access Mode

When a file is accessed in random mode, input-output statements are used to access the records in a programmer-specified order. The random access mode may only be used with relative, indexed, or INFOS file organizations.

For a file with relative organization, the programmer specifies the desired record by placing its relative record number in a relative key data item. With the indexed organization, the programmer specifies the desired record by placing the value of one of its record keys in a record key or an alternate record key data item. With INFOS organization, the programmer specifies the desired record by placing the value of one or more of its record keys in appropriate record key data items.

C.4.3 Dynamic Access Mode

With dynamic access mode, the programmer may change at will from sequential accessing to random accessing, using appropriate forms of input-output statements. The dynamic access mode may only be used on files with relative, indexed, or INFOS organizations.

C.4.4 Open Mode

The open mode of the file is related to the actions to be performed upon is in the file. The open modes and purposes are: INPUT, to retrieve records; OUTPUT, to place records into a file; EXTEND, to append records to an existing file; and I-O, to retrieve and update records. The open mode is specified in the OPEN statement.

When the open mode is INPUT, a file may be accessed by the READ and for ***VXCOBOL*** the RETRIEVE statement. The START statement may also be used for files organized as indexed, relative, INFOS which are in sequential or dynamic access modes or for files organized as sequential.

When the open mode is OUTPUT, the records are placed into the file by issuing WRITE statements.

When the open mode is EXTEND, new records are added to the logical end of a file by issuing WRITE statements.

Only mass storage files may be referenced in the open I-O mode. The additional capabilities of mass storage devices permit updating in place, thus READ and REWRITE statements may always be used. A mass storage file may be updated in the same manner as a file on a sequential medium, by transcribing the entire file into another file (perhaps in a separate area of mass storage) using READ and WRITE statements. However, it is sometimes more efficient to update a mass storage file in place. This mass storage file maintenance technique uses the REWRITE statement to return to their previous locations on the storage medium only those records which have changed.

READ, REWRITE, and START statements are the only operations allowed, while updating in place sequentially organized files. However, for indexed, relative, or INFOS organized files, the following additional functions may be applied: the DELETE Statement may be used with any access mode to remove a record logically from a file; the UNDELETE Statement may be used with any access mode to add a record that had been logically removed from a file; the WRITE statement may be used in random or dynamic access mode to insert a new record into the file.

C.4.5 Current Volume Pointer

The current volume pointer is a conceptual entity used in this document to facilitate exact specification of the current physical volume of a sequential file. The status of the current volume pointer is affected by the CLOSE, OPEN, READ, and WRITE statements.

C.4.6 File Position Indicator

The file position indicator is a conceptual entity used in this document to facilitate exact specification of the next record to be accessed within a given file during certain sequences of input-output operations. The concept of a file position indicator has no meaning for a file opened in the output or extend mode.

For sequential, relative, and indexed files, the setting of the file position indicator is affected only by the OPEN, READ, and START statements. The file position indicator can be updated on all INFOS file operations.

C.5. File Operations

Several COBOL statements operate upon files as entities or as collections of records. These are the CLOSE, DELETE FILE, and OPEN statements. For ***VXCOBOL***, the EXPUNGE statement is also included.

C.6. Exception Handling

During the execution of any input or output operation, unusual conditions may arise which preclude normal completion of the operation. There are four methods by which these conditions are communicated to the object program; status keys, exception declaratives, optional phrases associated with the imperative statement, and the ACCEPT FROM EXCEPTION STATUS statement. If a fatal I/O error is encountered and the program terminates, the current Exception Status is displayed right after the current opcode location and current PC.

C.6.1 I-O Status (FILE STATUS)

The I-O status is a two-character conceptual entity whose value is set to indicate the status of an input-output operation during the execution of a CLOSE, DEFINE SUB-INDEX, DELETE, DELETE FILE, EXPUNGE, EXPUNGE SUB-INDEX, LINK SUB-INDEX, OPEN, READ, RETRIEVE, REWRITE, START, UNDELETE, UNLOCK, or WRITE statement and prior to the execution of any imperative statement associated with that input-output statement or prior to the execution of any applicable USE AFTER STANDARD EXCEPTION procedure. The value of the FILE STATUS is made available to the COBOL program through the use of the FILE STATUS clause in the file control entry for the file.

For ***VXCOBOL***, whenever the I-O status is updated the INFOS STATUS is also updated. INFOS STATUS is an extension to ANSI COBOL.

The I-O status also determines whether an applicable USE AFTER STANDARD EXCEPTION procedure will be executed. If any condition other than those contained under the heading "Successful Completion" below results, such a procedure may be executed depending on rules stated elsewhere. If one of the conditions listed under the heading "Successful Completion" below results, no such procedure will be executed. (See The USE Statement, page 479).

Certain classes of I-O status values indicate critical error conditions. These are:

any that begin with the digit 3 or 4, and
any that begin with the digit 9.

If the value of the I-O status for an input-output operation indicates such an error condition, and an applicable USE AFTER STANDARD EXCEPTION procedure exists, it is executed. After execution of the USE procedure, control returns to the statement following the statement that caused the error. If no applicable USE AFTER STANDARD EXCEPTION applies, after completion of the normal input-output control system error processing, and NO I-O status (FILE STATUS) or INFOS STATUS was associated with this file, the COBOL program is terminated with a Fatal Error indicating the type of error encountered and the COBOL pc. To prevent this from happening, a Declaratives section with an applicable USE procedure should be defined.

C.6.2 I-O Status (***ANSI 74***)

I-O status expresses one of the following conditions upon completion of the input-output operation:

(1) Successful Completion (0x). The input-output statement was successfully executed.

(2) At End (1x). A sequential READ statement was unsuccessfully executed as a result of an at end condition.

(3) Invalid Key (2x). The input-output statement was unsuccessfully executed as a result of an invalid key condition.

(4) Permanent Error (3x). The input-output statement was unsuccessfully executed as the result of an error that precluded further processing of the file. Any specified exception procedures are executed. The permanent error condition remains in effect for all subsequent input-output operations on the file unless an implementor-defined technique is invoked to correct the permanent error condition.

(5) **ICOBOL**-Defined (Implementor-Defined) (9x). The input-output statement was unsuccessfully executed as a result of a condition that is specified by **ICOBOL**.

The following is a list of the values placed in the I-O status for the previously named conditions resulting from the execution of an input-output operation on a file.

(1) Successful Completion

00 The input-output statement is successfully executed and no further information is available concerning the input-output operation.

02 The input-output statement is successfully executed but a duplicate key is detected. **SUPPORTED WHEN USING VERSION 7 INDEXED FILES WITH -G d OPTION TO ICRUN**

a. For a READ random or READ NEXT statement, the key value for the current key of reference is equal to the value of the same key in the next record within the current key of reference. For a READ PREVIOUS statement, the key value for the current key of reference is equal to the value of the same key in the previous record within the current key of reference.

b. For a REWRITE or WRITE statement, the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.

04 A READ statement is successfully executed but the length of the record being processed does not conform to the fixed file attributes for that file.

(2) At End Condition With Unsuccessful Completion

10 A sequential READ statement is attempted and no next logical record exists in the file because:

a. The end of the file has been reached, or

b. A sequential READ statement is attempted for the first time on an optional input file that is not present.

(3) Invalid Key Condition With Unsuccessful Completion

21 A sequence error exists for a sequentially accessed indexed file. The primary record key value has been changed by the program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file, or the ascending sequence requirements for successive record key values are violated.

22 The duplicates condition exists because:

a. An attempt is made to write a record that would create a duplicate key in a relative file, on the primary key, or on an alternate key that does not allow duplicates in an indexed file, or

b. An attempt is made to UNDELETE a record that was not deleted. **THIS IS AN EXTENSION TO ANSI COBOL.**

23 The no record found condition exists because:

a) An attempt is made to randomly access a record that does not exist in the file, or

b) A START or random READ statement is attempted on an optional input file that is not present.

24 An attempt is made to write beyond the externally defined boundaries of a relative or indexed file. Under **ICOBOL** this implies: for a relative file writing beyond the record number limit; and for an Indexed file the index structure is full.

(4) Permanent Error Condition With Unsuccessful Completion

30 A permanent error exists and no further information is available concerning the input-output operation. Generally related to some hardware condition.

34 A permanent error exists because of a boundary violation; an attempt is made to write beyond the externally defined boundaries of a sequential file. Generally out of disk space.

(5) **ICOBOL**-Defined (Implementor-Defined) Condition With Unsuccessful Completion.

91 An OPEN error. The possible violations are:

a. An OPEN statement referred to a file that was nonexistent.

b. An OPEN statement referred to a file that was already open. **This is a 41 with *ANSI 85*.**

c. An OPEN statement referred to a file that was had an illegal name.

d. A CLOSE statement referred to a file that had not been opened. **This is a 42 with *ANSI 85*.**

e. On OPEN, the filename already existed.

f. On OPEN, a nondirectory argument was in the pathname.

g. On OPEN, a zero-length filename was specified.

h. On OPEN, no more files could be opened from the operating system.

i. On OPEN, for devices the hardware is not present.

j. On a data-sensitive READ, the line is too long for the record. **This is a 34 with ANSI 85.**

92 An Access mode error. The possible violations are:

a. File not opened.

b. WRITE attempted to file opened for input. **This is a 48 with ANSI 85.**

c. DELETE attempted to file opened for input. **This is a 49 with ANSI 85.**

d. READ attempted for file opened for output. **This is a 47 with ANSI 85.**

e. OPEN attempted for file closed with lock. **This is a 38 with ANSI 85.**

f. DELETE or REWRITE statement not preceded by a READ statement for a file in sequential access mode. **This is a 43 with ANSI 85.**

g. OPEN attempted on a file with insufficient access rights for OPEN mode. **This is a 37 with ANSI 85.**

h. An attempt is made to WRITE or REWRITE a record that is larger than the largest or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated filename. **This is a 44 with ANSI 85.**

94 An In Use Error. The possible violations are:

a. File cannot be exclusively opened because it is in use.

b. Record cannot be accessed because it is locked.

c. DELETE FILE attempted for an opened file.

96 A directory named by the program does not exist.

97 Maximum number of open files exceeded.

98 Attempt to write more than 65,535 records to a relative file. **This is a 24 with ANSI 85.**

99 Printer control file is full.

9A File description inconsistency. Record length, key length, or key positions specified in program does not agree with the data file. **This is a 39 with ANSI 85.** ICISAM file version is not valid.

9B Corruption error. The possible violations are:

a. After a successful OPEN of an ISAM file, the runtime system has detected possible corruption in the file. Close this file; this sets the ISAM reliability flags and prevents further access to the file.

b. The data (.XD) portion of an Indexed or relative file is full. The ICISAM reliability flags are set.

c. On an attempted OPEN of an ICISAM file, the runtime has detected that the file is possibly corrupt although the ICISAM reliability flags are clear. The appropriate reliability flag(s) are set and the file is not opened.

9C Index (.NX) portion of an Indexed or relative file is full. The ICISAM reliability flags are not set.

9E Record lock limit has been exceeded.

9F Possible corruption of an ICISAM file has been detected on an attempted OPEN of the file; i.e., one or both of the ICISAM reliability flags had previously been set.

9T A time out condition has occurred on an I/O operation.

C.6.3 I-O Status (*ANSI 85*)

I-O status expresses one of the following conditions upon completion of the input-output operation:

(1) Successful Completion (0x). The input-output statement was successfully executed.

(2) At End (1x). A sequential READ statement was unsuccessfully executed as a result of an at end condition.

(3) Invalid Key (2x). The input-output statement was unsuccessfully executed as a result of an invalid key condition.

(4) Permanent Error (3x). The input-output statement was unsuccessfully executed as the result of an error that precluded further processing of the file. Any specified exception procedures are executed. The permanent error condition remains in effect for all subsequent input-output operations on the file unless an implementor-defined technique is invoked to correct the permanent error condition.

(5) Logic Error (4x). The input-output statement was unsuccessfully executed as a result of an improper sequence of input-output operations that were performed on the file or as a result of violating a limit defined by the user.

(6) **ICOBOL**-Defined (Implementor-Defined) (9x) Condition With Unsuccessful Completion. The input-output statement was unsuccessful executed as a result of a condition that is specified by **ICOBOL**.

The following is a list of the values placed in the I-O status for the previously named conditions resulting from the execution of an input-output operation on a file.

(1) Successful Completion

00 The input-output statement is successfully executed and no further information is available concerning the input-output operation.

02 The input-output statement is successfully executed but a duplicate key is detected. **SUPPORTED WHEN USING VERSION 7 INDEXED FILES.**

a) For a READ random or READ NEXT statement, the key value for the current key of reference is equal to the value of the same key in the next record within the current key of reference. For a READ PREVIOUS statement, the key value for the current key of reference is equal to the value of the same key in the previous record within the current key of reference.

b) For a REWRITE or WRITE statement, the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.

04 A READ statement is successfully executed but the length of the record being processed does not conform to the fixed file attributes for that file.

05 An OPEN statement is successfully executed but the referenced optional file is not present at the time the OPEN statement is executed. If the open mode is I-O or extend, the file has been created.

(2) At End Condition With Unsuccessful Completion

10 A sequential READ statement is attempted and no next logical record exists in the file because:

a. The end of the file has been reached, or

b. A sequential READ statement is attempted on an optional input file that is not present.

14 A sequential READ statement is attempted for a relative file and the number is larger than the size of the relative key data item described for the file. **NEVER GENERATED BY ICOBOL.**

(3) Invalid Key Condition With Unsuccessful Completion

21 A sequence error exists for a sequentially accessed indexed file. The primary record key value has been changed by the program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file, or the ascending sequence requirements for successive record key values are violated.

22 The duplicates condition exists because:

a. An attempt is made to write a record that would create a duplicate key in a relative file, on the primary key, or on an alternate key that does not allow duplicates in an indexed file.

b. An attempt is made to UNDELETE a record that was not deleted. **THIS IS AN EXTENSION TO ANSI COBOL.**

23 The no record found condition exists because:

a. An attempt is made to randomly access a record that does not exist in the file; or

b. A START or random READ statement is attempted on an optional input file that is not present.

24 An attempt is made to write beyond the externally defined boundaries of a relative or indexed file. Under **ICOBOL** this implies: for a relative file writing beyond the record number limit; and for an Indexed file the index structure is full.

(4) Permanent Error Condition With Unsuccessful Completion

30 A permanent error exists and no further information is available concerning the input-output operation. Generally related to some hardware condition.

34 A permanent error exists because of a boundary violation; an attempt is made to write beyond the externally defined boundaries of a sequential file. Generally out of disk space. On a DATA-SENSITIVE READ the line is too long for the record.

35 A permanent error exists because an OPEN statement with the INPUT, I-O, or EXTEND phrase is attempted on a non-optional file that is not present.

37 A permanent error exists because an OPEN statement is attempted on a file and that file will not support the open mode specified in the OPEN statement.

The possible violations are:

a. The EXTEND or OUTPUT phrase is specified but the file will not support write operations.

b. The I-O phrase is specified but the file will not support the input and output operations that are permitted for a sequential file when opened in the I-O mode.

c. The INPUT phrase is specified but the file will not support read operations.

38 A permanent error exists because an OPEN statement is attempted on a file previously closed with lock.

39 The OPEN statement is unsuccessful because a conflict has been detected between the fixed file attributes and the attributes specified for that file in the program.

(5) Logic Error Condition With Unsuccessful Completion.

41 An OPEN statement is attempted for a file in the open mode.

42 A CLOSE statement is attempted for a file not in open mode.

43 For a mass storage file in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of a REWRITE statement was not a successfully executed READ statement.

44 A boundary violation exists because:

a. An attempt is made to write or rewrite a record that is larger than the largest or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated file-name, or

b. An attempt is made to rewrite a record to a sequential, relative, or indexed file and the record is not the same size as the record being replaced.

46 A sequential READ statement is attempted on a file open in the input or I-O mode and no valid next record has been established because:

a. The preceding READ statement was unsuccessful but did not cause an at end condition, or

b. The preceding READ statement caused an at end condition.

c. The preceding START statement was unsuccessful.

47 The execution of a READ or START statement is attempted on a file not open in the input or I-O mode.

48 The execution of a WRITE statement is attempted on a file not open in the I-O, output or extend mode.

49 The execution of a DELETE, REWRITE, or UNDELETE statement is attempted file not open in the I-O mode.

(6) **ICOBOL**-Defined (Implementor-Defined) Condition With Unsuccessful Completion.

91 An OPEN error. The possible violations are:

a. An OPEN statement referred to a file that was nonexistent.

b. An OPEN statement referred to a file that was had an illegal name.

c. On OPEN, the filename already existed.

d. On OPEN, a nondirectory argument was in the pathname.

e. On OPEN, a zero-length filename was specified.

f. On OPEN, no more files could be opened from the operating system.

g. On OPEN, for devices the hardware is not present.

92 An Access mode error. The possible violations are:

a. File not opened.

94 An In Use Error. The possible violations are:

a. File cannot be exclusively opened because it is in use.

b. Record cannot be accessed because it is locked.

c. DELETE FILE attempted for an opened file.

96 A directory named by the program does not exist.

97 Maximum number of open files exceeded.

99 Printer control file is full.

9A ICISAM file version is not valid.

9B Corruption error. The possible violations are:

a. After a successful OPEN of an ICISAM file, the runtime system has detected possible corruption in the file. Close this file; this sets the ICISAM reliability flags and prevents further access to the file.

b. The data (.XD) portion of an Indexed or relative file is full. The ICISAM reliability flags are set.

c. On an attempted OPEN of an ICISAM file, the runtime has detected that the file is possibly corrupt although the ICISAM reliability flags are clear. The appropriate reliability flag(s) are set and the file is not opened.

9C Index (.NX) portion of an Indexed or relative file is full. The ICISAM reliability flags are not set.

9E Record lock limit has been exceeded.

9F Possible corruption of an ICISAM file has been detected on an attempted OPEN of the file; i.e., one or both of the ICISAM reliability flags had previously been set.

9T A time out condition has occurred on an I/O operation.

C.6.4 I-O Status (***VXCOBOL***)

I-O status expresses one of the following conditions upon completion of the input-output operation:

(1) Successful Completion (0x). The input-output statement was successfully executed.

(2) At End (1x). A sequential READ statement was unsuccessfully executed as a result of an at end condition.

(3) Invalid Key (2x). The input-output statement was unsuccessfully executed as a result of an invalid key condition.

(4) Permanent Error (3x). The input-output statement was unsuccessfully executed as the result of an error that precluded further processing of the file. Any specified exception procedures are executed. The permanent error condition remains in effect for all subsequent input-output operations on the file unless an implementor-defined technique is invoked to correct the permanent error condition.

(5) **ICOBOL**-Defined (Implementor-Defined) (9x). The input-output statement was unsuccessfully executed as a result of a condition that is specified by **ICOBOL**.

The following is a list of the values placed in the I-O status for the previously named conditions resulting from the execution of an input-output operation on a file.

(1) Successful Completion

00 The input-output statement is successfully executed and no further information is available concerning the input-output operation.

02 The input-output statement is successfully executed but a duplicate key is detected.

a. For a READ random or READ NEXT statement, the key value for the current key of reference is equal to the value of the same key in the next record within the current key of reference. For a READ BACKWARD statement, the key value for the current key of reference is equal to the value of the same key in the previous record within the current key of reference.

b. For a REWRITE or WRITE statement, the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.

(2) At End Condition With Unsuccessful Completion

10 A sequential READ statement is attempted and no next logical record exists in the file because:

a. The end of the file has been reached,

b. The end of a subindex has been reached, or

c. A sequential READ statement is attempted for the first time on an optional input file that is not present.

(3) Invalid Key Condition With Unsuccessful Completion

21 A sequence error exists for a sequentially accessed indexed file. The primary record key value has been changed by the program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file, or the ascending sequence requirements for successive record key values are violated.

22 The duplicates condition exists because:

a. An attempt is made to write a record that would create a duplicate key in a relative file, on the primary key, or on an alternate key that does not allow duplicates in an indexed file, or any key which does not allow duplicates in an INFOS file.

b. For an INFOS file, an attempt has been made to write a record or partial record which already exists.

c. For an INFOS file, an attempt to write a duplicate key in a subindex which does not allow duplicate keys.

23 The no record found condition exists because:

a. An attempt is made to randomly access a key, data record, or partial record that does not exist in the file;

b. A START or random READ statement is attempted on an optional input file that is not present.

c. Relative key is too large.

d. For relative and indexed files, no valid current record pointer has been established.

e. A subindex referenced in an INFOS key path does not exist.

f. The total length of an INFOS key path is too long or is a single null byte.

g. Attempt to UNDELETE a record which is not logically deleted.

24 An attempt is made to write beyond the externally defined boundaries of a relative or indexed file. Under **ICOBOL**, this implies that the index structure is full.

(4) Permanent Error Condition With Unsuccessful Completion

30 A permanent error exists and no further information is available concerning the input-output operation. Generally related to some hardware condition or any condition for which there is no logical I-O status. (more specific information is found in EXCEPTION Status.)

34 A permanent error exists because of a boundary violation; an attempt is made to write beyond the externally defined boundaries of a file. Generally out of disk space.

(5) **ICOBOL**-Defined (Implementor-Defined) Condition With Unsuccessful Completion.

91 An OPEN error. The possible violations are:

a. An OPEN statement referred to a file that was nonexistent.

b. An OPEN statement referred to a file that was already open.

c. An OPEN statement referred to a file that was had an illegal name.

d. On OPEN, the filename already existed.

e. On OPEN, a nondirectory argument was in the pathname.

f. On OPEN, a zero-length filename was specified.

g. On OPEN, no more files could be opened from the operating system.

h. On OPEN, for devices the hardware is not present.

i. On OPEN, access to the file or device is denied.

j. Any consistency errors on open of an INFOS file.

92 An Access mode error. The possible violations are:

a. An I/O operation referred to a file that was not opened.

b. WRITE attempted to file opened for input.

c. DELETE attempted to file opened for input.

d. READ attempted for file opened for output.

e. OPEN attempted for file closed with lock.

f. OPEN attempted on a file with insufficient access rights for OPEN mode.

94 An In Use Error. The possible violations are:

a. File cannot be exclusively opened because it is in use.

b. Record cannot be accessed because it is locked.

c. DELETE FILE attempted for an opened file.

96 The record the program is trying to access has been previously marked as logically deleted either globally or locally.

97 REWRITE or DELETE attempted without executing previous READ statement for an indexed file with sequential access.

99 On a data-sensitive READ, the line is too long for the record, or for INFOS, an INFOS error has occurred for which there is no corresponding file status code. See INFOS Status for more information.

9A File description inconsistency. Record length, key length, or key positions specified in program does not agree with the data file. ICISAM file version is not valid.

9B Corruption error. The possible violations are:

a. After a successful OPEN of an ICISAM file, the runtime system has detected possible corruption in the file. Close this file; this sets the ICISAM reliability flags and prevents further access to the file.

b. The data (.XD) portion of an ICISAM file is full. The ICISAM reliability flags are set.

c. On an attempted OPEN of an ICISAM file, the runtime has detected that the file is possibly corrupt although the ICISAM reliability flags are clear. The appropriate reliability flag(s) are set and the file is not opened.

9C Index (.NX) portion of an ICISAM file is full. The ICISAM reliability flags are not set.

9E Record lock limit has been exceeded.

9F Possible corruption of an ICISAM file has been detected on an attempted OPEN of the file; i.e., one or both of the ICISAM reliability flags had previously been set.

9T A time out condition has occurred on an I/O operation.

C.6.5 INFOS Status (*VXCOBOL*)

The INFOS STATUS data item receives a value representing an exception code that INFOS II, U/FOS, or the operating system returns during an input-output operation. Whenever the I-O status (FILE Status) is updated, INFOS STATUS is also updated. INFOS STATUS is an 11-character item taking one of two forms:

(1) A string representing an octal AOS/VS error message code. For example, "00000007030" represents the octal AOS/VS error code 7030, "Keyed positioning error".

(2) A string beginning with the letter `X' and representing a decimal **ICOBOL** exception status. For example, "X0000000073" corresponds to exception status 73, "Reliability flag indicates the .NX file may be corrupt.".

In the first form, an AOS/VS-compatible error code is returned even on UNIX or Windows systems.

On a successful input-output operation INFOS STATUS will be set to zero, i.e. "00000000000".

C.6.6 The At End Condition

The at end condition can occur as a result of the execution of a READ or RETRIEVE statement.

C.6.7 The Invalid Key Condition

The invalid key condition can occur as a result of the execution of a DEFINE SUB-INDEX, DELETE, EXPUNGE SUB-INDEX, LINK SUB-INDEX, READ, RETRIEVE, REWRITE, START, UNDELETE, or WRITE statement. When the invalid key condition occurs, execution of the input-output statement which recognized the condition is unsuccessful and the file is not affected.

If the invalid key condition exists after the execution of the input-output operation specified in an input-output statement, the following actions occur in the order shown:

(1) The I-O status of the file connector associated with the statement is set to a value indicating the invalid key condition.

(2) If the INVALID KEY phrase is specified in the input-output statement, any USE AFTER STANDARD EXCEPTION procedure associated with the file connector is not executed and control is transferred to the imperative-statement specified in the INVALID KEY phrase. Execution then continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the INVALID KEY phrase, control is transferred to the end of the input-output statement and the NOT INVALID KEY phrase is ignored, if specified.

(3) If the INVALID KEY phrase is not specified in the input-output statement, a USE AFTER STANDARD EXCEPTION procedure must be associated with the file connector and that procedure is executed and control is transferred according to the rules of the USE statement. The NOT INVALID KEY phrase is ignored, if specified.

(4) For ***VXCOBOL***, if neither the INVALID KEY phrase nor a USE procedure is applicable, then control proceeds to the end of the input-output statement if either INFOS STATUS or FILE STATUS is specified. Otherwise the program is aborted.

If the invalid key condition does not exist after the execution of the input-output operation specified by an input-output statement, the INVALID KEY phrase is ignored, if specified. The I-O status of the file connector associated with the statement is updated and the following actions occur:

(1) If an exception condition which is not an invalid key condition exists, control is transferred according to the rules of the USE statement following the execution of any USE AFTER STANDARD EXCEPTION procedure associated with the file connector. (See The USE Statement, page 479.) For ***VXCOBOL***, if there is no applicable USE statement and either INFOS STATUS or FILE STATUS has been specified, control passes to the end of the input-output statement. Otherwise, for all dialects, the program is aborted.

(2) If no exception condition exists, control is transferred to the end of the input-output statement or to the imperative-statement specified in the NOT INVALID KEY phrase, if it is specified. In the latter case, execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement in the NOT INVALID KEY phrase, control is transferred to the end of the input-output statement.

C.6.8 The File Attribute Conflict Condition

The file attribute conflict condition can result from the execution of an OPEN, REWRITE, or WRITE statement. When the file attribute conflict condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected. (See The OPEN Statement, page 399; The REWRITE Statement, page 433; and The WRITE Statement, page 483.)

When the file attribute conflict condition is recognized, these actions take place in the following order:

(1) A value is placed in the I-O status associated with the file-name to indicate the file attribute conflict condition.

(2) A USE AFTER STANDARD EXCEPTION procedure, if any, associated with the file-name is executed.

C.6.9 Exception Declaratives

A USE AFTER STANDARD EXCEPTION procedure, when one is specified for the file, is executed whenever an input or output condition arises which results in an unsuccessful input-output operation. However, the exception declarative is not executed if the condition is invalid key and the INVALID KEY phrase is specified, or if the condition is at end and the AT END phrase is specified.

C.6.10 Optional Phrases

The INVALID KEY and NOT INVALID KEY phrases may be associated with the DEFINE SUB-INDEX, DELETE, EXPUNGE SUB-INDEX, LINK SUB-INDEX, READ, RETRIEVE, REWRITE, START, UNDELETE, or WRITE statements.

Some of the conditions that give rise to an invalid key condition are:

(1) A requested key does not exist in the file (DELETE, READ, START, or UNDELETE statements),

(2) A key is already in a file and duplicates are not allowed (WRITE statement),

(3) A key does not exist in the file, or

(4) A key was not the last key read (REWRITE statement).

If the invalid key condition occurs during the execution of a statement for which the INVALID KEY phrase has been specified, the statement identified by that INVALID KEY phrase is executed.

The AT END and NOT AT END phrase may be associated with a READ statement. The at end condition occurs in a sequentially accessed file when no next logical record exists in the file, when the number of significant digits in the relative record number is larger than the size of the relative key data item, when an optional file is not present, or when a READ statement is attempted and the at end condition already exists. If the at end condition occurs during the execution of a statement for which the AT END phrase has been specified, the statement identified by that AT END phrase is executed.

C.6.11 ACCEPT FROM EXCEPTION STATUS

The exception status is a very specific error number that allows much better reporting of errors than I-O status (FILE STATUS) values. An exception status is not specific to I-O. More on exception status can be found in the ACCEPT FROM EXCEPTION STATUS statement discussion starting on page 290. ACCEPT FROM EXCEPTION STATUS is an extension to ANSI COBOL.

C.7. Shared Record Area

This feature saves memory space in the object program, as it allows more than one file to share the same file area and input-output areas.

When the RECORD option of the SAME clause is used, only the record area is shared and the input-output areas for each file remain independent. In this case, any number of the files sharing the same record area may be active at one time. This can increase the execution speed of the object program.

To illustrate this point, consider file maintenance. If the programmer assigns the same record area to both the old and new files, he not only saves memory in the object program, but because this technique eliminates a move of each record from the input to the output area, significant time savings result. An additional benefit of this technique is that the programmer need not define the record in detail as a part of both the old and new files. Rather, he defines the record completely in one case and simply includes the level 01 entry in the other. Because these record areas are in fact the same area, one set of names suffices for all processing requirements without requiring qualification.

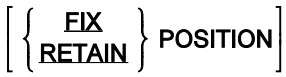
C.8. INFOS File I-O Common Phrases (***VXCOBOL***)

Many of the INFOS input-output statements share a set of common phrases that direct the operation of the statement. In particular, they direct positioning of the record pointer, motion through the index structure, and the manner in which keys are used.

C.8.1 The POSITION Phrase (***VXCOBOL***)

The POSITION phrase allows for control of positioning within an INFOS file. The current position is a marker in an INFOS file which establishes a reference point for relative motion within the file.

The format of the position phrase is:



If FIX POSITION is specified, the file's current position is set to the key accessed by the statement if the operation was successful. The file's current position remains unchanged if RETAIN POSITION is specified.

Each input-output statement has a default positioning behavior. This behavior can be overridden with the POSITION phrase.

RETAIN POSITION is the default for the DEFINE SUB-INDEX, EXPUNGE SUB-INDEX, LINK SUB-INDEX, RETRIEVE SUB-INDEX, RETRIEVE STATUS, REWRITE, UNDELETE, and WRITE statements. The current position remains unchanged from its last position.

FIX POSITION is the default for READ, RETRIEVE HIGH KEY, and RETRIEVE KEY. The current position is set to the key it last accessed.

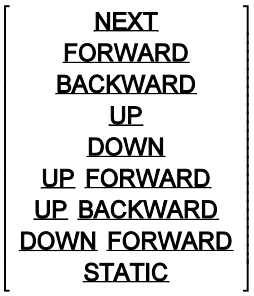
The default positioning for the START and DELETE statements cannot be overridden. START sets the current position to the key it last accessed. DELETE sets the current position to the key prior to the one just deleted, possibly in front of a subindex if it was the first key.

OPEN sets the current position in front of the main index for files in SEQUENTIAL or DYNAMIC access modes.

C.8.2 The Relative Motion Phrase (***VXCOBOL***)

The relative motion phrase is used to control motion within an INFOS file. With relative motion, the key being sought in the INFOS file is in a position relative to the current position.

The format of the relative motion phrase is:



NEXT and FORWARD are equivalent. They imply movement to the next higher key in the index relative to the current position. If there is no next higher key, an "end of subindex" error (I-O status 10, INFOS STATUS 7011) occurs.

BACKWARD implies movement to the next lower key in the index relative to the current position. If there is no next lower key, an "end of subindex" error (I-O status 10, INFOS STATUS 7011) occurs.

UP implies movement to the key entry in the immediately higher index level relative to the current position. If the current position is in the top level index (main index), a "positioned above main index" error (I-O status 99, INFOS STATUS 7006) will occur with upward motion.

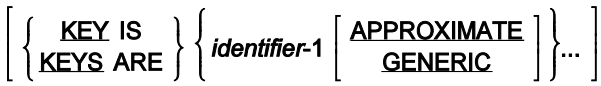
DOWN implies movement to a position prior to the first key in the subindex defined for the current key. If the current key does not have an associated subindex, a "subindex not defined" error (I-O status 99, INFOS STATUS 7010) occurs.

UP FORWARD, UP BACKWARD, and DOWN FORWARD combine processing between index levels with movement to keys in the index. UP FORWARD and UP BACKWARD imply movement to the next higher level and movement to the next higher or lower key respectively. DOWN FORWARD implies movement to the first key in the subindex defined for the current key.

STATIC means no movement relative to the current position.

C.8.3 The KEY Series Phrase (***VXCOBOL***)

The key series phrase is used to specify a specific key in an INFOS file. The format of the key series phrase is:



where identifier is a RECORD KEY named in the SELECT statement for the INFOS file.

For a single level file, at most one key may be specified. If the key series phrase is present then no relative motion phrase (NEXT, FORWARD, BACKWARD, or STATIC) may be specified on the input-output statement.

For a multilevel file, the maximum number of keys that may be specified in the key series phrase on an input-output statement is equal to the number of levels in the file. If no relative motion phrase is specified on the statement, each key identifies an index entry at increasingly lower levels, i.e. the first key identifies the entry at the top level, the second key indicates an entry in the subindex defined for the top key, etc. If a relative motion phrase is specified on the input-output statement, the relative motion is performed first and the key series phrase identifies a path beginning at the key determined by the relative motion.

Each key specified in the key series phrase may be modified with the GENERIC or APPROXIMATE clauses. **ICOBOL** searches for keys in the following manner:

(1) Without either clause, the key value sought is the value contained in the identifier up to the length of the identifier or an optionally specified KEY LENGTH. The match must be exact in both content and length.

(2) If the GENERIC clause is specified, the first key in the current index or subindex that matches the key up to the length specified will be a match. The key located may be longer than the key that was specified. This allows for matching based only on the first few characters of a value.

(3) If the APPROXIMATE clause is specified, the first key in the current index or subindex that is greater than or equal to the value specified, within the length specified, will be a match.

C.8.4 The SUPPRESS Phrase (***VXCOBOL***)

ICOBOL allows for suppressing the input or output of a data record or partial record. The SUPPRESS phrase has the following format:

[SUPPRESS [PARTIAL RECORD] [DATA RECORD]]

If PARTIAL RECORD is specified, the contents of the partial record for the key is neither read nor written. For example, a WRITE statement with a SUPPRESS PARTIAL RECORD will write only the data record.

If DATA RECORD is specified, the contents of the data record for the key is neither read nor written. For example, a READ statement with a SUPPRESS DATA RECORD will retrieve only the data in the partial record.

Both clauses may be specified together. If both clauses are specified on a READ, the result is to change the current position without retrieving any data. If both clauses are specified on a WRITE, only a key is written.

SUPPRESS alone is equivalent to specifying both phrases.

C.8.5 The LOCK/UNLOCK Phrase (*VXCOBOL*)

Many input-output statements for INFOS files support the LOCK/UNLOCK phrase. The format of the phrase is:

```
[ LOCK   ]
[ UNLOCK ]
```

Record locks are a binary condition. A record is either locked or it is not locked. Data records and partial records can be locked and unlocked independently.

If the LOCK phrase is specified on an operation, the record is locked and no other user can access the record until it is unlocked. If the UNLOCK phrase is specified, the record is unlocked and becomes accessible to any user. (Locks typically occur at the beginning of an operation and unlocks at the end.)

Data record locks are not regarded if SUPPRESS DATA RECORD is specified on the input-output statement. Partial record locks are not regarded if SUPPRESS PARTIAL RECORD is specified.

All records in a file that have been locked can be unlocked at once with the UNLOCK statement or by closing the file.

D. Header

The Procedure Division is identified by, and must begin with, the following header:

PROCEDURE DIVISION [USING { *data-name-1* }...] .

The USING phrase is necessary only if the object program is to be invoked by a CALL statement or a CALL PROGRAM statement, and that statement includes a USING phrase.

The USING phrase of the Procedure Division header identifies the names used by the program for any parameters passed to it by a calling program. The parameters passed to a called program are identified in the USING phrase of the calling program's CALL statement. The correspondence between the two lists of names is established on a positional basis.

Data-name-1 must be defined as a level 01 entry or a level 77 entry in the Linkage Section. A particular user-defined word may not appear more than once as *data-name-1*. The data description entry for *data-name-1* must not contain a REDEFINES clause. *Data-name-1* may, however, be the object of a REDEFINES clause elsewhere in the Linkage Section.

The following additional rules apply:

(1) If the reference to the corresponding data item in the CALL statement declares the parameter to be passed by content, the value of the item is moved when the CALL statement is executed and placed into a system-defined storage item possessing the attributes declared in the Linkage Section for *data-name-1*. The data description of each parameter in the BY CONTENT phrase of the CALL statement must be the same, meaning no conversion or extension or truncation, as the data description of the corresponding parameter in the USINAG phrase of the Procedure Division header.

(2) If the reference to the corresponding data item in the CALL statement declares the parameter to be passed by reference, the object program operates as if the data item in the called program occupies the same storage area as the data item in the calling program. The description of the data item in the called program must describe the same number of character positions as described by the description of the corresponding data item in the calling program.

(3) At all times in the called program, references to *data-name-1* are resolved in accordance with the description of the item given in the Linkage Section of the called program.

(4) Data items defined in the Linkage Section of the called program may be referenced within the Procedure Division of that program if, and only if, they satisfy one of the following conditions:

a. They are operands of the USING phrase of the Procedure Division header.

b. They are subordinate to operands of the USING phrase of the Procedure Division header.

c. They are defined with a REDEFINES or RENAMES clause, the object of which satisfies the above conditions.

d. They are items subordinate to any item which satisfies the condition in rule 4c.

e. They are condition-names or index-names associated with data items that satisfy any of the above four conditions.

E. Statements

E.1. ACCEPT (keyboard)

E.1.1 Function

The ACCEPT statement causes data from the keyboard to be made available to data items in the File, Working-Storage, or Linkage sections.

Screens are an extension to ANSI COBOL. The TIME-OUT clause is an extension to ANSI COBOL.

E.1.2 General Format (***ANSI 74*** and ***ANSI 85***)

Format 1:

```
ACCEPT identifier-1 [ FROM mnemonic-name ] [ TIME-OUT AFTER { identifier-4 } ]
                                                            { literal-3    }
    [ ON ESCAPE imperative-statement-1 ]
    [ NOT ON ESCAPE imperative-statement-2 ]
    [ END-ACCEPT ]
```

Format 2:

```
ACCEPT screen-name [ AT { LINE { identifier-2 } [ { COLUMN } { identifier-3 } ] }
                        { literal-1            }   { COL    } { literal-2    }   }
                       { { COLUMN } { identifier-3 } [ LINE { identifier-2 } ] }
                       { { COL    } { literal-2    }   { literal-1 }   ]   }
    [ TIME-OUT AFTER { identifier-4 } ]
                     { literal-3    }
    [ ON ESCAPE imperative-statement-1 ]
    [ NOT ON ESCAPE imperative-statement-2 ]
    [ END-ACCEPT ]
```

Format 3:

```
ACCEPT { identifier-1 [ UNIT { identifier-5 } ] [ { accept-clause } ]... }...
                             { literal-4    }
    [ ON { ESCAPE    } [ identifier-14 ] { imperative-statement-1 } ]
         { EXCEPTION }                   { NEXT SENTENCE         }
      [ NOT ON { ESCAPE    } imperative-sentence-2 ]
               { EXCEPTION }
    [ END-ACCEPT ]
```

where *accept-clause* is one of the following:

```
{ BACKGROUND-COLOR } IS { identifier-6 }
{ BACKGROUND       }   { literal-5     }
                       { color-name-1  }
```

```
{ FOREGROUND-COLOR } IS { identifier-10 }
{ FOREGROUND       }    { literal-9    }
                        { color-name-2 }

NO { BELL }
   { BEEP }

BLINK

{ COLUMN }
{ COL    } { identifier-7 }
{ POSITION } { literal-6  }

CONTROL { identifier-8 }
        { literal-7    }

CONVERT

CONVERTING { UP   }
           { DOWN }

CURSOR { identifier-9 }
       { literal-8    }

ECHO

ERASE [ EOL    ]
      [ EOS    ]
      [ LINE   ]
      [ SCREEN ]

{ HIGH      }
{ HIGHLIGHT }
{ LOW       }
{ LOWLIGHT  }
{ BOLD      }
{ BRIGHT    }
{ DIM       }

LINE { identifier-11 }
     { literal-10    }

PROMPT [ literal-11 ]

{ REVERSE       }
{ REVERSED      }
{ REVERSE-VIDEO }

{ SECURE [ { WITH ECHO } ] }
{        [ { NO ECHO   } ] }
{                          }
{ OFF                      }

SIZE { identifier-12 }
     { literal-12    }
```

TAB

BEFORE TIME { *identifier*-13 / *literal*-13 }

TIME-OUT AFTER { *identifier*-4 / *literal*-3 }

{ UNDERLINE / UNDERLINED }

UPDATE

E.1.3 General Format (***VXCOBOL***)

Format 1:

ACCEPT *identifier-1* [FROM *mnemonic-name*] [TIME-OUT AFTER { *identifier*-4 / *literal*-3 } SECONDS]
[ON ESCAPE *imperative-statement-1*]
[NOT ON ESCAPE *imperative-statement-2*]
[END-ACCEPT]

Format 2:

ACCEPT *screen-name* [AT { LINE { *identifier*-2 / *literal*-1 } [{ COLUMN / COL } { *identifier*-3 / *literal*-2 }] / { COLUMN / COL } { *identifier*-3 / *literal*-2 } [LINE { *identifier*-2 / *literal*-1 }] }]
[TIME-OUT AFTER { *identifier*-4 / *literal*-3 } SECONDS]
[ON ESCAPE *imperative-statement-1*]
[NOT ON ESCAPE *imperative-statement-2*]
[END-ACCEPT]

E.1.4 Syntax Rules

(1) *Screen-name* may not be subscripted.

(2) If *screen-name* is a group format item, it must have at least one input, input-output, or update screen-data item; otherwise, it must specify an input, input-output, or update screen-data item.

(3) In Format 1 and 3, *identifier-1* cannot be larger than the 132 characters for **ANSI 74** and **ANSI 85** and 2048 for ***VXCOBOL***.

(4) In Format 2, *identifier-2*, *identifier-3*, *literal-1*, and *literal-2* must be elementary integer items.

(5) In Format 3, *identifier-5*, *identifier-6*, *identifier-7*, *identifier-9*, *identifier-10*, *identifier-11*, *identifier-12*, *identifier-13*, *identifier-14*, *literal-4*, *literal-5*, *literal-6*, *literal-8*, *literal-9*, *literal-10*, *literal-12*, literal-13 must be unsigned elementary integer items. *Identifier-8* must be a nonnumeric data-item and *literal-7* must be a nonnumeric literal. *Literal-6* must be a nonnumeric literal exactly one character in length.

(6) *Identifier-4* and *literal-3* may represent any numeric literal or elementary numeric data-item.

(7) *Color-name-1* and *color-name-2* represent one of the predefined color names: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, or WHITE.

(8) The word COL is an abbreviation for the word COLUMN.

(9) In Format 1, *Mnemonic-name* must be specified in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION and must be associated with a hardware device.

(10) In Format 1, the FROM clause is for documentation purposes only.

(11) In Format 3, the word EXCEPTION is a synonym for ESCAPE, the word POSITION is a synonym for COLUMN, and the word BEEP is a synonym for BELL.

E.1.5 General Rules

Format 1:

(1) The ACCEPT statement causes the transfer of data from the keyboard. This data replaces the contents of the data item referenced by *identifier-1* according to the rules for the MOVE statement.

(2) Data input for *identifier-1* must be valid for the identifier. For **ANSI 74 and ANSI 85**, if the characters that are input do not agree with the item's PICTURE, then an error message is displayed on the last line of the display screen, and the input must be corrected. For example, alphabetic characters entered into a numeric item will be rejected. For complete details of PICTURE definitions and acceptable input, see the PICTURE Clause discussion in the WORKING-STORAGE section. For ***VXCOBOL***, the data will be converted and assigned to *identifier-1* as closely as is possible if it is not valid for the identifier.

(3) If a *field terminator key* (i.e., any key configured in the terminal description to generate a value of 00) is pressed at any time during an ACCEPT statement, the data is validated and transferred to the data item referenced by *identifier-1*. The ON ESCAPE clause, if present, is bypassed and the NOT ON ESCAPE phrase, if specified, is executed; otherwise, control is transferred to the end of the ACCEPT statement.

(4) If the *ESC function key* (i.e., any key configured in the terminal description to generate a value of 01) is pressed at any time during an ACCEPT statement, the data from the keyboard is discarded, and the data item referenced by *identifier-1* is not changed. The ACCEPT terminates, and the ON ESCAPE clause, if present, is executed.

(5) If a *normal function key* (i.e., any key configured in the terminal description to generate a value that is not 00 or 01) is pressed at any time during an ACCEPT statement, the data is validated and transferred to the data item referenced by *identifier-1*. The ACCEPT terminates, and the ON ESCAPE clause, if present, is executed.

(6) Each keyboard sequence is interpreted as defined by the current ICTERM entry. If you wish to read binary data from the terminal, you should open a file whose SELECT clause contains an ASSIGN TO DISK "@CON" and perform a READ to get non-interpreted binary data with no positioning codes sent to the terminal.

(7) An ACCEPT statement should not be executed while in Print Pass Through mode on a terminal, as the ACCEPT will generate some output that will then be printed.

(8) For **ANSI 74 and ANSI 85**, A non-screen ACCEPT is limited to a single line and is truncated at the column width of the terminal. After entering the data, a <nl><nl><up-arrow> sequence is generated to position to the first column on the next line. For ***VXCOBOL***, up to 2048 characters are read with echoing and backspace processing starting at the current cursor position. An ESC or function key will exit with an ON ESCAPE processing, but no echoing of the ESC or function key. A NL or CR will echo as a newline with no ON ESCAPE processing.

Format 2:

(9) A screen ACCEPT that extends past a terminal's width is supported by allowing the ACCEPT to wrap to the next line since the screen does not have to scroll; i.e., the wrap would otherwise move to the line after the last line on the screen.

(10) ACCEPT *screen-name* transfers information entered on the screen via the keyboard to the data items associated with *screen-name*. The program should have executed a DISPLAY *screen-name* before the ACCEPT to display any associated prompts.

(11) ACCEPT *screen-name* without the LINE or COLUMN phrases is equivalent to ACCEPT *screen-name* AT LINE 0 COLUMN 0.

(12) If *screen-name* refers to a screen-group item, the ACCEPT statement processes all input, input-output, and update screen-data items subordinate to *screen-name*. The fields are processed in the order in which they appear in the source program.

(13) The LINE phrase and COLUMN phrase in DISPLAY and ACCEPT statements allow the entire screen description referenced by *screen-name* to be moved to a different starting position on the user's display device. This capability is called *variable origin*. All screen descriptions assume that the origin is at line 1 and column 1 on the user's display device. The value specified in the DISPLAY or ACCEPT's LINE phrase, if present, is treated as a relative offset to be added to all line positions in the screen. Similarly, the value of the COLUMN phrase, if specified, is treated as a relative offset to be added to all column positions in the screen. If any line or column position becomes larger than what is supported by the current screen, the screen will wrap at its limits, and the new (wrapped) values will in turn be offset again by the variable origin.

For example, consider the code fragments:
The following discussion describes how to determine the *origin point* for each of the two DISPLAY and ACCEPT pairs in the code fragments above. Assume the display device has 24 lines and 80 columns.

```
01 ANY-CHANGE-SCREEN.
    05 LINE 23 COL 60          "ANY CHANGE?".
    05 LINE 23 COL 75          PIC X TO ANY-CHANGE-ANSWER.

ANY-CHANGE-1.
    DISPLAY ANY-CHANGE-SCREEN.
    ACCEPT ANY-CHANGE-SCREEN.

ANY-CHANGE-2.
            DISPLAY ANY-CHANGE-SCREEN AT LINE 5 COLUMN 30.
            ACCEPT ANY-CHANGE-SCREEN AT LINE 5 COLUMN 30.
```

a. Remember, all screen descriptions assume an *origin point* of line 1, column 1. This screen has a positioning definition of line 23, column 60, and the first screen DISPLAY statement contains no positioning (line or column) clauses. Therefore, the *origin point* for the first DISPLAY is line 23, column 60.

b. For the second screen DISPLAY statement, which contains the positioning clauses AT LINE 5 COLUMN 30, the offset position will be line 28, column 90. (We added the line and column variable-positioning values in the DISPLAY statement to the *origin point* established in the previous step.)

c. Now, because the line and column numbers are larger than the size of the display device, we subtract the line and column size of the display device, to find the wrap values: line 4, column 10. This becomes the *new origin point*.

d. Finally, add the line and column positioning values which in turn will be offset to line 9, column 40. Therefore, the second screen DISPLAY will begin at line 9, column 40.

e. Determining the *origin point* for the ACCEPT field is similar. The table below illustrates how the *origin points* are calculated for the second ACCEPT and DISPLAY.

literal field		input field		Description
LINE	COLUMN	LINE	COLUMN	
23	60	23	75	Origin point in screen definition
5	30	5	30	ADD offset from DISPLAY/ACCEPT
28	90	28	105	Giving offset position
24	80	24	80	SUBTRACT display device size
4	10	4	25	Giving new origin point
5	30	5	30	ADD offset from DISPLAY/ACCEPT
9	40	9	55	Giving origin point for 2nd DISPLAY/ACCEPT

TABLE 18. Variable Origin for DISPLAY and ACCEPT

(14) If variable origin is used for an ACCEPT operation on a *screen-name*, the same variable origin specification should be used for the corresponding DISPLAY statement of the *screen-name* in order to have the correct visual association between prompts and data-entry items..

(15) The basic operation of the ACCEPT statement is described by the following steps. The discussion assumes that *screen-name* represents a group item in the screen description that has several subordinate input, input-output, and/or update fields. The case where *screen-name* specifies a single screen-data item is simply a subset of the description below.

a. The screen management system positions to the first (in terms of its position in the source definition of *screen-name*) input, input-output, or update field that is subordinate to screen-name.

b. The content of the screen field (which has either been set by a previous execution of a DISPLAY statement for the field, or which remains from a previous execution of an ACCEPT statement for the field) is redisplayed on the screen with the specified attributes. If the field is a numeric-edited picture, the field is first de-edited by removing all the editing characters (i.e., all but the plus or minus sign, the decimal point, and the numeric digits).

For ***ANSI 74*** and ***ANSI 85***, if the field has the SECURE ECHO attribute, the field is redisplayed as all asterisk (*) characters; if the field has the SECURE NO ECHO attribute, nothing is displayed.

For ***VXCOBOL***, if the field has the SECURE attribute, nothing is displayed.

c. The cursor is positioned to the first character of the field, and the screen control system waits for the user to enter data into the field. The user may enter new data characters, field editing keys, or field termination keys. The screen management system echoes input characters and positions the cursor appropriately in response to the user's input. The field is terminated by entering an appropriate field termination key (see the ESCAPE KEY table above) or, if the field has the AUTO attribute, by entering a character into the last data position in the field.

For ***ANSI 74*** and ***ANSI 85***, if the field has the SECURE ECHO attribute, the field is redisplayed as all asterisk (*) characters; if the field has the SECURE NO ECHO attribute, nothing is displayed and the cursor does not move.

For ***VXCOBOL***, if the field has the SECURE attribute, nothing is displayed and the cursor does not move.

d. If the field is terminated by an *ESC function key* (any key with an ESCAPE KEY value of 01), the data entered by the user is discarded, no field validation is performed, the screen field is not changed, the entire accept operation is ended, and the ESCAPE KEY value is set to 01.

e. If the field is terminated by a *field terminator key* (any key with an ESCAPE KEY value of 00), the screen control system checks that the data entered by the user is valid for its PICTURE. It also checks to make sure the data entry requirements implied by REQUIRED and FULL have been met. If there is an error, the screen control system sounds the tone, puts an error message on the last line of the display, and positions the cursor at the location of the error. The user must enter correct data before the field can be terminated. When the field passes the system checks, any error message that was displayed is erased, and the system processes the terminator. If the terminator indicates motion to a previous field, and the field is not the first field, the cursor is positioned to the previous field and the accept operation begins for that field; otherwise, the tone is sounded and the cursor remains at the first field. If the terminator indicates motion to the next field, and the field is not the last field, the cursor is moved to the next field and the accept operation begins for that field; otherwise, the action depends on additional attributes of the terminator. If it is a *field terminator key*, the entire accept operation is completed, and ESCAPE KEY is set to 00.

f. If the field is terminated by a *normal function key* (any key with an ESCAPE KEY value greater than 01), the field validation takes place as for a *field terminator key*. Once the field validation has been successfully completed, the entire accept operation is also terminated, and ESCAPE KEY is set to the value for the terminator.

g. When the entire accept operation is terminated, the screen fields are moved to their corresponding data items. Those fields that were processed during the execution of the ACCEPT will have the new data. Those fields that were not processed (whether due to entering an *ESC function key* or a *normal function key*) will have the old data (for input fields, this will usually be underscores). When the screen field is a numeric-edited item and the data item is a numeric item, the screen field is first de-edited before moving the data, thus only the numeric value is moved. In all other cases, the moves take place according to the rules for the MOVE statement.

(16) If the accept operation was terminated by a *field terminator key* (a key with an ESCAPE KEY value of 00), the ON ESCAPE clause, if specified, is bypassed and control passes to the NOT ON ESCAPE clause, if present, or to the end of the ACCEPT statement.

(17) If the accept operation was terminated by an *ESC function key* or a *normal function key* (any key with an ESCAPE KEY value that is not 00), control passes to the ON ESCAPE clause, if specified. If no ON ESCAPE clause was specified, control passes to the end of the ACCEPT statement.

(18) The value of the ESCAPE KEY is available through the Format 2 ACCEPT FROM ESCAPE KEY statement.

(19) Entries that start past column 128 are undefined. When hard coded, the **ICOBOL** compiler will give an error for entries past column 128. In all other cases, the runtime will behave in an undefined fashion for a particular terminal type.

Format 3:

(20) The ACCEPT statement causes the transfer of data from the keyboard. This data replaces the contents of the data item referenced by *identifier-1* according to the rules for the MOVE statement.

(21) Data input for *identifier-1* must be valid for the identifier. If the characters that are input do not agree with the item's PICTURE, then an error message is displayed on the last line of the display screen, and the input must be corrected. For example, alphabetic characters entered into a numeric item will be rejected. For complete details of PICTURE definitions and acceptable input, see the PICTURE Clause discussion in the WORKING-STORAGE section.

(22) If a *field terminator key* (i.e., any key configured in the terminal description to generate a value of 00) is pressed at any time during an ACCEPT statement, the data is validated and transferred to the data item referenced by *identifier-1*. When a *field terminator key* is pressed for the last *identifier-1,* the ON ESCAPE clause, if present, is bypassed and the NOT ON ESCAPE phrase, if specified, is executed; otherwise, control is transferred to the end of the ACCEPT statement. When a *field terminator key* is pressed for any other *identifier-1,* the ACCEPT statement continues processing with the next *identifier-1*.

(23) If the *ESC function key* (i.e., any key configured in the terminal description to generate a value of 01) is pressed at any time during an ACCEPT statement, the data from the keyboard is discarded, and the data item referenced by *identifier-1* is not changed. If the *ESC function key* is pressed for the last *identifier-1*, the ACCEPT terminates, and the ON ESCAPE clause, if present, is executed. Otherwise, processing continues with the next *identifier-1*.

(24) If a *normal function key* (i.e., any key configured in the terminal description to generate a value that is not 00 or 01) is pressed at any time during an ACCEPT statement, the data is validated and transferred to the data item referenced by *identifier-1*. When a *normal function key* is pressed for the last *identifier-1,* the ON ESCAPE clause, if present, is executed; otherwise, control is transferred to the end of the ACCEPT statement. When a *normal function key* is pressed for any other *identifier-1,* the ACCEPT statement continues processing with the next *identifier-1*.

(25) If the ON ESCAPE clause is executed and *identifier-14* has been specified, the two-digit code generated by the key that terminated the last *identifier-1* is stored into *identifier-14*. This is equivalent to executing an ACCEPT *identifier-14* FROM ESCAPE KEY statement as the first statement of the ON ESCAPE clause.

(26) Format 3 ACCEPTs that extend past a terminal's width are supported by allowing the ACCEPT to wrap to the next line since the screen does not have to scroll; i.e., the wrap would otherwise move to the line after the last line on the screen.

(27) The BACKGROUND-COLOR and FOREGROUND-COLOR phrases determine the background and foreground colors used during the processing of *identifier-1*. The color is identified by an integer value from 0 to 7 specified for *literal-5* or *literal-9* or as the contents of *identifier-6* or *identifier-10*. It may also be specified by use of *color-name-1* or *color-name-2*. The color names with their integer values are BLACK=0, BLUE=1, GREEN=2, CYAN=3, RED=4, MAGENTA=5, BROWN=6, WHITE=7. BACKGROUND is a synonym for BACKGROUND-COLOR and FOREGROUND is a synonym for FOREGROUND-COLOR.

(28) The NO BELL phrase causes suppression of the bell (or beep) signal which normally sounds as each *identifier-1* is processed.

(29) BLINK causes the PROMPT character and any data displayed for the field to be displayed in a blinking mode.

(30) The COLUMN and LINE phrases are used to position *identifier-1* on the screen based on the line and leftmost character position. The top line is line 1 and each succeeding line has a value one larger than the previous line. The leftmost character of a line is column 1 and the column value increases by one for each succeeding character on the line. The line number is specified by *literal-11* or the contents of *identifier-11* and should be between 1 and 128. The column number is specified by *literal-6* or the contents of *identifier-7*.

The line and column positions are determined as follows:

a. If the COLUMN phrase is omitted, column 1 is assumed for the first *identifier-1* or if a UNIT phrase has been specified for the same *identifier-1*. Otherwise the column position is set to zero.

b. If the LINE phrase is omitted or the line position is zero the line position is set as follows: If an ERASE or ERASE SCREEN phrase is specified for the same *identifier-1*, then line 1 is assumed. If the column position is not zero, the line position is the current line plus one. If the column position is zero, the line position is set to the current line.

c. If the column position is equal to zero, it is set to the current line.

At runtime, values outside the allowable ranges are wrapped.

(31) The CONTROL phrase is used to dynamically specify options to be used or overridden. *Identifier-8* or *literal-7* are used to hold an options list. This list consists of a series of keywords separated by commas. The keywords may be specified in any order, but are processed from left to right as they appear in the string. While

processing the list, lowercase characters are considered equivalent to the corresponding uppercase character and blanks or unprintable characters are ignored.

The following keywords impact execution of the ACCEPT statement: BEEP, BLINK, CONVERT, ECHO, ERASE, ERASE EOL, ERASE EOS, ERASE LINE, ERASE SCREEN, HIGH, LOW, LOWER, NO BEEP, NO BLINK, NO CONVERT, NO ECHO, NO ERASE, NO LOWER, NO OFF, NO PROMPT, NO REVERSE, NO SECURE, NO TAB, NO UNDERLINE, NO UPDATE, NO UPPER, OFF, PROMPT, SECURE, SECURE ECHO, SECURE NO ECHO, TAB, UNDERLINE, UPDATE, and UPPER.

Each of the keywords has the same meaning as when statically coded plus the negative versions (NO xxx) to allow suppression of the of the option. The keywords UPPER, LOWER, NO UPPER, and NO LOWER are used to enable or suppress the CONVERTING UP or CONVERTING DOWN options.

(32) The CONVERT phrase is used to control input conversion. If *identifier-1* is numeric and the CONVERT phrase is specified, the data input from the screen is converted to a signed numeric value and stored in *identifier-1* according to the rules for a numeric MOVE. (CONVERT is implied for numeric values unless the "NO CONVERT" is specified as a value for the CONTROL option.) CONVERT is implied by the UPDATE option when *identifier-1* is numeric. If *identifier-1* is numeric and input conversion is not specified either implicitly or explicitly, *identifier-1* is treated as an elementary alphanumeric item of the same size and the unconverted input data is moved to that item according to the rules for an alphanumeric MOVE.

If *identifier-1* is numeric edited and the CONVERT phrase is specified, the data input from the screen is converted to a signed numeric value and stored in *identifier-1* according to the PICTURE of *identifier-1*.

If *identifier-1* is alphanumeric edited and the CONVERT phrase is specified, the data input from the screen is stored in *identifier-1* according to the rules for a alphanumeric to alphanumeric edited MOVE. (CONVERT is implied when *identifier-1* is alphanumeric edited.)

In all other cases or if CONVERT is not specified, data is moved from the screen to *identifier-1* according to the rules for an alphanumeric MOVE.

NOTE: Interactive validation is performed on numeric or numeric edited values whenever the CONVERT option is supplied or implied.

(33) The CONVERTING phrase is used to control the case of the data accepted. If CONVERTING UP is specified character data entered during an ACCEPT is echoed to the screen and stored in uppercase. In particular, characters between 'a' and 'z' inclusive are converted to the corresponding character between 'A' and 'Z'. If CONVERTING DOWN is specified character data entered during an ACCEPT is echoed to the screen and stored in lowercase. In particular, characters between 'A' and 'Z' inclusive are converted to the corresponding character between 'a' and 'z.

(34) The CURSOR option is used to specify the initial cursor position within the screen field. The initial position is specified by *literal-8* or the contents of *identifier-9*. The leftmost position is 1. A value of 0 is treated as one and a value greater than the size of the screen field is treated as the size of the screen field. If *identifier-9* is specified, the cursor position at field termination is returned in it.

(35) The ECHO phrase causes the contents of *identifier-1* to be displayed in the screen field following completion of data input for the field. The display is performed as if a DISPLAY with similar options was performed. Note that CONVERT in an ACCEPT statement controls only input conversion. Output conversion is controlled by the UPDATE phrase. If *identifier-1* is numeric and input conversion was specified or implied, output conversion will be used on the display. If the ECHO phrase is not specified, the original input data remains in the screen field.

(36) The ERASE clause is used to control erasure of portions of the screen prior to accepting data. ERASE SCREEN and ERASE with no additional modifiers erases the entire screen and positions the cursor to line 1 column 1. ERASE LINE erases the current line from column 1 to the end of the line without changing the cursor position. ERASE EOL erase the screen starting at the cursor position to the end of the line. The cursor is not affected.

ERASE EOS erase the screen starting at the cursor position and continuing to the end of the screen. The cursor position is not changed.

(37) The HIGH, HIGHLIGHT, BOLD, and BRIGHT options cause the accepted and displayed data to be displayed at high intensity. The LOW, LOWLIGHT, and DIM options cause the accepted and displayed data to be displayed at low intensity.

(38) The PROMPT clause causes fill characters to be displayed on the screen in the positions in which data is to be accepted. If *literal-11* is not specified, the fill character used is an underscore. If *literal-11* is specified, it must be of length one and represents the fill character. When PROMPT is not specified, no prompting occurs an the original contents of the screen field are not modified unless UPDATE is specified. If both PROMPT and UPDATE are specified, all positions in the screen field not occupied by characters in *identifier-1* are filled with the fill character.

(39) The REVERSE, REVERSED, and REVERSE-VIDEO options cause the accepted and displayed data to be displayed in reverse video mode. If not specified, data is displayed in normal mode.

(40) The SECURE clause controls echoing of input data as it is entered. If either SECURE with no additional options or SECURE WITH ECHO is specified, an asterisk is echoed and the cursor moved right one position as each character is entered. If SECURE NO ECHO is specified, no echoing or cursor movement takes place. If OFF is specified, a space is echoed and the cursor moved right one position as each character is entered.

(41) The SIZE clause controls the size of the screen input field. If the SIZE clause is present and *literal-12* or the contents of *identifier-12* is not zero, the size of the screen field is determined by the value of *literal-12* or *identifier-12*. Otherwise, the size of the screen field is determined by description of *identifier-1*.

When *identifier-1* is numeric and input conversion is specified or implied, the size is the number of digits in *identifier-1's* PICTURE plus 1 if its is signed plus 1 if it is not an integer. When identifier-1 is numeric and input conversion is not specified, the size value is determined by the number of bytes of stored required for *identifier-1*.

(42) The TAB clause causes the ACCEPT statement to wait for a field termination key to be pressed before completing the accept of the screen field. If the TAB clause is not present, the field will terminate when the end of the screen input field is reached or when a field termination key is pressed. (When TAB is absent, the field behaves much like an AUTO field in a screen description.)

(43) The BEFORE TIME clause is used to specify an interval of time to wait before automatically terminating the field when no data has been entered. *Literal-13* or the contents of *identifier-13* are integer values which specify this time interval in hundredths of seconds. If the user enters any data in the field prior to the expiration of the time interval, then the timer is cancelled and the ACCEPT of the field behaves as if no BEFORE TIME clause was specified. Valid values and their behavior are:

Time-out value	Meaning
>= 4,294,967,295	No time-out (Wait forever)
0	Time-out immediately
> 630000	Set to 6300 seconds
1-630000	Set to n seconds

If the specified time interval completes before any data is entered, the field is terminated as if a Newline or Enter key was pressed. The escape key code returned will be 99.

NOTE: The TIME-OUT clause described below is similar, but is expressed in seconds and does not have to be an integer. It represents a time to wait between keystrokes before terminating a field. If the time-out occurs it behaves as if the ESC key were pressed. Both TIME-OUT and BEFORE TIME may not be specified for the same *identifier-1*.

(44) The UNDERLINE and UNDERLINED options cause the accepted and displayed data to be displayed in underlined mode.

(45) The UNIT clause is for documentation only and is ignored except for its impact on the COLUMN clause as previously described.

(46) The UPDATE clause control output conversion of the current value of *identifier-1*. This option changes the contents from its internal form into a form appropriate for display. The user may then modify the screen field and upon field termination the data in the screen field is stored with input conversion back into *identifier-1*.

With output conversion, numeric data is converted such that a leading separate sign is provided for negative values, an explicit decimal point is added for non-integers, leading zeros are removed and the remaining digits are left-justified.

If both UPDATE and CONVERT are specified for a numeric edited item, a numeric value for *identifier-1* is determined by the rules for a MOVE from a numeric edited item to numeric item. The numeric values is then converted as described above. If *identifier-1* is numeric edited, but only the UPDATE clause is present, then it is not converted before display.

Output conversion does not itself change the value of *identifier-1*, but only the appearance of data in the screen field. The UPDATE clause signals output conversion, and implies input conversion. Unlike with the DISPLAY statement, CONVERT does not signal output conversion, but rather signals input conversion.

All formats:

(47) The TIME-OUT phrase enables a local time-out for the particular ACCEPT statement. If provided, it overrides any other specified time-out value. The time-out specifies the amount of time, in seconds, that the runtime will wait between keystrokes. If the time expires, the ACCEPT terminates as if an ESCAPE had been struck and sets the ESCAPE KEY value to 99. Valid values are:

Time-out value	Meaning
<= 0 or >= 65535	No time-out (Wait forever)
65534	Time-out immediately
> 6300	Set to 6300 seconds
1-6300	Set to n seconds

(48) If the time-out value specified by *identifier-4* or *literal-3* is not an integer, its value is rounded to the nearest tenth of a second..

(49) When using timeouts, **ICOBOL** handles them in the following order for both the ACCEPT statement and the STOP literal statement:

a. If a local timeout was specified by the TIME-OUT or BEFORE TIME clause of the ACCEPT statement, then it is used; otherwise,

b. If a timeout had been set with the IC_SET_TIMEOUT builtin, then it is used; otherwise,

c. The global timeout as set with ICTIMEOUT will be used. The default case for global timeout is to wait forever.

NOTE:

Using an extended open option to set timeout on your console does NOT affect an ACCEPT or STOP statement. Extended open options are discussed later Developer's Guide Section.

IC_SET_TIMEOUT is discussed in this document beginning on page 558, 559.

E.2. ACCEPT (system)

E.2.1 Function

The ACCEPT (system) statements cause data from the system to be made available to data items in the File, Working-Storage, or Linkage sections.

ENVIRONMENT, ESCAPE KEY, EXCEPTION STATUS, LINE NUMBER, and USER NAME are extensions to ANSI COBOL.

E.2.2 General Format (***ANSI 74*** and ***ANSI 85***)

Format 1:

```
ACCEPT identifier FROM { DATE [ YYYYMMDD ]
                       | DAY [ YYYYDDD ]
                       | DAY-OF-WEEK
                       | TIME
                       | TIMESTAMP }
```

Format 2:

```
ACCEPT identifier FROM { ENVIRONMENT
                       | ESCAPE KEY
                       | EXCEPTION STATUS [ WITH ERROR IN identifier-5 ]
                       | LINE NUMBER
                       | USER NAME }
```

E.2.3 General Format (***VXCOBOL***)

Format 1:

```
ACCEPT identifier FROM { DATE [ YYYYMMDD ]
                       | DAY [ YYYYDDD ]
                       | TIME
                       | TIMESTAMP }
```

Format 2:

```
ACCEPT identifier FROM { ENVIRONMENT
                       | ESCAPE KEY
                       | EXCEPTION STATUS [ WITH ERROR IN identifier-5 ]
                       | USER NAME }

ACCEPT identifier FROM LINE NUMBER
d       [ ON VIRTUAL TERMINAL imperative-statement [ END-ACCEPT ]]
```

E.2.4 Syntax Rules

(1) (***ISQL***) In Format 1, the TIMESTAMP phrase may only be specified if the ***ISQL*** feature-set is enabled.

(2) (***ISQL***) If *identifier* specifies an item of class date-time and category date, the DATE phrase must be specified. The YYYYMMDD phrase is implied if it is omitted.

(3) (***ISQL***) If *identifier* specifies an item of class date-time and category time, the TIME phrase must be specified.

(4) (***ISQL***) If *identifier* specifies an item of class date-time and category timestamp, the TIMESTAMP phrase must be specified.

E.2.5 General Rules

(1) The ACCEPT statement causes the information requested to be transferred to the data item specified by *identifier* according to the rules for the MOVE statement. DATE, DAY, DAY-OF-WEEK TIME, and TIMESTAMP reference the current date and time provided by the system on which the ACCEPT statement is executed. DATE, DAY, DAY-OF-WEEK and TIME are standard COBOL conceptual data items and, therefore, are not described in the COBOL program. TIMESTAMP, ENVIRONMENT, ESCAPE KEY, EXCEPTION STATUS, LINE NUMBER, and USER NAME are conceptual data items and, therefore, are not described in the COBOL program.

(2) DATE, without the phrase YYYYMMDD, is composed of the data elements: year of century, month of year, and day of month (yymmdd). Therefore, December 25, 1986, would be expressed as 861225. DATE without the phrase YYYYMMDD, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item six digits in length (PIC 9(6)).

(3) DATE, with the phrase YYYYMMDD behaves as it had been described as an unsigned elementary integar data item of usage display eight digits in length, the character positions of which, numbered from left to right, are:

Character Positions	Contents
1-4	Four numeric characters of the year in the Gregorian calendar.
5-6	Two numeric characters of the day of the year in the range 01 through 12.
7-8	Two numeric characters of the day of the month in the range 01 through 31.

(4) DAY, without the phrase YYYYDDD, is composed of the data elements: year of century and day of year (yyddd). Therefore, December 25, 1986, would be expressed as 86359. DAY, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item five digits in length (PIC 9(5)).

(5) DAY with the phrase YYYYDDD behaves as if it had been described as an unsigned elementary integer data item of usage display seven digits in length, the character positions of which, numbered from left to right are:

Character Positions	Contents
1-4	Four numeric characters of the year in the Gregorian calendar.
5-7	Three numeric characters of the day of the year in the range 001 through 366.

(6) TIME is composed of the data elements hours, minutes, seconds, and hundredths of a second (hhmmsshh). TIME is based on elapsed time after midnight on a 24-hour clock basis; thus, 2:41 p. m. would be expressed as 14410000. TIME, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item eight digits in length (PIC 9(8)). The minimum value of TIME is 00000000; the maximum value of TIME is 23595999. If the system does not have the facility to provide fractional parts of a second, the value zero is returned for those parts which cannot be determined (e.g., 386UNIX returns 00 as the hundredths of a second in the seventh and eight character positions).

> **NOTE**: If the ***ISQL*** feature-set is enabled, one can use the TIMESTAMP features to retrieve a date and time as one operation. Otherwise, the recommended method is to use either the IC_FULL_DATE builtin call or the CURRENT-DATE function. Each of the three methods return a four-digit year and assure that the both date and time were retrieved without the system crossing midnight, which can occur if one uses separate ACCEPT FROM DATE and ACCEPT FROM TIME statements. IC_FULL_DATE is discussed in this document beginning on page 518, and the CURRENT-DATE function is discussed on page 594.

(7) DAY-OF-WEEK is composed of a single data element whose content represents the day of the week. DAY-OF-WEEK, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item one digit in length. In DAY-OF-WEEK, the value 1 represents Monday, 2 represents Tuesday, ... , 7 represents Sunday.

(8) (***ISQL***) TIMESTAMP is composed of a 4-digit year field, a 2-digit month field, a 2-digit day field, a 2-digit hour field, a 2-digit minute field, a 2-digit second field, and a 2-digit hundredths of second field. It is equivalent to SQL TIMESTAMP(2). Conceptually it is equivalent to PIC 9(16). If <*identifier*> is a timestamp, then the internal timestamp will have all 6 fractional digits for seconds.

(9) ENVIRONMENT is composed of a structure containing specific information for a particular operating system environment. The amount of data transferred depends on the environment and the revision of the runtime system. For revision 3.30 of **ICOBOL**, the structure is defined as follows:

```
01 ENV-STRUCTURE.
    02 SYSTEM-CODE                    PIC 99.
       88 IC-AOSVS                          VALUE IS 01.
       88 IC-AOSVSII                        VALUE IS 04.
       88 IC-MSDOS                          VALUE IS 30.
       88 IC-386UNIX                        VALUE IS 31.
       88 IC-DGUX-88K                       VALUE IS 34.
       88 IC-AIX-RS                         VALUE IS 39.
       88 IC-SUN-SPARC                      VALUE IS 40.
       88 IC-HPUX-PA-RISC                   VALUE IS 41.
       88 IC-MOTOROLA-88K                   VALUE IS 43.
       88 IC-STRATUS-860                    VALUE IS 44.
       88 IC-LINUX-INTEL                    VALUE IS 45.
       88 IC-DGUX-INTEL                     VALUE IS 47.
       88 IC-SCO-UNIX-INTEL                 VALUE IS 48.
       88 IC-UNIXWARE-INTEL                 VALUE IS 49.
       88 IC-WINDOWS-9X                     VALUE IS 60.
       88 IC-WINDOWS-NT-INTEL               VALUE IS 61
    02 REVISION-CODE                  PIC 99.
    02 PROGRAM-NAME                   PIC X(28).
    02 PID                            PIC 9(5).
    02 CONSOLE-TYPE                   PIC X.
       88 CON-BATCH                         VALUE IS "B".
       88 CON-NORMAL                        VALUE IS "C".
       88 CON-MASTER                        VALUE IS "M".              (end rev 00)
    02 SCREEN-LINES                   PIC 9(3).
    02 SCREEN-COLUMNS                 PIC 9(3).
    02 PRIVILEGES                     PIC X(16).
    02 PRIV-REDEF REDEFINES PRIVILEGES.
       03 ABORT-PROGRAM               PIC X.
       03 INTERNAL-INFORMATION        PIC X.
       03 MESSAGE-SENDING             PIC X.
       03 TERMINAL-STATUS             PIC X.
       03 PRINTER-CONTROL             PIC X.
       03 PRINTER-CONTROL-MGMT        PIC X.
       03 SHUTDOWN-RUNTIME            PIC X.
       03 BG-CONSOLE-OR-HOST-EXEC     PIC X.
       03 CONSOLE-INTERRUPT           PIC X.
       03 DEBUG-PROGRAM               PIC X.
       03 WATCH-FACILITY              PIC X.
       03 XWATCH-FACILITY             PIC X.                           (new rev 05)
       03 FILLER                      PIC X(4).
    02 FILENAME-CASE                  PIC X.
       88 CONVERT-TO-LOWER                  VALUE "L".
       88 CONVERT-TO-UPPER                  VALUE "U".
       88 CONVERT-NONE                      VALUE "N".                 (end rev 01)
    02 ICREV-INFO                     PIC X(8).                        (end rev 02)
    02 PROGRAM-TYPE                   PIC X.
       88 NORMAL-PROGRAM                    VALUE IS "N".
       88 HOTKEY-PROGRAM                    VALUE IS "H".
       88 NORMAL-PROGRAM-CHILD              VALUE IS "C".
    02 MAX-LEVELS                     PIC 99.
    02 CURRENT-LEVEL                  PIC 99.                          (end rev 03)
    02 LARGE-PID                      PIC 9(10).                       (end rev 04)
    02 SCREEN-COLUMNS-MIN             PIC 9(3).
    02 SCREEN-COLUMNS-MAX             PIC 9(3).
    02 SYS-NODENAME                   PIC X(16).                       (end rev 05)
```

Where

SYSTEM-CODE indicates that this COBOL program is currently running under **ICOBOL** on the operating system corresponding to the 2-digit code that is returned. New codes are added as additional systems are supported. Please see the **ICOBOL** product's README file for the latest values. The current system-code can be overridden when starting the runtime with the Set System code switch (-S).

REVISION-CODE indicates the current revision of this structure under **ICOBOL** for this system and is set to 05 for this revision.

PROGRAM-NAME is the current program that is running (i.e., the same as would be seen by a IC_TERM_STAT on another console).

PID is the current process id.

CONSOLE-TYPE is `B' if this process is a batch job or detached program or otherwise has the standard input set to the null device; `C' if it is attached to an interactive console, or `M' if this is console 0 in the configuration file (.cfi), even if the Master Console has been reset to a console number other than 0 by use of the Lowest Console number switch to ICEXEC.

SCREEN-LINES and SCREEN-COLUMNS is the number of lines and columns that **ICOBOL** is currently using for this terminal. When in Batch mode these numbers are undefined.

PRIVILEGES contains characters defining the privileges that the current program has. If the privilege is granted the indicated column will contain the letter specified, otherwise the column will contain a space.

Position	Contents	Meaning
1	A	User can run Abort Terminals
2	I	User can run System Information
3	M	User can run Message Sending
4	T	User can run Terminal Status
5	P	User can run Printer Control
6	C	User has printer control management
7	S	User can run Shutdown
8	O	User can Detach jobs or call host
9	B	Program Interrupts are allowed
10	D	User can debug
11	W	User can use the Watch Facility
12	X	This user can NOT be watched
13-16	space	Undefined (reserved)

FILENAME-CASE contains the case that **ICOBOL** on UNIX is using for filenames, i.e., the -C value from the command line as U=upper, L=lower, and N=none.

ICREV-INFO contains the 8-byte string set with the ICREVSET utility or with the compiler OEM Version Switch (-o|-O *ver*).. If not set, it will contain nulls (LOW-VALUES).

PROGRAM-TYPE is 'H' if the current program was called via a hotkey, 'C' if the current program was called from within a hotkey program, or 'N' if the current program is a normal program.

FILLER will always contain zeros (00). (Formerly MAX-LEVELS, the maximum configured number of CALL levels allowed. This item is obsolete.)

CURRENT-LEVEL shows the current number of active and inactive programs in this run-unit. If greater than 99, then only 99 is shown.

LARGE-PID shows a 10 character pid number on those systems that support larger pid ranges, otherwise LARGE-PID matches PID

SCREEN-COLUMNS-MIN, SCREEN-COLUMNS-MAX is the minimum and maximum values for a terminal that supports compressed mode.

SYS-NODENAME is a 16 character name of the current computer.

.

ICOBOL sets batch job (`B' in CONSOLE-TYPE) when it detects that the standard input is set to the null device. An ACCEPT will generate an immediate end-of-file. All programs started with the IC_DETACH will be considered as batch jobs. CGICOBOL programs are considered as batch jobs.

The runtime system uses the rules for a MOVE statement to transfer data into the environment structure. If the identifier is smaller than the data, data is truncated on the right. If the identifier is larger, the data is left-justified and the identifier is padded with spaces.

(10) ESCAPE KEY contains a two-digit (PIC 99) code generated by the key that terminated the last Format 3 (ACCEPT identifier-1) or Format 4 (ACCEPT screen-name) ACCEPT statement in the program. It should be queried immediately after the ACCEPT you wish to test.

The ESCAPE KEY will return a zero if a valid ACCEPT has not been done since the program was started via either a CALL PROGRAM or CALL.

The following table shows the default ESCAPE KEY codes for a Data General D2xx compatible terminal.

Key	Key alone	Key + SHIFT	Key + CTRL	Key + SHIFT+CTRL
CR	00	00	00	00
NEWLINE	00	00	00	00
ESC	01	01	01	01
F1	02	10	18	26
F2	03	11	19	27
F3	04	12	20	28
F4	05	13	21	29
F5	06	14	22	30
F6	07	15	23	31
F7	08	16	24	32
F8	09	17	25	33
F9	34	41	48	55
F10	35	42	49	56
F11	36	43	50	57
F12	37	44	51	58
F13	38	45	52	59
F14	39	46	53	60
F15	40	47	54	61
C1	62	66	62	66
C2	63	67	63	67
C3	64	68	64	68
C4	65	69	65	69
Down-arrow	00	77	00	00
Up-arrow	n/a	70	n/a	70
Right-arrow	n/a	71	n/a	71
Left-arrow	n/a	72	n/a	72
CMD-Print	73	74	73	74
HOME	n/a	75	n/a	75

TABLE 19. Function Key Escape Codes

Escape key codes are configurable in the terminal description files (.tdi) on a terminal type basis. See the Installing and Configuring manuals for complete details.

(11) EXCEPTION STATUS, without the WITH ERROR IN phrase, contains a five-digit (PIC 9(5)) code, for the most recent I/O operation. This includes all I/O operations: file I/O (which also set File Status), plus ACCEPT, DISPLAY, CALL and CALL PROGRAM. The returned Exception Status value can be used with the IC_MSG_TEXT builtin to get the error message text for the particular error.

Remember: to retrieve the correct status, the ACCEPT FROM EXCEPTION STATUS must be issued prior to any further I/O or CALL operation, including screen I/O operations.

APPENDICES F (**ANSI**) and G (**VXCOBOL**), starting on pages 827 and 835 respectively, show all possible Exception Status values with their meaning, along with any UNIX or Windows error that will generate that Exception Status.

If a fatal I/O error is encountered and the program terminates, the current Exception Status is displayed right after the COBOL PC as E=nnn.

(12) EXCEPTION STATUS with the WITH ERROR IN phrase returns the operating system error that caused the exception, if such was the case. The program's definition of *identifier-5* should be PIC 9(5).

(13) LINE NUMBER contains a five-digit number of the console number (n of @CONn) on which this program is running. Its PICTURE is 9(5). The ON VIRTUAL TERMINAL clause, available for ***VXCOBOL***, is for documentation purposes only and is therefore ignored.

(14) USER NAME contains the current system user name (if available) of the user currently running this program. Up to 15 characters are returned; i.e., its PICTURE is X(15). By default, the user name is returned in lower-case. A runtime switch (-U) may be specified to convert the case of the user name that is returned by ACCEPT FROM USER NAME. The name may be changed by the IC_SET_USERNAME builtin, which is discussed in this document beginning on page 560.

E.3. ADD

E.3.1 Function

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

E.3.2 General Format

Format 1:

```
ADD { identifier-1 } ...          TO { identifier-2 [ ROUNDED ] } ...
    { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-ADD ]
```

Format 2:

```
ADD { identifier-1 } ...  { identifier-2 } TO          GIVING { identifier-3 [ ROUNDED ] } ...
    { literal-1    }      { literal-2    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-ADD ]
```

Format 3:

```
ADD { CORRESPONDING } identifier-1 TO identifier-2 [ ROUNDED ]
    { CORR          }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-ADD ]
```

E.3.3 Syntax Rules

(1) In Formats 1 and 2, each identifier must refer to an elementary numeric item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item. In Format 3, each identifier must refer to a group item.

(2) Each literal must be a numeric literal.

(3) The composite of operands must not contain more than 18 digits.

a. In Format 1, the composite of operands is determined by using all of the operands in a given statement.

b. In Format 2, the composite of operands is determined by using all of the operands in a given statement, excluding the data items that follow the word GIVING.

c. In Format 3, the composite of operands is determined separately for each pair of corresponding data items.

(4) CORR is an abbreviation for CORRESPONDING.

E.3.4 General Rules

(1) If Format 1 is used, the values of the operands preceding the word TO are added together and the sum is stored in a temporary data item. The value in this temporary data item is added to the value of the data item referenced by *identifier-2*, with the result stored into the data item referenced by identifier-2. This process is repeated for each successive occurrence of *identifier-2,* in the left-to-right order in which *identifier-2* is specified.

(2) If Format 2 is used, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new content of each data item referenced by *identifier-3*.

(3) If Format 3 is used, data items in *identifier-1* are added to and stored in corresponding data items in *identifier-2*.

(4) The compiler insures that enough places are carried, so as not to lose any significant digits during execution.

(5) Additional rules and explanations relative to this statement are given under the appropriate paragraphs. (See Scope of Statements, page 253; The ROUNDED Phrase, page 245; The ON SIZE ERROR Phrase, page 246; The Arithmetic Statements, page 249; Overlapping Operands, page 249; Multiple Results in Arithmetic Statements, page 249; and The CORRESPONDING Phrase, page 246.)

E.4. CALL

E.4.1 Function

The CALL statement causes control to be transferred from one object program to another, within the run unit or to an external executable program as defined by a particular **ICOBOL** operating system version.

To see how **ICOBOL** processes the program name see the External Filename description in the Developer's Guide section on page 751.

E.4.2 General Format

Format 1:

```
CALL { identifier-1 }        { [ BY REFERENCE ] identifier-2... }
     {   literal    }        {   [ USING ]
                             {   BY CONTENT identifier-2...     }                ... ]
    [ ON EXCEPTION imperative-statement-1 ]
    [ NOT ON EXCEPTION imperative-statement-2 ]
    [ END-CALL ]
```

Format 2:

```
CALL { identifier-1 }        { [ BY REFERENCE ] identifier-2... }
     {   literal    }        {   [ USING ]
                             {   BY CONTENT identifier-2...     }                ... ]
    [ ON OVERFLOW imperative-statement-1 ]
    [ END-CALL ]
```

E.4.3 Syntax Rules

(1) *Literal* must be a nonnumeric literal.

(2) *Identifier-1* must be defined as an alphanumeric data item such that its value can be a program-name.

(3) Each of the operands (*identifier-2*) in the USING phrase must have been defined as a data item in the File Section, Working-Storage Section, or Linkage Section.

E.4.4 General Rules

(1) *Literal* or the content of the data item referenced by *identifier-1* must contain the name of the called program. The program in which the CALL statement appears is the calling program.

(2) If, when a CALL statement is executed, the program specified by the CALL statement is made available for execution, control is transferred to the called program. After control is returned from the called program, the ON OVERFLOW or ON EXCEPTION phrase, if specified is ignored and control is transferred to the end of the CALL statement or, if the NOT ON EXCEPTION phrase is specified, to *imperative-statement-2*. If control is transferred to *imperative-statement-2*, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the CALL statement.

(3) If it is determined, when a CALL statement is executed, that the program specified by the CALL statement cannot be made available for execution at that time the appropriate Exception Status is set and one of the two actions listed below will occur.

a. If the ON OVERFLOW or ON EXCEPTION phrase is specified in the CALL statement, control is transferred to *imperative-statement-1*. Execution then continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the CALL statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

b. If the ON OVERFLOW or ON EXCEPTION phrase is not specified in the CALL statement, the NOT ON EXCEPTION phrase, if specified, is ignored, and control is transferred to the end of the CALL statement.

(4) If the called program does not possess the initial attribute, the called program is in its initial state the first time it is called within a run unit and the first time it is called after a CANCEL to the called program. On all other entries into the called program, the state of the program remains unchanged from its state when last exited.

If the called program possesses the initial attribute it is placed into its initial state every time the called program is called within a run unit.

(5) Files associated with a called program's internal file connectors are not in the open mode when the program is in an initial state.

On all other entries into the called program, the states and positioning of all such files is the same as when the called program was last exited.

External file connectors always maintain their state across a CALL.

(6) The USING phrase is included in the CALL statement only if there is a USING phrase in the Procedure Division header of the called program, in which case the number of operands in each USING phrase must be identical. If the program being called is other than a COBOL program, the use of the USING phrase is defined by the program being called. For example, builtins define the expected operands.

(7) The sequence of appearance of the data-names in the USING phrase of the CALL statement and in the corresponding USING phrase in the called program's Procedure Division header determines the correspondence between the data-names used by the calling and called programs. This correspondence is *positional* and not by name equivalence; the first data-name in one USING phrase corresponds to the first data-name in the other, the second to the second, etc.

(8) The values of the parameters referenced in the USING phrase of the CALL statement are made available to the called program at the time the CALL statement is executed.

(9) Both the BY CONTENT and BY REFERENCE phrases are transitive across the parameters which follow them until another BY CONTENT or BY REFERENCE phrase is encountered. If neither the BY CONTENT nor BY REFERENCE phrase is specified prior to the first parameter, the BY REFERENCE phrase is assumed.

(10) For a parameter that is described either explicitly or implicitly as BY REFERENCE, the object program operates as if the corresponding data item in the called program occupies the same storage area as the data item in the calling program. The description of the data item in the called program must describe the same number of character positions as described by the description of the corresponding data item in the calling program.

(11) For a parameter that is described as BY CONTENT, the object program operates as if the storage area in the calling program is copied to a storage area reserved in the LINKAGE Section of the called program, by the USING phrase in the Procedure Division header, for the corresponding item in the USING phrase of the CALL. The storage area of the calling program remains unchanged when the EXIT PROGRAM statement is executed in the called program. The description of the data item in the called program must describe the same number of character

positions as described by the description of the corresponding data item in the calling program. See Values of Parameters on page 56 for more information.

(12) Called programs may contain CALL statements. However, a called program must not execute a CALL statement that directly or indirectly calls the calling program. If a CALL statement is executed within the range of a declarative, that CALL statement cannot directly or indirectly reference any called program in which control has been transferred and which has not completed execution.

(13) The maximum number of parameters that may be specified in a USING phrase is 32.

(14) The CALL statement cannot pass switches to a called program. Switches are the same for the entire run unit.

(15) A few of the more-common error conditions and their exception status codes are:

Exception Status Code	Error Condition
83	The file does not have the correct revision
203	Program not found
207	Program is already active
209	Parameter count or parameter size mismatch
213	Program file cannot be loaded.

TABLE 20. Common Error Conditions for a CALL Statement

(16) The END-CALL phrase delimits the scope of the CALL statement.

(17) CALL can be used to execute user-written C subroutines that have been bound into the currently executing runtime by using the **ICOBOL** Link Kit. In most cases these user-written are bound in dynamically using icbltn.so (UNIX) or icbltn.dll (Windows). See the readlink.txt file in the **ICOBOL** link_kit subdirectory for details.

(18) CALL can be used to execute operating system executable programs.

E.4.5 Calling Operating System Executables

(1) The name of the executable file *literal-1* or the contents of *identifier-1* must begin with the special character vertical bar ("|") which indicates that the name following is an executable file and should be passed to the operating system to be executed with the given arguments *identifier-2* and then return to **ICOBOL** when finished.

(2) If the program specified cannot be executed, the Exception Status is set and the ON EXCEPTION clause, if specified, will be performed. Otherwise, the returned error code is placed into Exception Status, but the ON EXCEPTION clause is not executed.

(3) By using the CALL to an operating system executable, other copies of **ICOBOL** or **ICOBOL** utilities can be started from within a COBOL program. These other processes will get console numbers from the range of consoles that have been enabled with no device name specified; thus, you will never have another **ICOBOL** runtime running with the same console number.

(4) Multiple arguments can be passed but the contents of the data items are never modified by the operating system executables.

UNIX examples

EXAMPLE: To call the Bourne shell you could use the following:

```
MOVE "-s" TO ARGUMENT.
CALL "|sh" USING ARGUMENT.
```

EXAMPLE 17. CALL the Bourne shell from a COBOL program (UNIX)

The above example code starts the *sh* program with the initial argument "-s", which tells the shell to use stdin and stdout for its input and output. When the shell is terminated, control returns to **ICOBOL**, with the exit code being stored into Exception Status.

EXAMPLE: To call the shell and have it execute a single "ls" command and return, use the following:

```
MOVE "-c"    TO ARGUMENT1.
MOVE "ls -l" TO ARGUMENT2.
CALL "|sh"   USING ARGUMENT1, ARGUMENT2.
```

EXAMPLE 18. CALL the shell, have it execute "ls" and return (UNIX)

EXAMPLE: To call the ls command directly and return, use the following:

```
MOVE "-l"   TO ARGUMENT.
CALL "|ls"  USING ARGUMENT.
```

EXAMPLE 19. CALL the "ls" command directly and return (UNIX)

Windows examples

EXAMPLE: To call the Windows command processor.

```
CALL "|c:\winnt\system32\cmd.exe".
```

EXAMPLE 20. CALL the command processor (Windows)

EXAMPLE: To call the Windows command processor and have it executed the DIR command:.

```
MOVE "/C DIR" TO ARGUMENT.
CALL "|c:\winnt\system32\cmd.exe" USING ARGUMENT.
```

EXAMPLE 21. CALL the command processor and execute the DIR command (Windows)

E.5. CALL PROGRAM

E.5.1 Function

The CALL PROGRAM statement begins a new run unit with another COBOL program or it performs a system function as defined by a particular **ICOBOL** operating system version. CALL PROGRAM is an extension to ANSI COBOL. Also see the table, CALL and CALL PROGRAM Compared, at the end of this description.

To see how **ICOBOL** processes the program name see the External Filename description in the Developer's Guide section on page 751.

E.5.2 General Format

```
CALL PROGRAM { identifier-1 }                    [ USING { identifier-2 }... ]
             { literal      }
    [ ON EXCEPTION imperative-statement-1 ]
    [ NOT ON EXCEPTION imperative-statement-2 ]
    [ END-CALL ]
```

E.5.3 Syntax Rules

(1) *Literal* must be a nonnumeric literal.

(2) *Identifier-1* must be defined as an alphanumeric data item such that its value can be a program-name.

(3) In addition to the program-name, *literal* or *identifier-1* can include program switches, each a nonnumeric literal. For example:

```
CALL PROGRAM "REPORT/M/WEEKLY/QUARTERLY".
```

(4) Each of the operands in the USING phrase must have been defined as a data item in the File Section, Working-Storage Section, or Linkage Section.

E.5.4 General Rules

(1) *Literal* or the content of the data item referenced by *identifier-1* is the name of the called program and possibly program switches. The program in which the CALL PROGRAM statement appears is the calling program. *Literal* or the content of the data item referenced by *identifier-1* must contain the program-name of the program to be called or the system call to be executed.

(2) If, when a CALL PROGRAM statement is executed, the program specified by the statement is a COBOL program, it is made available for execution, all files in the current program are closed, and control is transferred to the called program. The successful transfer of control to a called program is equivalent to the execution of a STOP RUN statement within the calling program followed by the start of the called program. (You cannot return to the original or calling program, except when doing system calls.)

(3) If, when a CALL PROGRAM statement is executed, the program specified by the CALL PROGRAM statement is a system call, it is executed in accordance with the specifications for that system call. A system call is defined to be any program name starting with the `#' symbol. Valid system calls for a particular operating system and their function can be found in this manual in APPENDIX L beginning on page 851. If the specified system call returns to the program, and the NOT ON EXCEPTION phrase is specified control is transferred to *imperative-statement-2*; otherwise, control is transferred to the end of the CALL PROGRAM statement.

(4) If it is determined, when a CALL PROGRAM statement is executed, that the program specified by the CALL PROGRAM statement cannot be made available for execution at that time, the exception status is set to the appropriate value and one of the two actions listed below will occur.

a. If the ON EXCEPTION phrase is specified in the CALL PROGRAM statement, control is transferred to *imperative-statement-1*. Execution then continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the CALL PROGRAM statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

b. If the ON EXCEPTION phrase is not specified in the CALL PROGRAM statement, control is transferred to the end of the CALL PROGRAM statement and the NOT ON EXCEPTION phrase, if specified, is ignored.

(5) A few of the more common error conditions and their exception status codes are:

Exception Status Code	Error Condition
83	The file does not have the correct revision
203	Program not found, or this is a system call and the system call is not valid for the operating system
213	Program file could not be loaded. program

TABLE 21. Common Error Conditions for a CALL PROGRAM Statement

(6) The END-CALL phrase delimits the scope of the CALL PROGRAM statement.

(7) The USING phrase can be included in the CALL PROGRAM statement even if there is not a USING phrase in the Procedure Division header of the called program, in which case no parameters are passed to the called program. If the program being called is other than a COBOL program, the use of the USING phrase is defined by the program being called.

(8) The sequence of appearance of the data-names in the USING phrase of the CALL PROGRAM statement and in the corresponding USING phrase in the called program's Procedure Division header determines the correspondence between the data-names used by the calling and called programs. This correspondence is *positional* and not by name equivalence; the first data-name in one USING phrase corresponds to the first data-name in the other, the second to the second, etc.

(9) The values of the parameters referenced in the USING phrase of the CALL PROGRAM statement are made available to the called program at the time the CALL PROGRAM statement is executed.

(10) For a parameter, the object program operates as if the storage area in the calling program is copied to a storage area reserved in the LINKAGE Section of the called program by the USING phrase in the Procedure Division header for the corresponding item in the USING phrase of the CALL PROGRAM. The description of the data item in the called program does not have to describe the same number of character positions as described by the description of the corresponding data item in the calling program. If more bytes are passed than can be stored the extra bytes are ignored. If not enough bytes are passed the resulting storage is undefined.

(11) Without the **ICOBOL** runtime options '-G s' (Strict switch processing) or '-N e' (No embedded spaces) the following default rules describe how the runtime extracts switches from the value of *literal* or *identifier-1*:

a. The switch character is the forward slash '/'.

b. Using ':' or '\' as pathname separators or an initial '=' or '^' removes all ambiguity; i.e., everything starting with '/' is a switch.

c. Switches may be multiple characters.

d. Single character switches follow this special rule: All /x pairs are removed (beginning at the right and moving left) and treated as switches except for a pair occurring as the first 2 characters. This rule is in effect for compatibility with existing applications where program switches are a single character.

e. The first " /" (that's a space followed by a /) (from left to right) will always end a program name and begin program switches.

f. All processing is discontinued at the first CR, NL, FF or NUL.

g. By default, embedded spaces are allowed in *literal* or *identifier-1*.

(12) With the '-G s' (Strict switch processing) runtime option, a '/' in the value in *literal* or *identifier-1* always signals the start of a program switch.

(13) With the '-N e' (No embedded spaces) runtime option, embedded spaces are not allowed in program names, and processing of *literal* or *identifier-1* is discontinued at the first space not preceding either a '/' or spaces preceding a '/'.

(14) The following table shows example values for *literal* or *identifier-1* and how they are evaluated and processed by the **ICOBOL** runtime. It shows differences between a runtime that was brought up using the '-G s' option and a runtime that was brought up without that option.

		DEFAULT Behavior (without '-G s')		WITH '-G s' runtime option	
		Program	Switches	Program	Switches
1.	/x/a/b/c	/x	a, b, c	<error>	
2.	/x/a/b /c	/x/a/b	c	<error>	
3.	/x/a/b /c/d /e	/x/a/b	c, d, e	<error>	
4.	x/a/b/c	x	a, b, c	x	a, b, c
5.	xxx/a/b/c	xxx	a, b, c	xxx	a, b, c
6.	x/a/b/ccc	x/a/b/ccc	<none>	x	a, b, ccc
7.	aaa/bbb/c	aaa/bbb	c	aaa	bbb, c
8.	aaa/bbb/ccc	aaa/bbb/ccc	<none>	aaa	bbb, ccc
9.	aaa\bbb/ccc	aaa/bbb	ccc	aaa/bbb	ccc
10.	aaa:bbb/ccc	aaa/bbb	ccc	aaa/bbb	ccc
11.	=aaa/bbb/ccc	./aaa	bbb, ccc	./aaa	bbb, ccc
12.	^aaa/bbb/ccc	../aaa	bbb, ccc	../aaa	bbb, ccc
13.	aaa/bbb /ccc	aaa/bbb	ccc	aaa	bbb, ccc
14.	aaa/x<nl>/b /ccc	aaa	x	aaa	x
15.	my dir/my prg /sw [1]	my_dir/my prg	sw	my_dir	my

[1] - The last example shows the default behavior as far as allowing embedded spaces. With the '-N e' runtime option (no embedded spaces), an error is returned.

TABLE 22. How Program Switches are evaluated

E.5.5 CALL and CALL PROGRAM Compared

This table presents a high-level view of the major differences between the CALL statement and the CALL PROGRAM statement. Details for the CALL and CALL PROGRAM statements begin on pages 299 and 303 respectively. Also see related sections in this document: PROCEDURE DIVISION USING phrase on page 56 and EXIT PROGRAM statement on page 357.

CALL	CALL PROGRAM
A "PERFORM" equivalent.	A "chain" equivalent.
Returns to the calling program.	Does not return to the calling program (except for # or ## system calls, which may perform a task and return).
Called program runs in the same run unit as the calling program.	Called program begins a new run unit or performs a system call.
Cannot pass switches to the called program.	Can pass switches to the called program.
The calling program is left in the current state except that contents of items in the USING phrase may have been altered by the called program.	All files in the calling program are closed before control is passed to the called program.
The state of the called program remains unchanged from its state when last exited unless it has the INITIAL attribute in which case it will have its initial state when next called.	N/A - Called program is always in its initial state.
CANCEL logically removes called program from the run unit so it will be in its initial state next time it is called.	CANCEL is not applicable for a program called with CALL PROGRAM.
EXIT PROGRAM marks the logical end of a called program.	EXIT PROGRAM has no effect in a called program.

TABLE 23. CALL and CALL PROGRAM Compared

E.6. CANCEL

E.6.1 Function

The CANCEL statement ensures that the next time the referenced program is called it will be in its initial state.

E.6.2 General Format

CANCEL { *identifier* / *literal* }...

E.6.3 Syntax Rules

(1) *Literal* must be a nonnumeric literal.

(2) *Identifier* must be defined as an alphanumeric data item such that its value can be a program name.

E.6.4 General Rules

(1) *Literal* or the content of the data item referenced by *identifier* identifies the program to be canceled.

(2) Subsequent to the execution of an explicit or implicit CANCEL statement, the program referred to therein ceases to have any logical relationship to the run unit in which the CANCEL statement appears. If the program referenced by a successfully executed explicit or implicit CANCEL statement in a run unit is subsequently called in that run unit, that program is in its initial state.

(3) A program named in a CANCEL statement in another program must be callable by that other program.

(4) A program named in the CANCEL statement must not refer directly or indirectly to any program that has been called and has not yet executed an EXIT PROGRAM statement.

(5) A logical relationship to a canceled program is established only by execution of a subsequent CALL statement naming that program.

(6) A called program is canceled either by being referred to as the operand of a CANCEL statement, by the termination of the run unit of which the program is a member (STOP RUN, CALL PROGRAM, interrupt), or by execution of an EXIT PROGRAM statement in a called program that possesses the initial attribute.

(7) No action is taken when an explicit or implicit CANCEL statement is executed naming a program that has not been called in this run unit or has been called and is at present canceled. Control is transferred to the next executable statement following the explicit CANCEL statement.

(8) During execution of an explicit or implicit CANCEL statement, an implicit CLOSE statement without any optional phrases is executed for each file in the open mode that is associated with an internal file connector in the program named in the explicit CANCEL statement. Any USE procedures associated with any of these files are not executed.

(9) The contents of data items in external data records described by a program are not changed when that program is cancelled.

(10) The CANCEL statement does not close external files, even those open in the subprogram. You must explicitly close external files.

E.7. CLOSE

E.7.1 Function

The CLOSE statement terminates the processing of files with optional lock.

E.7.2 General Format

For sequential files: (***ANSI 74*** and ***ANSI 85***)

```
                                                                  ]
                                                                  ]
CLOSE { file-name [ { REEL } [ FOR REMOVAL ] ]                    }...
                    { UNIT }                                      ]
                    WITH { NO REWIND }                            ]
                         { LOCK      }                            ]
```

For sequential files: (***VXCOBOL***)

```
                                                                  ]
                                                                  ]
                                                                  ]
                                                                  ]
CLOSE { file-name [ { REEL } [ FOR REMOVAL ] ]                    }...
                    { UNIT } [ WITH NO REWIND ]                   ]
                    WITH { NO REWIND }                            ]
                         { LOCK      }                            ]
                         { RELEASE   }                            ]
```

For relative, indexed, and INFOS files:

CLOSE { *file-name* [WITH LOCK] }...

E.7.3 Syntax Rules

(1) The files referenced in the CLOSE statement need not all have the same organization or access.

E.7.4 General Rules

(1) A CLOSE statement may only be executed for a file in an open mode.

(2) If the LOCK phrase is specified for a file, the file cannot be reopened by the program that performed the CLOSE WITH LOCK.

(3) The execution of the CLOSE statement causes the value of the I-O status associated with *file-name* to be updated.

(4) If an optional input file is not present, no end-of-file processing is performed for the file and the file position indicator is unchanged.

(5) Following the successful execution of a CLOSE statement the record area associated with a *file-name* is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

(6) Following the successful execution of a CLOSE statement the file is removed from the open mode, and the file is no longer associated with the file connector.

(7) If more than one *file-name* is specified in a CLOSE statement, the result of executing this CLOSE statement is the same as if a separate CLOSE statement had been written for each *file-name* in the same order as specified in the CLOSE statement.

(8) If the CLOSE is unsuccessful, a USE procedure, if specified, is executed.

(9) The NO REWIND, REEL/UNIT, RELEASE, and FOR REMOVAL clauses are for documentation purposes only.

(10) When the CLOSE statement is executed for a file, any modified file buffers (including any that were modified by other users) are flushed to disk or other device by the **ICOBOL** system. For indexed and relative files, the ICISAM reliability flags in the file are cleared.

(11) On a CLOSE of a character device, if a timeout value was not specified on the OPEN, the CLOSE will try forever. If a timeout had been specified, the CLOSE will complete in that time, the line will be closed, and the buffer reset.

(12) An implicit CLOSE is executed for all open files within a program whenever it terminates.

E.8. COMMIT (***ISQL***)

E.8.1 Function

The COMMIT statement allows the program to commit an SQL database connection or connections..

E.8.2 General Format

```
COMMIT [ ALL ]
    [ ON SQLERROR imperative-statement-1 ]
    [ NOT ON SQLERROR imperative-statement-2 ]
    [ END-COMMIT ]
```

E.8.3 Syntax Rules

E.8.4 General Rules

(1) The ALL phrase specifies that all connections in the run unit will be committed. (if there are any). If not specified only the current connection is committed.

(2) Upon completion of the COMMIT statement, the following occurs in the order specified:

a. The value of the SQLSTATE data item is updated with the status of the operation.

b. If the value of the SQLSTATE class field is “00" or “01", the statement is successful. Control is transferred to the end of the COMMIT statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the COMMIT statement.

c. If the value of the SQLSTATE class field is not “00" or “01", the statement is unsuccessful. The statement container is deallocated and no statement container of the specified name will exist in the current program. Control is transferred to the end of the COMMIT statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the COMMIT statement.

(3) The END-COMMIT phrase delimits the scope of the COMMIT statement.

(4) More on SQLSTATE can be found on page 133.

E.9. COMPUTE

E.9.1 Function

The COMPUTE statement assigns to one or more data items the value of an arithmetic expression.

E.9.2 General Format

```
COMPUTE { identifier-1 [ ROUNDED ] }... = arithmetic-expression
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-COMPUTE ]
```

E.9.3 Syntax Rules

(1) *Identifier-1* must reference either an elementary numeric item or an elementary numeric edited item.

(2) (***ISQL***) *Identifier-1* may also be a date-time or interval elementary data item subject to the general rules for permissible combinations of operands.

E.9.4 General Rules

(1) An *arithmetic-expression* consisting of a single identifier or literal provides a method of setting the value of the data item reference by *identifier-1* equal to the literal or the value of the data item reference by the single identifier.

(2) If more than one identifier is specified for the result of the operation, the value of the arithmetic expression is developed, and then is stored as the new value of each of the data items referenced by *identifier-1*.

(3) The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE.

(4) (***ISQL***) The COMPUTE statement can be used with date-time and interval operands. The category of the data-item referenced by *identifier-1* must match the result category of *arithmetic-expression*. The rules for an arithmetic expression involving date-time and interval items are covered under Arithmetic Expressions, beginning on page 230.

(5) Additional rules and explanations to this statement are given under the appropriate paragraphs. (See Arithmetic Expressions, page 230; Scope of Statements, page 253; The ROUNDED phrase, page 245; The ON SIZE ERROR Phrase, page 246; The Arithmetic Statements, page 249; Overlapping Operands, page 249; and Multiple Results in Arithmetic Statements, page 249.)

E.10. CONNECT (***ISQL***)

E.10.1 Function

The CONNECT statement allows the program to establish a connection to an SQL database. Other SQL statements that occur in the program operate in the context of the currently active connection.

E.10.2 General Format

```
                                          DEFAULT
CONNECT TO { { identifier-1 } [ AS { identifier-2 } ] [ USER { identifier-3 } [ identifier-4 ] ] }
           { { literal-1 }        { literal-2 }              { literal-3 }   [ literal-4 ]   }
    [ ON SQLERROR imperative-statement-1 ]
    [ NOT ON SQLERROR imperative-statement-2 ]
    [ END-CONNECT ]
```

E.10.3 Syntax Rules

(1) *Literal-1*, *literal-2*, *literal-3*, and *literal-4* must specify a nonnumeric literal and may not specify a figurative constant.

(2) *Identifier-1*, *identifier-2*, *identifier-3*, and *identifier-4* must specify an alphanumeric data item.

(3) *Literal-2* or the value represented by *identifier-2* may not specify the value “default” (case-insensitive), which is reserved as the name for the connection established by specifying the DEFAULT phrase.

E.10.4 General Rules

(1) The DEFAULT phrase specifies that a system default value is to be used for the connection string, user name, and password. This default value is selected from the environment variables ICSQLDSN, ICSQLUID, and ICSQLPWD. If ICSQLDSN is not present, a data-set name of “default” is used. If ICSQLUID is not present, the current login name is used (the same value returned by ACCEPT FROM USER NAME). If ICSQLPWD is not present, a null string is used. The connection will have the name “default”.

(2) *Literal-1* or the content of the data item represented by *identifier-1* specifies the connection string that supplies the information necessary to connect to the database. Usually it specifies a data-set name (DSN).

(3) *Literal-2* or the content of the data item represented by *identifier-2* in the AS phrase specifies a name for the connection. The name can be used to identify the connection in a DISCONNECT or SET CONNECTION statement. The value is not case-sensitive. If the AS phrase is not supplied, the content of the connection string is used as the connection name.

(4) *Literal-3* or the content of the data item represented by *identifier-3* in the USER phrase specifies a user name for the connection. If the USER phrase is not specified, the system will use the current user login name.

(5) *Literal-4* or the content of the data item represented by *identifier-4* in the USER phrase specifies a password for the connection. If this optional field is not specified, the system will use a null string.

(6) It is an error if the run unit already has a connection with the same name, which includes the name “default” for a connection made by using the DEFAULT phrase.

(7) Connections are kept on a run unit basis, i.e., the scope of the connection name is the entire run unit, not the program containing the CONNECT statement.

(8) All connections in a run unit are implicitly disconnected when the run unit terminates, in a manner equivalent to the execution of a DISCONNECT ALL statement.

.

(9) Upon a successful connection, the currently active connection (if any) is made dormant, and the new connection is made the currently active connection.

(10) Upon completion of the CONNECT statement, the following occurs in the order specified:

a. The value of the SQLSTATE data item is updated with the status of the operation.

b. If the value of the SQLSTATE class field is "00" or "01", the statement is successful. Control is transferred to the end of the CONNECT statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the CONNECT statement.

c. If the value of the SQLSTATE class field is not "00" or "01", the statement is unsuccessful. The statement container is deallocated and no statement container of the specified name will exist in the current program. Control is transferred to the end of the CONNECT statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the CONNECT statement.

(11) The END-CONNECT phrase delimits the scope of the CONNECT statement.

(12) More on SQLSTATE can be found on page 133.

(13) CONNECT takes a DSN by default.

Under Windows, this DSN is defined in the ODBC Administrator in the User DSN or System DSN panels.

Under UNIX, this DSN is defined in the .odbc.ini file in the user's home directory for User DSN and the odbc.ini file for System DSN files. More on ODBC under UNIX can be found in the unixODBC documentation.

At this time only a UserDSN or a SystemDSN can be specified. FileDSN's are not supported.

(14) To help debug ODBC connections enable Tracing to the ODBC Driver.

Under Windows, this is done in the ODBC Administrator under the Tracing panel where the actual log file and Starting and Stopping tracing is performed.

Under UNIX, this is done in the odbcinst.ini file by adding:

```
[ODBC]
Trace = Yes
Trace File = filename
```

Generally tracing should not be enabled as it is VERY expensive in cpu and disk resources.

(15) A sample program that provides a Screen Interface to the ***ISQL*** statements is provided on the ftp server (ftp.icobol.com) in the examples subdirectory of icobol as as isqltest.sr

(16) In addition, the ICODBC Driver can be used to test with ICISAM files if needed.

(17) Under Windows, the odbc32.dll is loaded to allow the ***ISQL*** statements to communicate with ODBC. Under UNIX, the libodbc.so is loaded to allow the ISQL statements to communicate with the unixODBC module.

E.11. CONTINUE

E.11.1 Function

The CONTINUE statement is a no operation (or "no op") statement. It indicates that no executable statement is present.

E.11.2 General Format

CONTINUE

E.11.3 Syntax Rules

(1) The CONTINUE statement may be used anywhere a conditional statement or an imperative-statement may be used.

E.11.4 General Rules

(1) The CONTINUE statement has no effect on the execution of the program.

E.12. DEALLOCATE (*ISQL*)

E.12.1 Function

The DEALLOCATE statement allows the program to deallocate a statement container that was allocated by a PREPARE statement once it is no longer needed.

E.12.2 General Format

```
DEALLOCATE PREPARE { identifier-1 }
                   { literal-1    }
    [ ON SQLERROR imperative-statement-1 ]
    [ NOT ON SQLERROR imperative-statement-2 ]
    [ END-CONNECT ]
```

E.12.3 Syntax Rules

(1) *Literal-1* must specify a nonnumeric literal and may not specify a figurative constant.

(2) *Identifier-1* must specify an alphanumeric data item.

(3) *Literal-1* or the value represented by *identifier-1* may not exceed 30 characters in length.

E.12.4 General Rules

(1) *Literal-1* or the content of the data item represented by *identifier-1* specifies the name of a statement container to be deallocated in the current program.

(2) If a statement container with the specified name is not found in the current program, the SQLSTATE class field is set to “01".

(3) If a statement container with the specified name is found in the current program, it is deallocated and a statement container with the specified name will no longer exist in the current program, the SQLSTATE class field is set to “00".

(4) Upon completion of the DEALLOCATE statement, the following occurs in the order specified:

a. The value of the SQLSTATE data item is updated with the status of the operation.

b. If the value of the SQLSTATE class field is “00" or “01", the statement is successful. Control is transferred to the end of the DEALLOCATE statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the DEALLOCATE statement.

c. If the value of the SQLSTATE class field is not “00" or “01", the statement is unsuccessful. Control is transferred to the end of the DEALLOCATE statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the DEALLOCATE statement.

(5) The END-DEALLOCATE phrase delimits the scope of the DEALLOCATE statement.

E.13. DEFINE SUB-INDEX (*VXCOBOL*)

E.13.1 Function

The DEFINE SUB-INDEX statement creates a subindex in an INFOS file and associates with it a specified index entry in that file.

E.13.2 General Format

```
DEFINE SUB-INDEX file-name [ { FIX    } POSITION ] [ NEXT          ]
                           [ { RETAIN }          ] [ FORWARD       ]
                                                   [ BACKWARD      ]
                                                   [ UP            ]
                                                   [ DOWN          ]
                                                   [ UP FORWARD    ]
                                                   [ UP BACKWARD   ]
                                                   [ DOWN FORWARD  ]
                                                   [ STATIC        ]

    [ { KEY IS   } { identifier-1 [ APPROXIMATE ] }... ]
    [ { KEYS ARE } {              [ GENERIC     ] }    ]
    FROM identifier-2
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-DEFINE ]
```

```
DEFINE SUB-INDEX file-name [ { FIX    } POSITION ] [ NEXT          ]
                           [ { RETAIN }          ] [ FORWARD       ]
                                                   [ BACKWARD      ]
                                                   [ UP            ]
                                                   [ DOWN          ]
                                                   [ UP FORWARD    ]
                                                   [ UP BACKWARD   ]
                                                   [ DOWN FORWARD  ]
                                                   [ STATIC        ]

    [ { KEY IS   } { identifier-1 [ APPROXIMATE ] }... ]
    [ { KEYS ARE } {              [ GENERIC     ] }    ]
    [ INDEX NODE SIZE IS integer-1 ]
    [ ALLOW DUPLICATES ]
    [ ALLOW SUB-INDEX ]
d   [ KEY COMPRESSION ]
    [ MAXIMUM KEY LENGTH IS integer-2 ]
    [ PARTIAL RECORD LENGTH IS integer-3 ]
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-DEFINE ]
```

E.13.3 Syntax Rules

(1) *File-name* is a filename that specifies an INFOS file opened for OUTPUT or I/O and selected for ALLOW SUB-INDEX.

(2) *Identifier-1* is an alphanumeric data item that specifies a record key associated with file-name.

(3) *Identifier-2* is an alphanumeric data item that contains data in the form of an AOS INFOS (16-bit) sub-index definition packet and that is defined in Working-Storage.

(4) *Integer-3* is an integer or integer literal data item that specifies the maximum partial record length for the sub-index.

(5) *Integer-2* is an integer or integer literal data item that specifies the maximum key length for a sub-index.

(6) *Integer-1* is an integer or integer data item that specifies the size of a sub-index node.

E.13.4 General Rules

(1) When using the FROM option, the packet specified should be the AOS INFOS packet, not the 32-bit INFOS II packet. This packet is 16 bytes long with the following format:

```
01 PACKET.
    03 FILLER                 PIC XX.
    03 NODE-SIZE              PIC 9(4) COMP.
    03 FILLER                 PIC X.
    03 MAX-KEYLEN             PIC 9(2) COMP.
    03 FILLER                 PIC X.
    03 PARTIAL-REC-LEN        PIC 9(2) COMP.
    03 FILLER                 PIC XX.
    03 FLAGS                  PIC 9(4) COMP.
    03 FILLER                 PIC X(4).
```

FLAGS values are: 2048 allow duplicates, 16384 Disallow sub-index.

(2) The location of the entry defined is determined according to that specified in the position phrase, the relative option phrase, and/or the KEY series phrase. The specification can be implicit if the program uses the defaults or explicit if the KEY or path is specified fully.

(3) FIX POSITION causes the record pointer to move from the current position to the position specified in this statement. RETAIN position causes the record position to remain at the position it was on before the execution of this statement. RETAIN is the default.

(4) The relative motion option without the KEY series phrase allows access to the index file relative to that file's current record position.

(5) Using the KEY series phrase without the relative motion option causes the key path specified to begin with the top index in the hierarchy and follow a downward motion.

(6) If the KEY series phrase is specified, each key, identifier-1, must be declared in the SELECT statement for file-name. If the relative motion option and KEY series phrase at both specified only UP, DOWN, and STATIC are allowed. The relative motion option is processed first, and the key path is used. If both are omitted, STATIC is the default.

(7) Transfer of control following the successful or unsuccessful execution of the DEFINE SUB-INDEX operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the DEFINE SUB-INDEX statement.

(8) The PARTIAL RECORD clause must be specified to allow partial records to be stored in the sub-index. For INFOS II, the length of partial records in the sub-index is established. For U/FOS, any non-zero length says to allow partial records, the specified length is disregarded.

(9) The ALLOW SUB-INDEX clause must be specified to allow subordinate sub-indexing for the specified sub-index.

(10) The DUPLICATES clause must be specified to allow for the creation of duplicate keys for the sub-index being created.

(11) If not specified, the KEY LENGTH defaults to 255.

(12) If not specified, the INDEX NODE SIZE defaults to the system default. (This value is ignored by U/FOS.)

E. 14. DELETE

E.14.1 Function

The DELETE statement logically removes a record from a mass storage file for relative, indexed, and INFOS files.

E.14.2 General Format (***ANSI 74*** and ***ANSI 85***)

```
DELETE file-name RECORD [ LOGICAL  ]
                        [ PHYSICAL ]
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-DELETE ]
```

E.14.3 General Format (***VXCOBOL***)

Relative:

```
DELETE file-name RECORD
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-DELETE ]
```

Indexed:

```
DELETE file-name RECORD [ PHYSICAL                          ]
                        [ LOGICAL { LOCAL        }          ]
                                  { GLOBAL       }          ]
                                  { LOCAL GLOBAL }          ]
    [ KEY IS identifier-1 ]
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-DELETE ]
```

INFOS:

```
DELETE file-name [ NEXT          ]
                 [ FORWARD       ]
                 [ BACKWARD      ]
                 [ UP            ]
                 [ DOWN          ]
                 [ UP FORWARD    ]
                 [ UP BACKWARD   ]
                 [ DOWN FORWARD  ]
                 [ STATIC        ]

                 [ PHYSICAL                          ]
                 [ LOGICAL { LOCAL        } RECORD   ]
                           { GLOBAL       }          ]
                           { LOCAL GLOBAL }          ]

    [ { KEY IS  } { identifier [ APPROXIMATE ] }... ]
      { KEYS ARE }             [ GENERIC     ]
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-DELETE ]
```

E.14.4 Syntax Rules

(1) The INVALID KEY and the NOT INVALID KEY phrases must not be specified for a DELETE statement which references a file which is in sequential access mode.

(2) The INVALID KEY phrase must be specified for a DELETE statement which references a file which is not in sequential access mode and for which an applicable USE AFTER STANDARD EXCEPTION procedure is not specified.

(3) The PHYSICAL designation applies to version 7 or greater ICISAM files.

For ***VXCOBOL***.

(4) Identifier-1 must be the RECORD KEY as defined in the SELECT.

(5) The key series specifier may not be present for files in SEQUENTIAL ACCESS mode.

E.14.5 General Rules (***ANSI 74*** and ***ANSI 85***)

(1) The file referenced by *file-name* must be an indexed or relative file and must be open in the I-O mode at the time of the execution of this statement.

(2) For files in the sequential access mode, the last input-output statement executed for *file-name* prior to the execution of the DELETE statement must have been a successfully executed READ statement. The file system removes from the file the record that was accessed by that READ statement.

(3) For a relative file in random or dynamic access mode, the file system removes from the file that record identified by the content of the relative key data item associated with *file-name*. If the file does not contain the record specified by the key, the invalid key condition exists.

(4) For an indexed file in random or dynamic access mode, the file system removes from the file the record identified by the content of the primary record key data item associated with *file-name*. If the file does not contain the record specified by the key, the invalid key condition exists.

(5) After the successful execution of a DELETE statement, the identified record has been removed from the file and can no longer be accessed, although the record may be restored by executing the UNDELETE statement if the removal was a logical deletion.

(6) The execution of a DELETE statement does not affect the content of the record area.

(7) The file position indicator is not affected by the execution of a DELETE statement.

(8) The execution of the DELETE statement causes the value of the I-O status associated with *file-name* to be updated.

(9) Transfer of control following the successful or unsuccessful execution of the DELETE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the DELETE statement.

(10) The END-DELETE phrase delimits the scope of the DELETE statement.

(11) If LOGICAL is specified, the record identified by the RECORD KEY or RELATIVE KEY is marked as being deleted in the file. It is not physically removed, but will not be accessible unless it is subsequently undeleted.

If PHYSICAL is specified, the space in the data file used by the record identified by the RECORD KEY or RELATIVE KEY is made available for reuse. It is no longer accessible to the program. Its space will be reused when needed to add another record to the file.

If neither LOGICAL nor PHYSICAL is specified, the delete will be either logical or physical based on the following:

a. For version 5 or version 6 indexed or relative files, deletes are always logical.

b. For version 7 indexed or relative files, the type of delete performed will be based on the status of the file's "delete-is-physical" attribute. This attribute bit is set at file creation time and is a permanent attribute of the file. (It is specified in a COBOL program using the DELETE IS clause of the file description entry (SELECT)).

E.14.6 General Rules (***VXCOBOL***)

(1) The file referenced by *file-name* must be a relative, indexed, or INFOS file and must be open in the I-O mode at the time of the execution of this statement.

(2) For files in the sequential access mode, the last input-output statement executed for *file-name* prior to the execution of the DELETE statement must have been a successfully executed READ statement. The file system removes from the file the record that was accessed by that READ statement.

(3) The execution of a DELETE statement does not affect the content of the record area.

(4) The file position indicator is not affected by the execution of a DELETE statement, for indexed and relative files.

(5) The execution of the DELETE statement causes the value of the I-O status associated with file-name to be updated.

(6) Transfer of control following the successful or unsuccessful execution of the DELETE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the DELETE statement.

(7) The END-DELETE phrase delimits the scope of the DELETE statement.

For relative files:

(8) For a relative file in random or dynamic access mode, the file system removes the record identified by the content of the relative-key data-item associated with file-name. If the files does not contain the record specified by the key, the invalid key condition exists.

(9) Records in relative files are removed on the basis of the “delete-is-physical” attribute set in the file's header. Files created by ***VXCOBOL*** programs will normally have this bit set for purging records (physical deletes).

(10) After the successful execution of a DELETE statement, the identified record has been removed from the file and can no longer be accessed or restored.

For indexed files:

(11) For an indexed file in random or dynamic access mode, the file system logically or physically removes from the file the record identified by the content of the primary key data-item associated with file-name. If the files does not contain the record specified by the key, the invalid key condition exists.

(12) If PHYSICAL is specified, the data record is purged from the file. After the successful execution of a DELETE statement with the PHYSICAL clause, the identified record has been removed from the file and can no longer be accessed or restored.

(13) If LOGICAL GLOBAL is specified, the data record is logically deleted from the file. After the successful execution of a DELETE statement with the LOGICAL GLOBAL clause, the identified record may still be accessed. The record may be restored by executing the UNDELETE statement.

(14) If LOGICAL LOCAL is specified, it is ignored.

(15) If LOGICAL LOCAL GLOBAL is specified, it is equivalent to LOGICAL GLOBAL.

(16) If no type of deletion is specified, PHYSICAL is the default.

For INFOS files:

(17) The occurrence number is used.

(18) FEEDBACK is not used and is not updated.

(19) KEY LENGTH is unaffected.

(20) The record to DELETE is determined according to what is specified in the relative option phrase and/or the KEY series phrase. The specification can be implicit if the program uses the defaults or explicit if the KEY or path is fully specified.

(21) The relative motion option without the KEY series phrase allows access to the index file relative to that file's current record position.

(22) Using the KEY series phrase without the relative motion option cause the key path specified to begin with the top index in the hierarchy and follow a downward motion.

(23) If the KEY series phrase is specified, each key, identifier-1, must be declared in the SELECT statement for file-name. If the relative motion option and KEY series phrase at both specified only UP, DOWN, and STATIC are allowed. The relative motion option is processed first and the key path is used.

(24) If both the relative option and the KEY series phrase are omitted the file is accessed sequentially if the file access mode is sequential. If the access mode is not sequential the first key named in the SELECT clause is used.

(25) If LOGICAL LOCAL is specified, the key (and any partial record associated with that key) is logically deleted. Whenever the record or key is accessed through this index a FILE STATUS 96 will be returned.

(26) If LOGICAL GLOBAL is specified, the data record is logically deleted. Whenever the record is accessed through any index a FILE STATUS 96 will be returned. The index entry including the partial record and any subindex can still be accessed without receiving a FILE STATUS 96.

(27) If LOGICAL LOCAL GLOBAL is specified, the key (and any partial data record associated with that key) and the data record is logically deleted.

(28) If PHYSICAL is specified, the key (and any partial data record associated with that key) is deleted and the data record's use count is decremented. If the data record's use count is decremented to zero, then the data record itself is deleted such that it is no longer in the file and there is no inversion in the file pointing to it.

(29) If no type of deletion is specified, PHYSICAL is the default.

(30) If you want to know whether a record has been deleted, use the RETRIEVE statement.

(31) A FILE STATUS 02 is returned when a successful physical deletion of a record with a duplicate key is performed.

(32) A DELETE statement does not change the current position of the record pointer unless it is a PHYSICAL deletion and the pointer's current position is at the deleted record. If this is the case, the record pointer points to the record immediately before the deleted record.

(33) If DUPLICATES was specified in the SELECT clause then the occurrence number should be set to the desired value for the key that should be deleted.

E.15. DELETE FILE

E.15.1 Function

The DELETE FILE statement physically removes a file from the file system. DELETE FILE is an extension to ANSI COBOL. For ***VXCOBOL***, it is equivalent to EXPUNGE.

To see how **ICOBOL** processes the filename see the External Filename description in the Developer's Guide section on page 751.

E.15.2 General Format

DELETE FILE { *file-name* }...

E.15.3 General Rules

(1) The file referenced by *file-name* must be a disk file, you must have appropriate permissions, and the file must not be open anywhere in the **ICOBOL** system at the time of the execution of this statement. If the file does not exist, no error is given.

(2) For a relative, indexed, or INFOS file, all parts of that file are removed from the file system.

(3) For ***VXCOBOL***: for an INFOS II file, the indexed file and the database file specified in the SELECT statement are deleted. If the name of the database file was not specified with an ASSIGN DATA clause, a .DB file with the same name as that of the indexed file is deleted. For a U/FOS file, the database specified in the SELECT is deleted, i.e., name.udb.

(4) After the successful execution of a DELETE FILE statement, the identified file has been physically removed from the file system and can no longer be accessed.

(5) The execution of the DELETE FILE statement causes the value of the I-O status associated with *file-name* to be updated.

(6) For systems supporting UNIX symbolic links, DELETE FILE will delete the symbolic link, not the resolution file.

(7) On UNIX systems, files cannot be individually delete-protected. To make a file delete-protected on UNIX, you must remove write (w) permission to the directory in which the file resides. If a directory has no write access, you cannot create, modify or delete files in that directory. On Windows systems, the read-only attribute will protect the file from deletion.

(8) For **ANSI 74 and ANSI 85**, If the specified file is a sequential file, **ICOBOL** will scan the Printer Control file and if there is an entry there that points to the file being deleted, the entry in the Printer Control file will be removed.

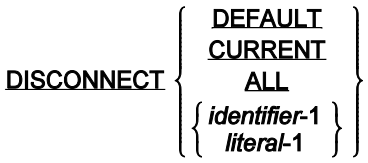
(9) For ***VXCOBOL***, if *file-name* is a sort/merge file, it is ignored.

E.16. DISCONNECT (*ISQL*)

E.16.1 Function

The DISCONNECT statement allows the program to disconnect from an SQL database connection.

E.16.2 General Format



[ON SQLERROR *imperative-statement-1*]
[NOT ON SQLERROR *imperative-statement-2*]
[END-DISCONNECT]

E.16.3 Syntax Rules

(1) *Literal-1* must specify a nonnumeric literal and may not specify a figurative constant.

(2) *Identifier-1* must specify an alphanumeric data item.

E.16.4 General Rules

(1) The DEFAULT phrase specifies that the default connection (which has the name “default”) is to be disconnected. It is an error if there is no default connection either active or dormant. If the default connection is the current connection, it is replaced as the current connection by the most recently used previous connection.

(2) The CURRENT phrase specifies that the currently active connection is to be disconnected. The most recently used previous connect becomes the current connection. It is an error if there is no current connection.

(3) The ALL phrase specifies that all connections in the run unit will be disconnected (if there are any).

(4) The value of *literal-1* or the content of the data item represented by *identifier-1* specifies a specific, named connection. If the value “default” is specified, it is the same as having specified the DEFAULT phrase. If the specified connection is the current connection, it is replaced as the current connection by the most recently used previous connection.

(5) Connections are kept on a run unit basis, i.e., the scope of the connection name is the entire run unit, not just the program containing the DISCONNECT statement. If a specified connection does not exist, it is an error and SQLSTATE will be set to “08003", which is “Connection does not exist”.

(6) All connections in a run unit are implicitly disconnected when the run unit terminates in a manner equivalent to the execution of a DISCONNECT ALL statement.

(7) Any statement containers associated with a connection that is being disconnected are implicitly deallocated before the connection is disconnected.

.

(8) Upon completion of the DISCONNECT statement, the following occurs in the order specified:

a. The value of the SQLSTATE data item is updated with the status of the operation.

b. If the value of the SQLSTATE class field is "00" or "01", the statement is successful. Control is transferred to the end of the DISCONNECT statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the DISCONNECT statement.

c. If the value of the SQLSTATE class field is not "00" or "01", the statement is unsuccessful. The statement container is deallocated and no statement container of the specified name will exist in the current program. Control is transferred to the end of the DISCONNECT statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the DISCONNECT statement.

(9) The END-DISCONNECT phrase delimits the scope of the DISCONNECT statement.

(10) More on SQLSTATE can be found on page 133.

E.17. DISPLAY

E.17.1 Function

The DISPLAY statement causes low volume data to be transferred to the console. Screens are an extension to ANSI COBOL.

E.17.2 General Format

Format 1:

```
DISPLAY { identifier-1 | literal-1 }...        [ UPON mnemonic-name ] [ WITH NO ADVANCING ]
    [ END-DISPLAY ]
```

Format 2:

```
DISPLAY { screen-name [ AT { LINE { identifier-2 | literal-2 } [ { COLUMN | COL } { identifier-3 | literal-3 } ]
                           { { COLUMN | COL } { identifier-3 | literal-3 } [ LINE { identifier-2 | literal-2 } ] } } ]
    [ END-DISPLAY ]
```

Format 3 (ANSI 74 and ANSI 85):

```
DISPLAY { { identifier-1 | literal-1 } [ UNIT { identifier-4 | literal-4 } ]        display-clause... }...
    [ END-DISPLAY ]
```

where *display-clause* is one of the following:

```
{ BACKGROUND-COLOR | BACKGROUND } IS { identifier-5 | literal-5 | color-name-1 }

{ FOREGROUND-COLOR | FOREGROUND } IS { identifier-8 | literal-8 | color-name-2 }

{ BELL | BEEP }

BLINK

{ COLUMN | COL | POSITION } { identifier-6 | literal-6 }

CONTROL { identifier-7 | literal-7 }

CONVERT
```

```
ERASE [ EOL    ]
      [ EOS    ]
      [ LINE   ]
      [ SCREEN ]

{ HIGH       }
{ HIGHLIGHT  }
{ LOW        }
{ LOWLIGHT   }
{ BOLD       }
{ BRIGHT     }
{ DIM        }

LINE { identifier-9 }
     { literal-9    }

{ REVERSE       }
{ REVERSED      }
{ REVERSE-VIDEO }

SIZE { identifier-10 }
     { literal-10    }

{ UNDERLINE  }
{ UNDERLINED }
```

E.17.3 Syntax Rules

(1) In Format 1, you cannot use the figurative constant ALL with a DISPLAY statement.

(2) In Format 2, *identifier-2*, *identifier-3*, *literal-2*, and *literal-3* must be unsigned integers.

(3) *Screen-name* may not be subscripted.

(4) The word COL is an abbreviation for the word COLUMN.

(5) END-DISPLAY is supported only for ***ANSI 74*** and ***ANSI 85***. It is an extension to standard COBOL.

(6) *Mnemonic-name* is associated with a hardware device in the SPECIAL-NAMES paragraph.

(7) In Format 3, *identifier-4, identifier-5*, *identifier-6*, *identifier-8*, *identifier-9*, *identifier-10*, *literal-4*, *literal-5*, *literal-6*, *literal-8*, *literal-9*, and *literal-10* must be unsigned elementary integer items. *Identifier-7* must be a nonnumeric data-item and *literal-7* must be a nonnumeric literal.

(8) *Color-name-1* and *color-name-2* represent one of the predefined color names: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, or WHITE.

(9) In Format 3, the word POSITION is a synonym for COLUMN and the word BEEP is a synonym for BELL.

E.17.4 General Rules

Format 1: (non-screen display)

(1) The DISPLAY statement causes the content of each operand to be transferred to the console device in the order listed.

(2) If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

(3) If the device is capable of receiving data of the same size as the data item being transferred, then the data item is transferred.

(4) If a device is not capable of receiving data of the same size as the data item being transferred, then one of the following applies:

a. If the size of the data item being transferred exceeds the size of the data that the device is capable of receiving in a single transfer, the data beginning with the left-most character is stored aligned to the left in the receiving device, and the remaining data is then transferred according to General Rules 4 and 5 until all the data has been transferred.

b. If the size of the data item that the device is capable of receiving exceeds the size of the data being transferred, the transferred data is stored aligned to the left in the receiving device.

(5) When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered without modifying the positioning of the hardware device between the successive operands.

(6) If the WITH NO ADVANCING phrase is specified, then the positioning of the device will not be reset to the next line or changed in any other manner following the display of the last operand. If the device is capable of positioning to a specific character position, it will remain positioned at the character position immediately following the last character of the last operand displayed. If the device is not capable of positioning to a specific character position, only the vertical position, if applicable, is affected. This may cause overprinting if the device supports overprinting.

(7) If the WITH NO ADVANCING phrase is **NOT** specified, then after the last operand has been transferred to the device, the positioning of the device will be reset to the left-most position of the next line of the device.

(8) If vertical positioning is not applicable on the device, the operating system will ignore the vertical positioning specified or implied.

(9) For ***VXCOBOL***: If the data to be transferred has USAGE COMPUTATIONAL or USAGE COMPUTATIONAL-3, **ICOBOL** moves the data to a temporary data-item defined as USAGE DISPLAY, SIGN LEADING SEPARATE with the same PICTURE. The temporary item is then transferred.

(10) For **ANSI 74 and ANSI 85**, integer or numeric functions are displayed as if they were defined with USAGE DISPLAY, SIGN LEADING SEPARATE.

(11) The UPON clause is for documentation only except in the one case where *mnemonic-name* refers to "@AUDIT". If it refers to "@AUDIT" and auditing is enabled, then the DISPLAY will be sent to the audit log. If auditing is not enabled, nothing is done. Thus, in the procedure division a:

```
DISPLAY foo1 foo2 UPON mnemonic-1.
```

Will send the data in foo1 and foo2 to the audit log.

This facility is especially useful when debugging ThinClients.

(If this statement is executed with a pre-3.13 runtime the DISPLAY will come to the screen.)

Format 2: (screen display)

(12) Format 2 assumes that the device is capable of random positioning.

(13) DISPLAY *screen-name* is equivalent to DISPLAY *screen-name* AT LINE 0 COLUMN 0.

(14) If the LINE or COLUMN variable in the SCREEN SECTION has a value of zero (0), **ICOBOL** treats the value as one (1).

(15) **Variable Origin**: The LINE phrase and COLUMN phrase in DISPLAY and ACCEPT statements allow the entire screen description referenced by *screen-name* to be moved to a different starting position on the user's display device. This capability is called *variable origin*. All screen descriptions assume that the origin is at line 1 and column 1 on the user's display device. The value specified in the DISPLAY or ACCEPT's LINE phrase, if present, is treated as a relative offset to be added to all line positions in the screen. Similarly, the value of the COLUMN phrase, if specified, is treated as a relative offset to be added to all column positions in the screen. If any line or column position becomes larger than that supported by the current screen, the screen will wrap at its limits, and the new (wrapped) values will in turn be offset again by the variable origin.

For example, consider the code fragments:

```
01 ANY-CHANGE-SCREEN.
    05 LINE 23 COL 60  "ANY CHANGE?".
    05 LINE 23 COL 75  PIC X TO ANY-CHANGE-ANSWER.

ANY-CHANGE-1.
    DISPLAY ANY-CHANGE-SCREEN.
    ACCEPT ANY-CHANGE-SCREEN.

ANY-CHANGE-2.
    DISPLAY ANY-CHANGE-SCREEN AT LINE 5 COLUMN 30.
    ACCEPT ANY-CHANGE-SCREEN AT LINE 5 COLUMN 30.
```

The following discussion describes how to determine the *origin point* for each of the two DISPLAY and ACCEPT pairs in the code fragments above. Assume the display device has 24 lines and 80 columns.

a. Remember, all screen descriptions assume an *origin point* of line 1, column 1. This screen has a positioning definition of line 23, column 60, and the first screen DISPLAY statement contains no positioning (line or column) clauses. Therefore, the *origin point* for the first DISPLAY is line 23, column 60.

b. For the second screen DISPLAY statement, which contains the positioning clauses AT LINE 5 COLUMN 30, the offset position will be line 28, column 90. (We added the line and column variable-positioning values in the DISPLAY statement to the *origin point* established in the previous step.)

c. Now, we subtract the line and column size of the display device, to find the wrap values: line 4, column 10. This becomes the *new origin point*.

d. Finally, add the line and column positioning values which in turn will be offset to line 9, column 40. Therefore, the second screen DISPLAY will begin at line 9, column 40.

e. Determining the *origin point* for the input field is similar. See the table, Variable Origin for DISPLAY and ACCEPT, on page 284 in the discussion of the ACCEPT statement.

(16) If variable origin is used for a DISPLAY operation on a screen-name, the same variable origin specification should be used for the corresponding ACCEPT statement of the screen-name in order to have the operation to be correct.

(17) If *screen-name* specifies a group item, the group item and all subordinate group, literal, input-output, output, and update fields are processed in the order in which they appear in the source definition of the screen description.

(18) The basic operation of the DISPLAY statement is described by the following steps. The discussion assumes that *screen-name* represents a group item in the screen description that has several subordinate literal, output, input-output, and/or update fields. The case where *screen-name* specifies a single screen-data item is just a simple subset of the description below.

a. The system moves the data items corresponding to all output, input-output, and update fields (either specified by or subordinate to *screen-name*) to the screen-data item. The moves take place according to the rules for the MOVE statement.

b. The system moves underscores to all input fields (either specified by or subordinate to *screen-name*).

c. The screen management system processes each field in the order in which it was defined in the source.

d. The various clauses of the screen field are processed in the following order:

BACKGROUND-COLOR & FOREGROUND-COLOR
BLANK SCREEN
COLUMN and LINE positioning
BLANK LINE/ERASE EOL, ERASE EOS, ERASE LINE
BELL
display screen-literal or screen-data with appropriate attributes

e. The screen-data or screen-literal value is displayed with the display attributes set by implied attributes or the explicit use of attribute control keywords in the screen description entry.

f. The cursor is left positioned at the character position following the last character of the last field or literal displayed according to the preceding steps.

Format 3: (data-item display with screen control)

(19) The DISPLAY statement causes the content of each operand to be transferred to the console device in the order listed.

(20) Format 3 assumes that the device is capable of random positioning.

(21) The BACKGROUND-COLOR and FOREGROUND-COLOR phrases determine the background and foreground colors used during the processing of *identifier-1* or *literal-1*. The color is identified by an integer value from 0 to 7 specified for *literal-5* or *literal-8* or as the contents of *identifier-5* or *identifier-8*. It may also be specified by use of *color-name-1* or *color-name-2*. The color names with their integer values are BLACK=0, BLUE=1, GREEN=2, CYAN=3, RED=4, MAGENTA=5, BROWN=6, WHITE=7. BACKGROUND is a synonym for BACKGROUND-COLOR and FOREGROUND is a synonym for FOREGROUND-COLOR.

(22) The BELL phrase causes the bell (or beep) signal to sound as each *identifier-1* or *literal-1* is processed.

(23) BLINK causes the data displayed for the field to be displayed in a blinking mode.

(24) The COLUMN and LINE phrases are used to position *identifier-1* or *literal-1* on the screen based on the line and leftmost character position. The top line is line 1 and each succeeding line has a value one larger than the

previous line. The leftmost character of a line is column 1 and the column value increases by one for each succeeding character on the line. The line number is specified by *literal-9* or the contents of *identifier-9* and should be between 1 and 128. The column number is specified by *literal-6* or the contents of *identifier-9*.

The line and column positions are determined as follows:

(a) If the COLUMN phrase is omitted, column 1 is assumed for the first *identifier-1* or *literal-1* if a UNIT phrase has been specified for the same *identifier-1* or *literal-1*. Otherwise the column position is set to zero.

(b) If the LINE phrase is omitted or the line position is zero the line position is set as follows: If an ERASE or ERASE SCREEN phrase is specified for the same *identifier-1* or *literal-1*, then line 1 is assumed. If the column position is not zero, the line position is the current line plus one. If the column position is zero, the line position is set to the current line.

(c) If the column position is equal to zero, it is set to the current line.

At runtime, values outside the allowable ranges are wrapped.

(25) The CONTROL phrase is used to dynamically specify options to be used or overridden. *Identifier-7* or *literal-7* are used to hold an options list. This list consists of a series of keywords separated by commas. The keywords may be specified in any order, but are processed from left to right as they appear in the string. While processing the list, lowercase characters are considered equivalent to the corresponding uppercase character and blanks or unprintable characters are ignored.

The following keywords impact execution of the DISPLAY statement:
BEEP, BLINK, CONVERT, ERASE, ERASE EOL, ERASE EOS, ERASE LINE, ERASE SCREEN, REVERSE, HIGH, LOW, NO BEEP, NO BLINK, NO CONVERT, NO ERASE, NO REVERSE, NO UNDERLINE, and UNDERLINE.

Each of the keywords has the same meaning as when statically coded plus the negative versions (NO xxx) to allow suppression of the of the option.

(26) The CONVERT phrase is used to control output conversion. If *identifier-1* or *literal-1* is numeric or numeric edited and the CONVERT phrase is specified, its value is converted from its internal form a displayable form such that a leading separate sign is provided for negative values, an explicit decimal point is added for non-integers, leading zeros are removed and the remaining digits are left-justified. If the SIZE clause adjusts the width of the field, spaces will fill any unused character positions to the right of the value or the converted values will be truncated if the field size is too small.

If the CONVERT phrase is not specified or if *identifier-1* or *literal-1* is not numeric, then *identifier-1* or *literal-1* will be treated as an alphanumeric item of its internal size and moved to the display field according to the rules for a alphanumeric to alphanumeric edited MOVE.

(27) The ERASE clause is used to control erasure of portions of the screen prior to displaying *identifier-1* or *literal-1*. ERASE SCREEN and ERASE with no additional modifiers erases the entire screen and positions the cursor to line 1 column 1. ERASE LINE erases the current line from column 1 to the end of the line without changing the cursor position. ERASE EOL erase the screen starting at the cursor position to the end of the line. The cursor is not affected. ERASE EOS erase the screen starting at the cursor position and continuing to the end of the screen. The cursor position is not changed.

(28) The HIGH, HIGHLIGHT, BOLD, and BRIGHT options cause *identifier-1* or *literal-1* to be displayed at high intensity. The LOW, LOWLIGHT, and DIM options cause *identifier-1* or *literal-1* to be displayed at low intensity.

(29) The REVERSE, REVERSED, and REVERSE-VIDEO options cause *identifier-1* or *literal-1* to be displayed in reverse video mode. If not specified, data is displayed in normal mode.

(30) The SIZE clause controls the size of the screen input field. If the SIZE clause is present and *literal-10* or the contents of *identifier-10* is not zero, the size of the screen field is determined by the value of *literal-10* or *identifier-10*. Otherwise, the size of the screen field is determined by description of *identifier-1* or *literal-1*.

When *identifier-1* is numeric and output conversion(CONVERT) is specified or implied, the size is the number of digits in *identifier-1's* PICTURE plus 1 if its is signed plus 1 if it is not an integer.

If *literal-1* is a figurative constant, the constant will be repeated up to the size specified by *identifier-10* or *literal-10*.

(31) The UNDERLINE and UNDERLINED options cause *identifier-1* or *literal-1* to be displayed in underlined mode.

(32) The UNIT clause is for documentation only and is ignored except for its impact on the COLUMN clause as previously described.

NOTES:

(1) **ICOBOL** treats all DISPLAY statements as if they are going to a DG terminal. (**ICOBOL** also treats all WRITE statements for ASSIGN TO PRINTER or ASSIGN TO DISPLAY files that are opened on the current console as if they are going to a DG terminal.) It does this to optimize characters sent to the terminal and to keep track of the state of the screen. To send binary data transparently to the terminal, an ASSIGN TO DISK "@CON" should be used in conjunction with a WRITE statement. This will insure that **ICOBOL** will not interpret the characters as screen display.

(2) The special characters the **ICOBOL** display module understands are the Print Pass Through ON and OFF codes, Read Model-ID, Compress mode ON and OFF, and display attributes like dim, blink, roll, reverse, etc. All tab characters will display as a space when not in binary mode. Other non-printable characters are sent to the screen as is, but the cursor is not moved.

(3) Neither a non-screen DISPLAY without the NO ADVANCING clause nor a screen DISPLAY statement should be executed while the terminal has Print Pass Through ON.

E.18. DIVIDE

E.18.1 Function

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder.

E.18.2 General Format

Format 1:

```
DIVIDE { identifier-1 }          INTO { identifier-2 [ ROUNDED ] }...
       { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

Format 2:

```
DIVIDE { identifier-1 }  { identifier-2 }          GIVING { identifier-3 [ ROUNDED ] }...
       { literal-1    }  { literal-2    } INTO
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

Format 3:

```
DIVIDE { identifier-1 }  { identifier-2 }          GIVING { identifier-3 [ ROUNDED ] }...
       { literal-1    }  { literal-2    } BY
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

Format 4:

```
DIVIDE { identifier-1 }  { identifier-2 }          GIVING identifier-3 [ ROUNDED ] REMAINDER identi
       { literal-1    }  { literal-2    } INTO
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

Format 5:

```
DIVIDE { identifier-1 }  { identifier-2 }          GIVING identifier-3 [ ROUNDED ] REMAINDER identifie
       { literal-1    }  { literal-2    } BY
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-DIVIDE ]
```

E.18.3 Syntax Rules

(1) Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING or REMAINDER phrase must refer to either an elementary numeric item or an elementary numeric edited item.

(2) Each literal must be a numeric literal.

(3) The composite of operands, which is the hypothetical data item resulting from the superimposition of all receiving data items (except the REMAINDER data item) of a given statement aligned on their decimal points, must not contain more than 18 digits.

E.18.4 General Rules

(1) When Format 1 is used, *literal-1* or the value of the data item referenced by *identifier-1* is divided into the value of the data item referenced by *identifier-2*. The value of the dividend (the value of the data item referenced by *identifier-2*) is replaced by this quotient.

(2) When Format 2 is used, *literal-1* or the value of the data item referenced by *identifier-1* is divided into *literal-2* or the value of the data item referenced by *identifier-2* and the result is stored in each data item referenced by *identifier-3*.

(3) When Format 3 is used, *literal-1* or the value of the data item referenced by *identifier-1* is divided by *literal-2* or the value of the data item referenced by *identifier-2* and the result is stored in each data item referenced by *identifier-3*.

(4) When Format 4 is used, *literal-1* or the value of the data item referenced by *identifier-1* is divided into *literal-2* or the value of the data item referenced by *identifier-2* and the result is stored in the data item referenced by *identifier-3*. The remainder is then calculated and the result is stored in the data item referenced by *identifier-4*. If *identifier-4* is subscripted, the subscript is evaluated immediately before the remainder is stored in the data item referenced by *identifier-4*.

(5) When Format 5 is used, *literal-1* or the value of the data item referenced by *identifier-1* is divided by *literal-2* or the value of the data item referenced by *identifier-2* and the division continues as specified for Format 4 above.

(6) Formats 4 and 5 are used when a remainder from the division operation is desired, namely *identifier-4*. The remainder in COBOL is defined as the result of subtracting the product of the quotient (*identifier-3*) and the divisor from the dividend. If *identifier-3* is defined as a numeric edited item, the quotient used to calculate the remainder is an intermediate field which contains the unedited quotient. If ROUNDED is specified, the quotient used to calculate the remainder is an intermediate field which contains the quotient of the DIVIDE statement, truncated rather than rounded. This intermediate field is defined as a numeric field which contains the same number of digits, the same decimal point location, and the same presence or absence of a sign as the quotient (*identifier-3*).

(7) In Formats 4 and 5, the accuracy of the REMAINDER data item (*identifier-4*) is defined by the calculation described above. Appropriate decimal alignment and truncation (not rounding) will be performed for the value of the data item referenced by *identifier-4*, as needed.

(8) When the ON SIZE ERROR phrase is used in Formats 4 and 5, the following rules pertain:

a. If the size error occurs on the quotient, no remainder calculation is meaningful. Thus, the contents of the data items referenced by both *identifier-3* and *identifier-4* will remain unchanged.

b. If the size error occurs in the remainder, the content of the data item referenced by *identifier-4* remains unchanged. However, as with other instances of multiple results of arithmetic statements, the user will have to do his own analysis to recognize which situation has actually occurred.

(9) Additional rules and explanations relative to this statement are given under the appropriate paragraphs. (See Scope of Statements, page 253; The ROUNDED Phrase, page 245; The ON SIZE ERROR Phrase, page 246; The Arithmetic Statements, page 249; Overlapping Operands, page 249; and Multiple Results in Arithmetic Statements, page 249.

E.19. EVALUATE (***ANSI 74*** and ***ANSI 85***)

E.19.1 Function

The EVALUATE statement describes a multi-branch, multi-join structure. It may cause multiple conditions to be evaluated. The subsequent action of the runtime element depends on the results of these evaluations.

E.19.2 General format

```
EVALUATE selection-subject [ ALSO selection-subject ]...
    { { WHEN selection-object [ ALSO selection-object ]... }... imperative-statement-1 }...
     [ WHEN OTHER imperative-statement-2 ]
    [ END-EVALUATE ]
```

where

selection-subject is:

```
[ identifier-1          ]
[ literal-1             ]
[ arithmetic-expression-1 ]
[ condition-1           ]
[ TRUE                  ]
[ FALSE                 ]
```

selection-object is:

```
{ [ NOT ] identifier-2          }
{ [ NOT ] literal-2             }
{ [ NOT ] arithmetic-expression-2 }
{ [ NOT ] range-expression      }
{ [ NOT ] indicator-value       }
{ condition-2                   }
{ TRUE                          }
{ FALSE                         }
{ ANY                           }
```

range-expression is:

```
{ identifier-3          } { THROUGH } { identifier-4          }
{ literal-3             } { THRU    } { literal-4             }
{ arithmetic-expression-3 }           { arithmetic-expression-4 }
```

indicator-value is:

```
{ NULL     }
{ VALID    }
{ OVERFLOW }
```

E.19.3 Syntax rules

(1) The words THROUGH and THRU are equivalent.

(2) The number of selection objects within each set of selection objects shall be equal to the number of selection subjects.

(3) The two operands in a *range-expression* shall be of the same class and shall not be of class pointer.

(4) Each selection object within a set of selection objects shall correspond to the selection subject having the same ordinal position within the set of selection subjects according to the following rules:

a. Identifiers, literals, or expressions appearing within a selection object shall be valid operands for comparison to the corresponding operand in the set of selection subjects in accordance with the rules for Relation conditions, on Page 233.

b. *Condition-2* or the words TRUE or FALSE appearing as a selection object shall correspond to *condition-1* or the words TRUE or FALSE in the set of selection subjects.

c. The word ANY may correspond to a selection subject of any type.

d. (***ISQL***) Date-time and interval operands are permitted subject to the rules for Relation Conditions, page 233, and Arithmetic Expressions, page 230.

e. (***ISQL***) *Indicator-value* appearing as a selection object shall correspond to *identifier-1* as a selection subject, where *identifier-1* has usage INDICATOR.

(5) The permissible combinations of selection subject and selection object operands are indicated in the following table, Combination of operands in the EVALUATE statement.

Selection object	Selection subject				
	Identifier	Literal	Arithmetic expression	Condition	TRUE or FALSE
[NOT] identifier	Y	Y	Y		
[NOT] literal	Y		Y		
[NOT] arithmetic-expression	Y	Y	Y		
[NOT] range-expression	Y	Y	Y		
[NOT] Indicator-value	Y*				
Condition				Y	Y
TRUE or FALSE				Y	Y
ANY	Y	Y	Y	Y	Y

The letter 'Y' indicates a permissible combination.
A space indicates an invalid combination.
* indicates restrictions apply

TABLE 24. Combination of operands in the EVALUATE statement

E.19.4 General rules

(1) At the beginning of the execution of the EVALUATE statement, each selection subject is evaluated and assigned a value, a range of values, or a truth value as follows:

a. Any selection subject specified by *identifier-1* is assigned the value and class of the data item referenced by the identifier.

b. Any selection subject specified by *literal-1* is assigned the value and class of the specified literal.

c. Any selection subject specified by *arithmetic-expression-1* is assigned a numeric value according to the rules for evaluating an arithmetic expression.

d. Any selection subject specified by *condition-1* is assigned a truth value according to the rules for evaluating conditional expressions.

e. Any selection subject specified by the words TRUE or FALSE is assigned a truth value. The truth value 'true' is assigned to those items specified with the word TRUE, and the truth value 'false' is assigned to those items specified with the word FALSE.

(2) The execution of the EVALUATE statement proceeds by processing each WHEN phrase from left to right in the following manner:

a. Each selection object within the set of selection objects for each WHEN phrase is paired with the selection subject having the same ordinal position within the set of selection subjects. The result of the analysis of this set of selection subjects and objects is either true or false as follows:

1. If the selection object is the word ANY, the result is true.

2. If the selection object is *condition-2*, the selection subject is either TRUE or FALSE. If the truth value of the selection subject and selection object match, the result of the analysis is true. If they do not match, the result is false.

3. If the selection object is either TRUE or FALSE, the selection subject is condition-1. If the truth value of the selection subject and selection object match, the result of the analysis is true. If they do not match, the result is false.

4. If the selection object is a *range-expression*, the pair is considered to be a conditional expression of one of the following forms:

   when "NOT" is not specified in the selection object;

   selection-subject >= left-part AND *selection-subject* <= right-part

   when "NOT" is specified in the selection object

   selection-subject < left-part OR *selection-subject* > right-part

   where left-part is *identifier-3*, *literal-3*, or *arithmetic-expression-3* and right-part is *identifier-4*, *literal-4*, or *arithmetic-expression-4*. The result of the analysis is the truth value of the resulting conditional expression.

5. If the selection object is *identifier-2*, *literal-2*, or *arithmetic-expression-2*, the pair is considered to be a conditional expression of the following form:

   selection-subject [NOT] = *selection-object*

   where "NOT" is present if it is present in the selection object. The result of the analysis is the truth value of the resulting conditional expression.

6. (***ISQL***) If the selection object is *indicator-value*, the pair is considered to be an indicator condition of the following form:

   Identifier-1 IS [NOT] *indicator-value*

b. If the result of the analysis is true for every pair in a WHEN phrase, that WHEN phrase satisfies the set of selection subjects and no more WHEN phrases are analyzed.

c. If the result of the analysis is false for any pair in a WHEN phrase, no more pairs in that WHEN phrase are evaluated and the WHEN phrase does not match the set of selection subjects.

d. This procedure is repeated for subsequent WHEN phrases, in the order of their appearance in the source element, until either a WHEN phrase satisfying the set of selection subjects is selected or until all sets of selection objects are exhausted.

(3) The execution of the EVALUATE statement then proceeds as follows:

a. If a WHEN phrase is selected, execution continues with the first *imperative-statement-1* following the selected WHEN phrase.

b. If no WHEN phrase is selected and a WHEN OTHER phrase is specified, execution continues with *imperative-statement-2*.

c. The execution of the EVALUATE statement is terminated when execution reaches the end of *imperative-statement-1* of the selected WHEN phrase or the end of *imperative-statement-2*, or when no WHEN phrase is selected and no WHEN OTHER phrase is specified.

E.19.5 Example

The following code demonstrates the EVALUATE statement:

```
EVALUATE YEAR-CODE ALSO LETTER-GRADE
    WHEN 1 THRU 2 ALSO "A" THRU "C"
        PERFORM PROC-1
    WHEN 3 ALSO "A" THRU "B"
        PERFORM PROC-2
    WHEN 4 ALSO ANY
        PERFORM PROC-3
    WHEN OTHER
        PERFORM PROC-4
    END-EVALUATE.
```

EXAMPLE 22. EVALUATE

In this example, if YEAR-CODE is 1 or 2 and LETTER-GRADE is A, B or C, PROC-1 is performed. If YEAR-CODE is 3 and LETTER-GRADE is A or B, PROC-2 is performed. If YEAR-CODE is 4, PROC-3 is performed regardless of LETTER-GRADE. Any other combination of YEAR-CODE and LETTER-GRADE will cause the execution of PROC-4.

E.20. EXECUTE (*ISQL*)

E.20.1 Function

The EXECUTE statement provides the ability to execute an SQL statement using a statement that has been prepared using the PREPARE statement.

E.20.2 General Format

```
EXECUTE { identifier-1 }          [ INTO { identifier-2 [ INDICATOR identifier-3 ] } ...]
        { literal-1    }
    [ USING {{ identifier-4 }          [ INDICATOR identifier-5 ] } ... ]
            { literal-2    }
  [ ON SQLERROR imperative-statement-1 ]
  [ NOT ON SQLERROR imperative-statement-2 ]
  [ END-EXECUTE ]
```

E.20.3 Syntax Rules

(1) *Literal-1* must specify a nonnumeric literal and must not specify a figurative constant.

(2) *Identifier-1* must specify an alphanumeric data item.

(3) *Literal-1* or the content of the data item referenced by *identifier-1* must not exceed 30 characters in length.

(4) *Identifier-3* and *identifier-5* must identify data items with usage INDICATOR.

E.20.4 General Rules

(1) Used to execute an SQL statement that was previously prepared by means of a PREPARE statement. See the PREPARE statement, page 412.

(2) *Literal-1* or the content of the data item represented by *identifier-1* specifies the name of a statement container at runtime. The statement container must hold the result of a previously executed PREPARE statement for the currently active connection.

(3) If there is no currently active connection, it is an error and SQLSTATE will be set to “HY010", which is “Function sequence error”.

(4) If the name of the statement container cannot be found in the context of the currently active connection, it is an error and SQLSTATE will be set to “26501", which is “The statement identifier does not exist”.

(5) If the INTO clause is specified, the data items specified by *identifier-2* will receive the first row of the result set of the executed statement. If any *identifier-2* has an associated INDICATOR variable, *identifier-3*, it will be set in conjunction with the setting of the value of *identifier-2*. The first *identifier-2* will be set to the first column in the row, the second *identifier-2* will be set to the second column in the row, etc. If there are more columns in the row than specified *identifier-2*'s then SQLSTATE will be set to “01503". If there are no rows in the result set, SQLSTATE will be set to “02000", which is “No data was affected by the operation”. If there are additional rows in the result set, they can be fetched with the FETCH statement.

(6) If the INTO clause is not specified, and the EXECUTE statement is successful, the results can be fetched with the FETCH statement.

(7) If there is no associated indicator variable for a null-able column that is null, SQLSTATE will be set to "22002", which is "Indicator variable required but not supplied".

(8) When the prepared statement uses dynamic parameter specifiers, the USING clause must be specified, and the values of *literal-2* or the data items specified by *identifier-4* are used in the order specified to satisfy the binding of values to dynamic parameter specifiers. The literals or data items should be of an appropriate class and category for their usage in the SQL statement and any associated INDICATOR variable, specified by *identifier-5*, should be set before the EXECUTE statement is executed.

(9) If there is no currently active connection at the time the EXECUTE statement is executed, it is an error and SQLSTATE will be set to "HY010", which is "Function sequence error".

(10) Upon completion of the EXECUTE statement, the following occurs in the order specified:

a. The value of the SQLSTATE data item updated with the status of the operation.

b. If the value of the SQLSTATE class field is "00" or "01", the statement is successful. Control is transferred to the end of the EXECUTE statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the EXECUTE statement.

c. If the value of the SQLSTATE class field is not "00" or "01", the statement is unsuccessful. Control is transferred to the end of the EXECUTE statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the EXECUTE statement.

(11) The END-EXECUTE phrase delimits the scope of the EXECUTE statement.

(12) More on SQLSTATE can be found on page 133.

E.21. EXECUTE IMMEDIATE (*ISQL*)

E.21.1 Function

The EXECUTE IMMEDIATE statement provides the ability to execute an SQL statement by directly preparing and executing the statement as a single operation. No result set is allowed. No parameter markers are allowed.

E.21.2 General Format

```
EXECUTE IMMEDIATE { identifier-1 }
                  { literal-1    }
    [ ON SQLERROR imperative-statement-1 ]
    [ NOT ON SQLERROR imperative-statement-2 ]
    [ END-EXECUTE ]
```

E.21.3 Syntax Rules

(1) *Literal-1* must specify a nonnumeric literal and must not specify a figurative constant.

(2) *Identifier-1* must specify an alphanumeric data item.

E.21.4 General Rules

(1) Used to both prepare and execute a basic dynamic SQL statement. It cannot be used with parameter markers. Use the PREPARE and EXECUTE statements for that.

(2) *Literal-1* or the content of the data item represented by *identifier-1* specifies the text of the SQL statement that is to be prepared for execution. The text of the SQL statement may not contain references to COBOL data items, nor may it contain any use of the dynamic parameter specifier.

(3) The set of SQL statements that may be specified for preparation and execution is limited to the following:

- DELETE
- INSERT
- UPDATE

(4) If there is no currently active connection at the time the EXECUTE IMMEDIATE statement is executed, it is an error and SQLSTATE will be set to “HY010", which is “Function sequence error”.

(5) Upon completion of the EXECUTE IMMEDIATE statement, the following occurs in the order specified:

a. The value of the SQLSTATE data item updated with the status of the operation.

b. If the value of the SQLSTATE class field is “00" or “01", the statement is successful. Control is transferred to the end of the EXECUTE IMMEDIATE statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the EXECUTE IMMEDIATE statement.

c. If the value of the SQLSTATE class field is not “00" or “01", the statement is unsuccessful. Control is transferred to the end of the EXECUTE IMMEDIATE statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is

transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the EXECUTE IMMEDIATE statement.

(6) The END-EXECUTE phrase delimits the scope of the EXECUTE IMMEDIATE statement.

(7) More on SQLSTATE can be found on page 133.

NOTE: If the same SQL statement is to be executed more than once, it is more efficient to use the PREPARE and EXECUTE statements rather than the EXECUTE IMMEDIATE statement.

E.22. EXIT

E.22.1 Function

The EXIT statement provides a common end point for a series of procedures.

E.22.2 General Format

EXIT

E.22.3 Syntax Rules

(1) The EXIT statement must appear only in a sentence by itself and comprise the only sentence in the paragraph.

E.22.4 General Rules

(1) An EXIT statement serves only to enable the user to assign a procedure-name to a given point in a program. Such an EXIT statement has no other effect on the compilation or execution of the program.

E.23. EXIT PROGRAM

E.23.1 Function

The EXIT PROGRAM statement marks the logical end of a called program.

E.23.2 General Format

EXIT PROGRAM

E.23.3 Syntax Rules

(1) If an EXIT PROGRAM statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

E.23.4 General Rules

(1) If the EXIT PROGRAM statement is executed in a program which is not under the control of a calling program, the EXIT PROGRAM statement causes execution of the program to continue with the next executable statement.

(2) The execution of an EXIT PROGRAM statement in a called program which does not possess the initial attribute causes execution to continue with the next executable statement following the CALL statement in the calling program. The program state of the calling program is not altered and is identical to that which existed at the time it executed the CALL statement except that the contents of data items and the contents of data files shared between the calling and called program may have been changed. The program state of the called program is not altered except that the ends of the ranges of all PERFORM statements executed by that called program are considered to have been reached.

(3) The storage areas associated with all items in the USING phrase of the Procedure Division header of the called program are copied to the associated storage areas, in the USING phrase, of the calling program.

(4) Besides the actions specified in general rule 2, the execution of an EXIT PROGRAM statement in a called program which possesses the initial attribute is equivalent to also executing a CANCEL statement referencing that program.

E.24. EXPUNGE (*VXCOBOL*)

E.24.1 Function

The EXPUNGE statement physically removes a file from the file system. EXPUNGE is an extension to ANSI COBOL. It is equivalent to DELETE FILE.

E.24.2 General Format

EXPUNGE { *file-name* }...

E.24.3 General Rules

(1) The file referenced by *file-name* must be a disk file, you must have appropriate permissions, and the file must not be open at the time of the execution of this statement. If the files does not exist, no error is given.

(2) For a relative, indexed, or INFOS file all parts of that file are removed from the file system.

(3) For an INFOS II file the indexed file and the database file specified in the SELECT statement are deleted. If the name of the database file was not specified with an ASSIGN DATA clause, a .DB file with the same name as that of the indexed file is deleted. For a U/FOS file, the database specified in the SELECT is deleted, i.e. name.udb.

(4) After the successful execution of an EXPUNGE statement, the identified file has been physically removed from the file system and can no longer be accessed.

(5) The execution of the EXPUNGE statement causes the value of the I-O status associated with *file-name* to be updated.

(6) For systems supporting UNIX symbolic links, DELETE FILE will delete the symbolic link, not the resolution file.

(7) On UNIX, files cannot be individually delete-protected. To make a file delete-protected on UNIX, you must remove write (w) permission to the directory in which the file resides. If a directory has no write access, you cannot create, modify or delete files in that directory. On Windows, the read-only attribute will protect the file from deletion.

(8) For **ANSI 74 and ANSI 85**, If the specified file is a sequential file, **ICOBOL** will scan the Printer Control file and if there is an entry there that points to the file being deleted, the entry in the Printer Control file will be removed.

(9) If *file-name* is a sort/merge file, it is ignored.

E.25. EXPUNGE SUB-INDEX (***VXCOBOL***)

E.25.1 Function

The EXPUNGE SUB-INDEX statement deletes or unlinks a subindex from a specified key.

E.25.2 General Format

```
EXPUNGE SUB-INDEX file-name [ { FIX    } POSITION ] [ NEXT          ]
                            { RETAIN }            [ FORWARD       ]
                                                  [ BACKWARD      ]
                                                  [ UP            ]
                                                  [ DOWN          ]
                                                  [ UP FORWARD    ]
                                                  [ UP BACKWARD   ]
                                                  [ DOWN FORWARD  ]
                                                  [ STATIC        ]

    [ { KEY IS  } { identifier-1 [ APPROXIMATE ] }... ]
      { KEYS ARE }               [ GENERIC     ]
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-EXPUNGE ] ]
```

E.25.3 Syntax Rules

(1) *File-name* is a filename that specifies an INFOS file opened for OUTPUT or I/O and selected for ALLOW SUB-INDEX.

(2) *Identifier-1* is an alphanumeric data item that specifies a record key associated with *file-name*.

E.25.4 General Rules

(1) **ICOBOL** decrements the use count of the subindex associated with the specified key. If the use count of the subindex goes to zero, the subindex is unlinked from the key and physically deleted. If the use count of the subindex remains one or more, the subindex is simply unlinked.

(2) If the position phrase is omitted, RETAIN POSITION is the default.

(3) If the relative option and the KEY series phrase are omitted, the default is the first key in the SELECT clause.

(4) The occurrence number is not updated.

(5) FEEDBACK is not used and is not updated.

(6) KEY LENGTH is unaffected.

(7) The subindex to remove is determined according to what is specified in the relative option phrase and/or the KEY series phrase.

(8) FIX POSITION causes the record pointer to move from the current position to the position specified in this statement. RETAIN POSITION causes the record position to remain at the position it was on before the execution of this statement. RETAIN is the default.

(9) The relative motion option without the KEY series phrase allows access to the index file relative to that file's current record position.

(10) Using the KEY series phrase without the relative motion option causes the key path specified to begin with the top index in the hierarchy and follow a downward motion.

(11) If the KEY series phrase is specified, each key, *identifier-1*, must be declared in the SELECT statement for *file-name*. If the relative motion option and KEY series phrase at both specified only UP, DOWN, and STATIC are allowed. The relative motion option is processed first and the key path is used. If both are omitted, STATIC is the default.

(12) Transfer of control following the successful or unsuccessful execution of the EXPUNGE SUB-INDEX operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the EXPUNGE SUB-INDEX statement.

(13) INVALID KEY clauses on I/O statements are ONLY invoked when an Invalid Key error, as determined by a File Status of 2*x* where *x* can be any character 0 - 9 or A - Z, is generated. All other error conditions will cause the associated USE procedure, if present, as defined in the DECLARATIVES section to be executed. (See The Invalid Key Condition, page 271, for more a more comprehensive discussion.)

E.26. FETCH (*ISQL*)

E.26.1 Function

The FETCH statement provides the ability to fetch the next row from a result set. FETCH works using a forward-only-cursor.

E.26.2 General Format

```
FETCH NEXT FOR { identifier-1 | literal-1 }        INTO { identifier-2 [ INDICATOR identifier-3 ] } ...
    [ ON SQLERROR imperative-statement-1 ]
    [ NOT ON SQLERROR imperative-statement-2 ]
    [ END-FETCH ]
```

E.26.3 Syntax Rules

(1) *Literal-1* must specify a nonnumeric literal and must not specify a figurative constant.

(2) *Identifier-1* must specify an alphanumeric data item.

(3) *Identifier-3* must identify a data item with usage INDICATOR.

E.26.4 General Rules

(1) *Literal-1* or the content of the data item represented by *identifier-1* specifies the name of a statement container at runtime. The statement container must hold the result of a previously executed EXECUTE statement for the currently active connection.

(2) If there is no currently active connection or the previously executed EXECUTE statement was not successful, it is an error and SQLSTATE will be set to “HY010", which is “Function sequence error”.

(3) If the name of the statement container cannot be found in the context of the currently active connection, it is an error and SQLSTATE will be set to “26501", which is “The statement identifier does not exist”.

(4) If there is no next row, the SQLSTATE will be set to “02000", which is “No data was affected by the operation”. If there were no rows at all in the result set, SQLSTATE will be set to “24000", which is “Invalid cursor state”.

(5) The data items specified by *identifier-2* will receive the results of the fetched row. If any *identifier-2* has an associated INDICATOR variable, *identifier-3*, it will be set in conjunction with the setting of the value of *identifier-2*. The first *identifier-2* will be set to the first column in the row, the second *identifier-2* will be set to the second column in the row, etc. If there are more columns in the row than specified *identifier-2*'s then SQLSTATE will be set to “01503".

(6) If there is no associated indicator variable for a null-able column that is null, SQLSTATE will be set to “22002", which is “Indicator variable required but not supplied”.

(7) Upon completion of the FETCH statement, the following occurs in the order specified:

a. The value of the SQLSTATE data item updated with the status of the operation.

b. If the value of the SQLSTATE class field is “00" or “01", the statement is successful. Control is transferred to the end of the FETCH statement or to *imperative-statement-2*, if specified. In the latter case,

execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the FETCH statement.

c. If the value of the SQLSTATE class field is not "00" or "01", the statement is unsuccessful. Control is transferred to the end of the FETCH statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the FETCH statement.

(8) The END-FETCH phrase delimits the scope of the FETCH statement.

(9) More on SQLSTATE can be found on page 133.

E.27. GET DIAGNOSTICS (*ISQL*)

E.27.1 Function

The GET DIAGNOSTICS statement allows the program to retrieve information from the diagnostics area of the SQL database connection. There are two formats to this statement. The first retrieves information relating to the overall execution of the immediately preceding SQL statement (not counting GET DIAGNOSTICS statements themselves). The second format is used to gain more specific information regarding some particular exception.

E.27.2 General Format

Format 1:

```
GET DIAGNOSTICS { identifier-1 = { ROW COUNT
                                   NUMBER
                                   COMMAND FUNCTION
                                   DYNAMIC FUNCTION } } . . .
    [ ON EXCEPTION imperative-statement-1 ]
    [ NOT ON EXCEPTION imperative-statement-2 ]
    [ END-GET ]
```

Format 2:

```
GET DIAGNOSTICS EXCEPTION { identifier-2 } { identifier-3 = { SQLSTATE
                          { literal-1    }                  { NATIVE ERROR
                                                            { MESSAGE TEXT
                                                            { MESSAGE LENGTH } } . . .
    [ ON EXCEPTION imperative-statement-1 ]
    [ NOT ON EXCEPTION imperative-statement-2 ]
    [ END-GET ]
```

E.27.3 Syntax Rules

Format 1:

(1) *Identifier-1* must specify an integer data item without any p-scaling with the ROW COUNT or NUMBER phrase.

(2) *Identifier-1* must specify an alphanumeric data item with the COMMAND FUNCTION or DYNAMIC FUNCTION phrase.

Format 2:

(3) *Identifier-2/Literal-1* must specify an integer value.

(4) *Identifier-3* must specify an integer data item without any p-scaling with the NATIVE ERROR or MESSAGE LENGTH phrase.

(5) *Identifier-3* must specify an alphanumeric data item with the SQLSTATE or MESSAGE TEXT phrase.

E.27.4 General Rules

(1) All assignment operations are carried out in the order specified in the source text.

(2) It is permissable to specify a given assignment phrase more than once.

(3) The GET DIAGNOSTICS statement itself does not effect the diagnostics information stored in the system.

(4) The diagnostics information is valid until the next ***ISQL*** statement is executed.

(5) Other than the requirement that the first diagnostic record corresponds to the SQLSTATE returned by an ***ISQL*** statement, the diagnostic records are not in any particular order. However, since they are added as they are encountered, they will generally follow the pattern that diagnostics pertaining to statement preparation (such as binding parameters) will occur before the diagnostics for the main operation, which will precede diagnostics from returning results.

Format 1:

(6) ROW COUNT returns the number of rows affected by an INSERT, UPDATE, or DELETE (***ISQL***) statement. It does not necessarily return a meaningful value for any other statement.

(7) NUMBER returns the number of diagnostic messages that are available in the diagnostics area. Format 2 can be used to retrieve each individual message.

(8) COMMAND FUNCTION returns a string that specifies the ***ISQL*** statement that was executed.

(9) DYNAMIC FUNCTION returns a string for EXECUTE or EXECUTE IMMEDIATE that specifies the dynamic SQL statement that was executed (e.g., SELECT). For all other statements, it will return an empty string.

Format 2:

(10) The exception number specifier in *identifier-2*|*literal-1* must be greater than zero and less than or equal to the number of exceptions as would be returned into identifier by a "GET DIAGNOSTICS *identifier* = NUMBER" statement.

(11) A non-success SQLSTATE returned by an ***ISQL*** statement corresponds to the value returned by GET DIAGNOSTICS EXCEPTION 1 id = SQLSTATE. I.E., SQLSTATE returns the SQLSTATE corresponding to the diagnostic record.

(12) NATIVE ERROR returns the numeric error code that may have orginiated in the driver, the driver manager, or the runtime system. It is usually not useful to the logic of the application but may provide additional diagnostic information.

(13) MESSAGE TEXT returns a diagnostic message that gives information about the error. It provides useful information as to the specific problem encountered.

(14) MESSAGE LENGTH returns the length of the text message returned in MESSAGE TEXT. This is usually not needed.

All Formats:

.

(15) Upon completion of the GET DIAGNOSTICS statement, the following occurs in the order specified:

a. If the GET DIAGNOSTICS was successful, control is transferred to the end of the GET DIAGNOSTICS statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement;

otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the GET DIAGNOSTICS statement.

b. If the GET DIAGNOSTICS is unsuccessful, control is transferred to the end of the GET DIAGNOSTICS statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the GET DIAGNOSTICS statement.

(16) The END-GET phrase delimits the scope of the GET DIAGNOSTICS statement.

(17) More on SQLSTATE can be found on page 133.

E.28. GO TO

E.28.1 Function

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

E.28.2 General Format

Format 1:

GO TO *procedure-name-1*

Format 2:

GO TO { *procedure-name-1* }... DEPENDING ON *identifier*

E.28.3 Syntax Rules

(1) *Identifier* must reference a numeric elementary data item which is an integer.

(2) If a GO TO statement represented by Format 1 appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

(3) A GO TO cannot transfer control between:

a. A procedure-name in a Declarative section from a nondeclarative section.

b. A procedure-name in a nondeclarative section from a Declaratives section.

c. A Declaratives section from another Declaratives section.

d. The above conditions are treated as errors for **ANSI 74 and ANSI 85**, but may be converted to warnings with the -G g compiler switch. ***VXCOBOL*** treats these conditions as warnings.

(4) No more than 254 *procedure-name-1* entries may be specified.

E.28.4 General Rules

(1) When a GO TO statement represented by Format 1 is executed, control is transferred to *procedure-name-1*.

(2) When a GO TO statement represented by Format 2 is executed, control is transferred to *procedure-name-1*, etc., depending on the value of *identifier* being 1, 2, ... , n. If the value of *identifier* is anything other than the positive or unsigned integers 1, 2, ... , n, (where n is the number of *procedure-name-1*'s specified), then no transfer occurs and control passes to the next statement in the normal sequence for execution.

E.29. GOBACK

E.29.1 Function

The GOBACK statement marks the logical end of a called program.

The GOBACK statement is equivalent to the sequence:

```
EXIT PROGRAM.
STOP RUN.
```

E.29.2 General Format

GOBACK

E.29.3 Syntax Rules

(1) If a GOBACK statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

E.29.4 General Rules

(1) If the GOBACK statement is executed in a program which is not under the control of a calling program, the GOBACK statement causes execution of the program to act as if a STOP RUN statement had been performed.

(2) The execution of an GOBACK statement in a called program which does not possess the initial attribute causes execution to continue with the next executable statement following the CALL statement in the calling program. The program state of the calling program is not altered and is identical to that which existed at the time it executed the CALL statement except that the contents of data items and the contents of data files shared between the calling and called program may have been changed. The program state of the called program is not altered except that the ends of the ranges of all PERFORM statements executed by that called program are considered to have been reached.

(3) The storage areas associated with all items in the USING phrase of the Procedure Division header of the called program are copied to the associated storage areas, in the USING phrase, of the calling program.

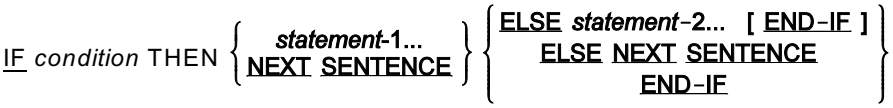
(4) Besides the actions specified in general rule 2, the execution of a GOBACK statement in a called program which possesses the initial attribute is equivalent to also executing a CANCEL statement referencing that program.

E.30. IF

E.30.1 Function

The IF statement causes a condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

E.30.2 General Format



E.30.3 Syntax Rules

(1) *Statement-1* and *statement-2* represent either an imperative statement or a conditional statement optionally preceded by an imperative statement. A further description of the rules governing *statement-1* and *statement-2* is given elsewhere.

(2) The ELSE NEXT SENTENCE phrase may be omitted if it immediately precedes to the terminal period of the sentence.

(3) If the END-IF phrase is specified, the NEXT SENTENCE phrase must not be specified.

E.30.4 General Rules

(1) The scope of the IF statement may be terminated by any of the following:

a. An END-IF phrase at the same level of nesting.

b. A separator period.

c. If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting.

(2) When an IF statement is executed, the following transfers of control occur:

a. If the condition is true and *statement-1* is specified, control is transferred to the first statement of *statement-1* and execution continues according to the rules for each statement specified in *statement-1*. If a procedure branching or conditional statement is executed which causes an explicit transfer of control, control is explicitly transferred in accordance with the rules of that statement. Upon completion of the execution of *statement-1*, the ELSE phrase, if specified, is ignored and control passes to the end of the IF statement.

b. If the condition is true and the NEXT SENTENCE phrase is specified instead of *statement-1*, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

c. If the condition is false and *statement-2* is specified, *statement-1* or its surrogate NEXT SENTENCE is ignored, control is transferred to the first statement of *statement-2*, and execution continues according to the rules for each statement specified in *statement-2*. If a procedure branching or conditional statement is executed which causes an explicit transfer of control, control is explicitly transferred in accordance with the rules of that statement. Upon completion of the execution of *statement-2*, control passes to the end of the IF statement.

d. If the condition is false and the ELSE phrase is not specified, *statement-1* is ignored and control passes to the end of the IF statement.

e. If the condition is false and the ELSE NEXT SENTENCE phrase is specified, *statement-1* is ignored and control passes to the next executable sentence.

(3) *Statement-1* and/or *statement-2* may contain an IF statement. In this case, the IF statement is said to be nested. More detailed rules on nesting are given in the appropriate paragraph. (See Scope of Statements, page 253.)

IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE or END-IF encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE or END-IF.

E.31. INITIALIZE (***ANSI 74*** and ***ANSI 85***)

E.31.1 Function

The INITIALIZE statement provides the ability to set selected data items to specified values.

E.31.2 General Format

```
INITIALIZE { identifier-1 }... [ WITH FILLER ] [ { ALL          } TO VALUE ]
                                               { category-name }
    [ THEN REPLACING { category-name DATA BY { identifier-2 } }... ]
                                             { literal-1    }
    [ THEN TO DEFAULT ]
```

where *category-name* is:

```
{ ALPHABETIC          }
{ ALPHANUMERIC        }
{ ALPHANUMERIC-EDITED }
{ NUMERIC             }
{ NUMERIC-EDITED      }
{ POINTER             }
```

(***ISQL***) The following *category-name* selections are added:

```
{ DATE                                  }
{ TIME                                  }
{ TIMESTAMP                             }
{ INTERVAL { YEAR TO MONTH }            }
{          { DAY TO TIME   }            }
{ INDICATOR                             }
```

E.31.3 Syntax rules

(1) *Identifier-1* must be a valid receiving operand of a MOVE statement, or an item with usage POINTER or INDICATOR.

(2) For each POINTER or INDICATOR phrase used as the *category-name* stated in the REPLACING phrase, *identifier-2* shall be specified, and a SET statement with *identifier-2* as the sending operand and an item of the specified category as the receiving item shall be valid..

(3) For each other *category-name* stated in the REPLACING phrase, a MOVE statement with *identifier-2* or *literal-1* as sending operand and an item of the category specified by *category-name* as receiving operand must be valid.

(4) An index data item may not appear as an operand of an INITIALIZE statement.

(5) The data description entry for the data item referenced by *identifier-1* shall not contain a RENAMES clause.

(6) The same category shall not be repeated in a REPLACING phrase.

E.31.4 General rules

(1) The data item referenced by *identifier-1* represents the receiving item.

(2) If the REPLACING phrase is specified, *literal-1* and the data item referenced by *identifier-2* represent the s7ending item.

(3) The keywords in *category-names* correspond to a category of data as specified in B.3 Concept of Classes of Data on page 118. If ALL is specified in the VALUE phrase, it is as if all of the categories listed in *category-names* were specified.

(4) Whether *identifier-1* references an elementary item or a group item, the effect of the execution of the INITIALIZE statement is as though a series of implicit MOVE or SET statements, each of which has an elementary data item as its receiving operand.

If the receiving operand is usage POINTER or INDICATOR, the implicit statement is

SET *receiving-operand* TO *sending-operand*

Otherwise, the implicit statement is

MOVE *sending-operand* TO *receiving-operand*

were executed, where the *sending-operand* is as defined in General Rule 6 and the *receiving-operand* is as defined in General Rule 5.

(5) The *receiving-operand* in each implicit MOVE or SET statement is determined by applying the following steps in order:

a. First, the following data items are excluded as *receiving-operands*:

1. Any identifiers that are not valid receiving operands of a MOVE statement, except items of usage POINTER or INDICATOR.

2. If the FILLER phrase is not specified, elementary data items with an explicit or implicit FILLER clause.

3. Any elementary data item subordinate to *identifier-1* whose data description entry contains a REDEFINES or RENAMES clause or is subordinate to a data item whose data description entry contains a REDEFINES clause. However, *identifier-1* may itself have a REDEFINES clause or be subordinate to a data item with a REDEFINES clause.

4. Any elementary data item with USAGE INDEX.

b. Second, an elementary data item is a possible receiving item if:

1. It is explicitly referenced by *identifier-1*; or

2. It is contained within the group data item referenced by *identifier-1*. If the elementary data item is a table element, each occurrence of the elementary data item is a possible *receiving-operand*.

c. Finally, each possible *receiving-operand* is a *receiving-operand* if at least one of the following is true:

1. The VALUE phrase is specified, a data-item format or table format VALUE clause is specified in the data description entry of the elementary data item, and the category of the data item is one of the categories specified or implied in the VALUE phrase; or

2. The REPLACING phrase is specified and the category of the elementary data item is one of the categories specified in the REPLACING phrase; or

3. The DEFAULT phrase is specified; or

4. Neither the REPLACING phrase nor the VALUE phrase is specified.

(6) The *sending-operand* in each implicit MOVE or SET statement is determined as follows:

a. If the data item qualifies as a *receiving-operand* because of the VALUE phrase:

1. If the *receiving-operand* is usage POINTER, the *sending-operand* is the predefined address item NULL

2. If the *receiving-operand* is usage INDICATOR, the *sending-operand* is the predefined indicator value NULL

3. Otherwise, the *sending-operand* is determined by the literal in the VALUE clause specified in the data description entry of the data item. If the data item is a table element, the literal in the VALUE clause that corresponds to the occurrence being initialized determines the *sending-operand*. The actual *sending-operand* is a literal that, when moved to the *receiving-operand* with a MOVE statement, produces the same result as the initial value of the data item as produced by the application of the VALUE clause.

b. If the data item does not qualify as a *receiving-operand* because of the VALUE phrase, but does qualify because of the REPLACING phrase, the *sending-operand* is the *literal-1* or *identifier-2* associated with the category specified in the REPLACING phrase.

c. If the data item does not qualify in accordance with general rules 6a and 6b, the *sending-operand* is an implied figurative constant or predefined item.

The figurative sending operand used depends on the category of the receiving operand as follows:

Receiving operand	Figurative constant or predefined item
Alphabetic	Alphanumeric SPACES
Alphanumeric	Alphanumeric SPACES
(***ISQL***) Character Varying	“” (the null string)
Alphanumeric-edited	Alphanumeric SPACES
Numeric	ZEROES
Numeric-edited	ZEROES
Date	ZEROES
Time	ZEROES
Timestamp	ZEROES
Year-to-month	ZEROES
Day-to-time	ZEROES
Pointer	NULL
Indicator	NULL

(7) The order of execution of these implicit MOVE or SET statements is the order, left to right, of the appearance of each *identifier-1* in the INITIALIZE statement. Within this sequence, whenever *identifier-1* references a group data item, affected elementary data items are initialized in the sequence of their definition within the group data item. If a fixed-length table is being initialized, all occurrences are initialized. If variable-length table is being initialized, the number of occurrences initialized is the number of occurrences specified by the value of the data item referenced in the DEPENDING phrase.

(8) If *identifier-1* occupies the same storage area as *identifier-2*, the result of the execution of this statement is undefined, even if they are defined by the same data description entry. (See Page 249, Overlapping Operands.)

E.32. INSPECT

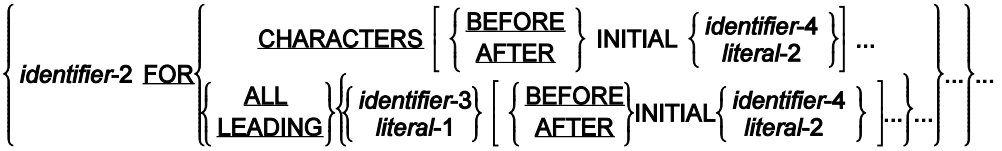
E.32.1 Function

The INSPECT statement provides the ability to tally or replace occurrences of single characters or groups of characters in a data item.

E.32.2 General Format

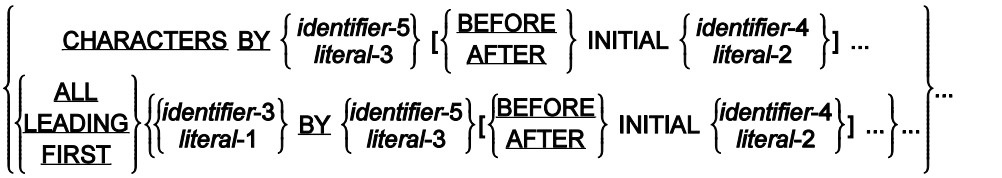
Format 1:

INSPECT *identifier-1* TALLYING



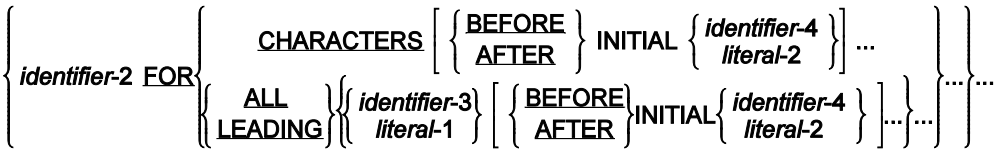
Format 2:

INSPECT *identifier-1* REPLACING

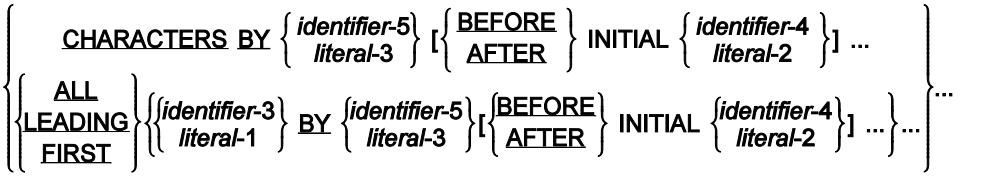


Format 3:

INSPECT *identifier-1* TALLYING



REPLACING



Format 4: (***ANSI 74*** and ***ANSI 85***)

INSPECT identifier-1 CONVERTING { *identifier-6* / *literal-4* } { *identifier-7* / *literal-5* } TO

[{ BEFORE / AFTER } INITIAL { *identifier-4* / *literal-2* }]...

E.32.3 Syntax Rules

All Formats:

(1) *Identifier-1* must reference either a group item or any category of elementary item described, implicitly or explicitly, as USAGE IS DISPLAY.

(2) *Identifier-3, ... , identifier-n* must reference an elementary item described, implicitly or explicitly, as USAGE IS DISPLAY.

(3) Each literal must be a nonnumeric literal and must not be a figurative constant that begins with the word ALL. If *literal-1, literal-2*, or *literal-4* is a figurative constant, it refers to an implicit one character data item.

(4) No more than one BEFORE phrase and one AFTER phrase can be specified for any one ALL, LEADING, CHARACTERS, FIRST, or CONVERTING phrase.

Format 1 and 3:

(5) *Identifier-2* must reference an elementary numeric data item.

Format 2 and 3:

(6) The size of *literal-3* or the data item referenced by *identifier-5* must be equal to the size of *literal-1* or the data item referenced by *identifier-3*. When a figurative constant is used as *literal-3*, the size of the figurative constant is equal to the size of *literal-1* or the size of the data item referenced by *identifier-3*.

(7) When the CHARACTERS phrase is used, *literal-2, literal-3*, or the size of the data item referenced by *identifier-4, identifier-5* must be one character in length.

Format 4:

(8) The size of *literal-5* or the data item referenced by *identifier-7* must be equal to the size of *literal-4* or the data item referenced by *identifier-6*. When a figurative constant is used as *literal-5*, the size of the figurative constant is equal to the size of *literal-4* or the size of the data item referenced by *identifier-6*.

(9) The same character must not appear more than once either in *literal-4* or in the data item referenced by *identifier-6*.

E.32.4 General Rules

All Formats:

(1) Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the left-most character position of the data item referenced by *identifier-1*, regardless of its class, and proceeds from left to right to the right-most character position as described in General Rules 5 and 6.

(2) For use in the INSPECT statement, the content of the data item referenced by *identifier-1, identifier-3, identifier-4, identifier-5, identifier-6*, or *identifier-7* will be treated as follows:

a. If any of *identifier-1, identifier-3, identifier-4, identifier-5, identifier-6* or *identifier-7* reference an alphabetic or alphanumeric data item, the INSPECT statement treats the contents of each such identifier as a character-string.

b. If any of *identifier-1, identifier-3, identifier-4, identifier-5, identifier-6* or *identifier-7* reference alphanumeric edited, numeric edited, or unsigned numeric data items, the data item is inspected as though it had been redefined as alphanumeric (see General Rule 2a) and the INSPECT statement had been written to reference the redefined data item.

c. If any of *identifier-1, identifier-3, identifier-4, identifier-5, identifier-6* or *identifier-7* reference a signed numeric data item, the data item is inspected as though it had been moved to an unsigned numeric data item with length equal to the length of the signed item excluding any separate sign position, and then the rules in General Rule 2b had been applied. (See The MOVE Statement, page 393.) If identifier-1 is a signed numeric item, the original value of the sign is retained upon completion of the INSPECT statement.

d. (***ISQL***) If *identifier-1* references a data item with usage CHARACTER VARYING, the length of the data item is evaluated only once at the beginning of the execution of the INSPECT statement. If the length evaluates to zero, there is no error and no inspection takes place. If any other identifier references a zero-length data item at the execution of the INSPECT statement, it is an error and no inspection takes place.

(3) In General Rules 5 through 17, all references to *literal-1, literal-2, literal-3, literal-4* or *literal-5* apply equally to the content of the data item referenced by *identifier-3, identifier-4, identifier-5, identifier-6* or *identifier-7* respectively.

(4) Subscripting associated with any identifier is evaluated only once as the first operation in the execution of the INSPECT statement.

Format 1 and 2:

(5) During inspection of the content of the data item referenced by *identifier-1*, each properly matched occurrence of *literal-1* is tallied (Format 1) or replaced by *literal-3* (Format 2).

(6) The comparison operation to determine the occurrence of *literal-1* to be tallied or to be replaced, occurs as follows:

a. The operands of the TALLYING or REPLACING phrase are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1 is compared to an equal number of contiguous characters, starting with the left-most character position in the data item referenced by *identifier-1*. *Literal-1* matches that portion of the content of the data item referenced by *identifier-1* if they are equal, character for character and:

1) If neither LEADING nor FIRST is specified; or

2) If the LEADING adjective applies to *literal-1* and *literal-1* is a leading occurrence as defined in General Rules 10 and 13; or

3) If the FIRST adjective applies to *literal-1* and *literal-1* is the first occurrence as defined in General Rule 13.

b. If no match occurs in the comparison of the first *literal-1*, the comparison is repeated with each successive *literal-1*, if any, until either a match is found or there is no next successive *literal-1*. When there is no next successive *literal-1*, the character position in the data item referenced by *identifier-1* immediately to the right of the left-most character position considered in the last comparison cycle is considered as the left-most character position, and the comparison cycle begins again with the first *literal-1*.

c. Whenever a match occurs, tallying or replacing takes place as described in General Rules 10 and 13. The character position in the data item referenced by *identifier-1* immediately to the right of the right-most character position that participated in the match is now considered to be the left-most character position of the data item referenced by *identifier-1*, and the comparison cycle starts again with the first *literal-1*.

d. The comparison operation continues until the right-most character position of the data item referenced by *identifier-1* has participated in a match or has been considered as the left-most character position. When this occurs, inspection is terminated.

e. If the CHARACTERS phrase is specified, an implied one character operand participates in the cycle described in paragraphs 6a through 6d above as if it had been specified by *literal-1*, except that no comparison to the content of the data item referenced by *identifier-1* takes place. This implied character is considered always to match the left-most character of the content of the data item referenced by *identifier-1* participating in the current comparison cycle.

(7) The comparison operation defined in General Rule 6 is restricted by the BEFORE and AFTER phrase as follows:

a. If neither the BEFORE nor AFTER phrase is specified, *literal-1* or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in General Rule 6. *Literal-1* or the implied operand of the CHARACTERS phrase is first eligible to participate in matching at the left-most character position of *identifier-1*.

b. If the BEFORE phrase is specified, the associated *literal-1* or the implied operand of the CHARACTERS phrase participates only in those comparison cycles which involve that portion of the content of the data item referenced by *identifier-1* from its left-most character position up to, but not including, the first occurrence of *literal-2* within the content of the data item referenced by *identifier-1*. The position of this first occurrence is determined before the first cycle of the comparison operation described in General Rule 6 is begun. If, on any comparison cycle, *literal-1* or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the content of the data item referenced by *identifier-1*. If there is no occurrence of *literal-2* within the content of the data item referenced by *identifier-1*, its associated *literal-1* or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

c. If the AFTER phrase is specified, the associated *literal-1* or the implied operand of the CHARACTERS phrase participate only in those comparison cycles which involve that portion of the content of the data item referenced by *identifier-1* from the character position immediately to the right of the right-most character position of the first occurrence of *literal-2* within the content of the data item referenced by *identifier-1* to the right-most character position of the data item referenced by *identifier-1*. This is the character position at which *literal-1* or the implied operand of the CHARACTERS phrase is first eligible to participate in matching. The position of this first occurrence is determined before the first cycle of the comparison operation described in General Rule 6 is begun. If, on any comparison cycle, *literal-1* or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the content of the data item referenced by *identifier-1*. If there is no occurrence of *literal-2* within the content of the data item referenced by *identifier-1*, its associated *literal-1* or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

Format 1:

(8) The required words ALL and LEADING are adjectives that apply to each succeeding *literal-1* until the next adjective appears.

(9) For **ANSI 85 and *VXCOBOL***, the content of the data item referenced by *identifier-2* is not initialized to zero at the beginning of the execution of the INSPECT statement. For **ANSI 74**, the tally counter (*identifier-2*) is set to zero at the beginning of the INSPECT statement. This is non-standard behavior and we recommend that you insert a "MOVE ZERO TO *identifier-2*" statement prior to the INSPECT TALLYING when using **ANSI 74**.

(10) The rules for tallying are as follows:

a. If the ALL phrase is specified, the content of the data item referenced by *identifier-2* is incremented by one for each occurrence of *literal-1* matched within the content of the data item referenced by *identifier-1*.

b. If the LEADING phrase is specified, the content of the data item referenced by *identifier-2* is incremented by one for the first and each subsequent contiguous occurrence of *literal-1* matched within the content of the data item referenced by identifier-1, provided that the left-most such occurrence is at the point where comparison began in the first comparison cycle in which *literal-1* was eligible to participate.

c. If the CHARACTERS phrase is specified, the content of the data item referenced by *identifier-2* is incremented by one for each character matched, in the sense of General Rule 6e, within the content of the data item referenced by *identifier-1*.

(11) If *identifier-1, identifier-3*, or *identifier-4* occupies the same storage area as *identifier-2*, the result of the execution of this statement is undefined, even if they are defined by the same data description entry.

Format 2:

(12) The required words ALL, LEADING, and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears.

(13) The rules for replacement are as follows:

a. When the CHARACTERS phrase is specified, each character matched, in the sense of General Rule 6e, in the content of the data item referenced by *identifier-1* is replaced by *literal-3*.

b. When the adjective ALL is specified, each occurrence of *literal-1* matched in the content of the data item referenced by *identifier-1* is replaced by *literal-3*.

c. When the adjective LEADING is specified, the first and each successive contiguous occurrence of literal-1 matched in the content of the data item referenced by *identifier-1* is replaced by *literal-3*, provided that the left-most occurrence is at the point where comparison began in the first comparison cycle in which *literal-1* was eligible to participate.

d. When the adjective FIRST is specified, the left-most occurrence of *literal-1* matched within the content of the data item referenced by *identifier-1* is replaced by *literal-3*. This rule applies to each successive specification of the FIRST phrase regardless of the value of *literal-1*.

(14) If *identifier-3, identifier-4*, or *identifier-5* occupies the same storage area as *identifier-1*, the result of the execution of this statement is undefined, even if they are defined by the same data description entry.

Format 3:

(15) A Format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written with one statement being a Format 1 statement with TALLYING phrases identical to those specified in the Format 3 statement, and the other statement being a Format 2

statement with REPLACING phrases identical to those specified in the Format 3 statement. The General Rules given for matching and counting apply to the Format 1 statement and the general rules given for matching and replacing apply to the Format 2 statement. Subscripting associated with any identifier in the Format 2 statement is evaluated only once before executing the Format 1 statement.

Format 4:

(16) A Format 4 INSPECT statement is interpreted and executed as though a Format 2 INSPECT statement specifying the same *identifier-1* has been written with a series of ALL phrases, one for each character of *literal-4*. The effect is as if each of these ALL phrases referenced, as *literal-1*, a single character of *literal-4* and referenced, as *literal-3*, the corresponding single character of *literal-5*. Correspondence between the characters of *literal-4* and the characters of *literal-5* is by ordinal position within the data item.

(17) If *identifier-4, identifier-6*, or *identifier-7* occupies the same storage area as *identifier-1*, the result of the execution of this statement is undefined, even if they are defined by the same data description entry.

E.29.5 Examples

In each of the following examples of the INSPECT statement, CNTn is assumed to be zero immediately prior to execution of the statement. The results shown for each example, except the last, are the result of executing the two successive INSPECT statements shown above them.

EXAMPLE 23. INSPECT TALLYING, REPLACING

```
INSPECT ITEM TALLYING
        CNTO FOR ALL "AB", ALL "D"
        CNT1 FOR ALL "BC"
        CNT2 FOR LEADING "EF"
        CNT3 FOR LEADING "B"
        CNT4 FOR CHARACTERS;

INSPECT ITEM REPLACING
        ALL "AB" BY "XY", "D" BY "X"
        ALL "BC" BY "VW"
        LEADING "EF" BY "TU"
        LEADING "B" BY "S"
        FIRST "G" BY "R"
        FIRST "G" BY "P"
        CHARACTERS BY "Z".
```

EXAMPLE 23. Source

Initial Value of ITEM	CNT0	CNT1	CNT2	CNT3	CNT4	Final Value of ITEM
EFABDBCGABEFGG	3	1	1	0	5	TUXYXVWRXTZZPZ
BABABC	2	0	0	1	1	SXYXYZ
BBBC	0	1	0	2	0	SSVW

EXAMPLE 23. Results

EXAMPLE 24. INSPECT TALLYING, REPLACING

```
INSPECT ITEM TALLYING
        CNTO FOR CHARACTERS
        CNT1 FOR ALL "A";

INSPECT ITEM REPLACING
        CHARACTERS BY "Z"
        ALL "A" BY "X".
```

EXAMPLE 24. source code

Initial Value of ITEM	CNT0	CNT1	Final Value of ITEM
BBB	3	0	ZZZ
ABA	3	0	ZZZ

EXAMPLE 24. results

EXAMPLE 25. INSPECT TALLYING, REPLACING

```
INSPECT ITEM TALLYING
        CNTO FOR ALL "AB" BEFORE "BC"
        CNT1 FOR LEADING "B" AFTER "D"
        CNT2 FOR CHARACTERS AFTER "A" BEFORE "C";

INSPECT ITEM REPLACING
        ALL "AB" BY "XY" BEFORE "BC"
        LEADING "B" BY "W" AFTER "D"
        FIRST "E" BY "V" AFTER "D"
        CHARACTERS BY "Z" AFTER "A" BEFORE "C".
```

EXAMPLE 25. source code

Initial Value of ITEM	CNT0	CNT1	CNT2	Final Value of ITEM
BBEABDABABBCABEE	3	0	2	BBEXYZXYXYZCABVE
ADDDDC	0	0	4	AZZZZC
ADDDDA	0	0	5	AZZZZZ
CDDDDC	0	0	0	CDDDDC
BDBBBDB	0	3	0	BDWWWDB

EXAMPLE 25. results

EXAMPLE 26. INSPECT TALLYING, REPLACING

```
INSPECT ITEM TALLYING
       CNTO FOR ALL "AB" AFTER "BA" BEFORE "BC";

INSPECT ITEM REPLACING
       ALL "AB" BY "XY" AFTER "BA" BEFORE "BC".
```

EXAMPLE 26. source code

Initial Value of ITEM	CNT0	Final Value of ITEM
ABABABABC	1	ABABXYABC

EXAMPLE 26. results

EXAMPLE 27. INSPECT CONVERTING

```
INSPECT ITEM CONVERTING
       "ABCD" TO "XYZX" AFTER QUOTE BEFORE "#".
```

EXAMPLE 27. source code

Initial Value of ITEM	Final Value of ITEM
AC"AEBDFBCD#AB"D	AC"XEYXFYZX#AB"D

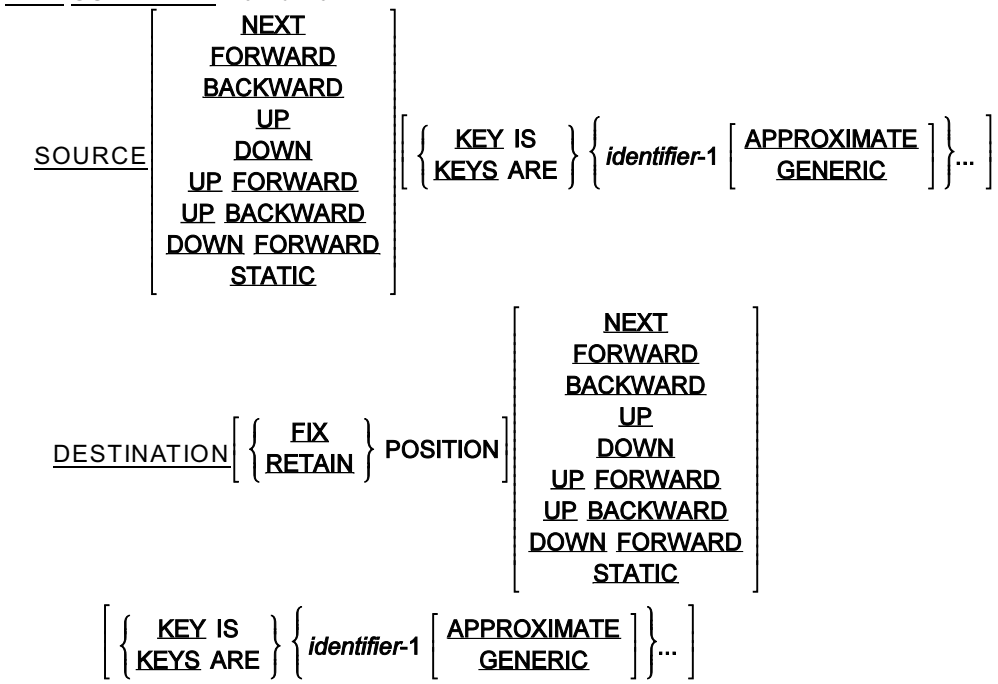
EXAMPLE 27. results

E.33. LINK SUB-INDEX (***VXCOBOL***)

E.33.1 Function

The LINK SUB-INDEX statement links a subindex to another index entry so that the subindex can be shared.

E.33.2 General Format



E.33.3 Syntax Rules

(1) *File-name* is a filename that specifies an INFOS file opened for OUTPUT or I/O and selected for ALLOW SUB-INDEX.

(2) *Identifier-1* is an alphanumeric data item that specifies a record key associated with *file-name*.

E.33.4 General Rules

(1) If the relative option and the KEY series phrase are omitted, the default is the first key in the SELECT clause.

(2) The occurrence number is not updated.

(3) FEEDBACK is not updated.

(4) KEY LENGTH is unaffected.

(5) The subindex to link is determined according to what is specified in the relative option phrase and/or the KEY series phrase in the SOURCE phrase. The link information is then transferred to the index entry specified by

the position phrase, the relative options phrase, and the KEY series phrase in the DESTINATION phrase. The DESTINATION key must not already have a subindex defined.

(6) The position phrase can only be specified in the DESTINATION phrase. FIX POSITION causes the record pointer to move from the current position to the position specified in this statement. RETAIN position causes the record position to remain at the position it was on before the execution of this statement. RETAIN is the default.

(7) The relative motion option without the KEY series phrase allows access to the index file relative to that file's current record position.

(8) Using the KEY series phrase without the relative motion option causes the key path specified to begin with the top index in the hierarchy and follow a downward motion.

(9) If the KEY series phrase is specified, each key, *identifier-1*, must be declared in the SELECT statement for *file-name*. If the relative motion option and KEY series phrase at both specified only UP, DOWN, and STATIC are allowed. The relative motion option is processed first and the key path is used.

(10) Transfer of control following the successful or unsuccessful execution of the LINK SUB-INDEX operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the LINK SUB-INDEX statement.

(11) INVALID KEY clauses on I/O statements are ONLY invoked when an Invalid Key error, as determined by a File Status of 2*x* where *x* can be any character 0 - 9 or A - Z, is generated. All other error conditions will cause the associated USE procedure, if present, as defined in the DECLARATIVES section to be executed. (See The Invalid Key Condition, page 271, for more a more comprehensive discussion.)

E.34. MERGE

E.34.1 Function

The MERGE statement combines two or more identically-sequenced files on a set of specified keys, and during the process makes records available, in merged order, to an output procedure or to an output file.

E.34.2 General Format (***ANSI 74*** and ***ANSI 85***)

```
MERGE file-name-1 { ON { ASCENDING } KEY { data-name-1 }... }...
                       { DESCENDING }
d     [ COLLATING SEQUENCE IS alphabet-name ]
      USING file-name-2 { file-name-3 }...
      { OUTPUT PROCEDURE IS procedure-name-1 [ { THROUGH } procedure-name-2 ] }
      {                                        { THRU    }                  }
      {                    GIVING { file-name-4 }...                         }
```

E.34.3 General Format (***VXCOBOL***)

```
MERGE file-name-1 { ON { ASCENDING } KEY { data-name-1 }... }...
                       { DESCENDING }
                                  { ASCII        }
                                  { NATIVE       }
      [ COLLATING SEQUENCE IS     { STANDARD-1   }                  ]
                                  { EBCDIC       }
                                  { alphabet-name }
      USING file-name-2 { file-name-3 }...
      { OUTPUT PROCEDURE IS procedure-name-1 [ { THROUGH } procedure-name-2 ]... }
      {                                      { THRU    }                      }
      {                    GIVING { file-name-4 }...                           }
```

E.34.4 Syntax Rules

(1) A MERGE statement may appear anywhere in the Procedure Division except in the declaratives portion.

(2) *File-name-1* must be described in a sort-merge file description entry in the Data Division.

(3) If the file referenced by *file-name-1* contains variable length records, the size of the records contained in the files referenced by *file-name-2* and *file-name-3* must not be less than the smallest record nor greater than the largest record described for *file-name-1*. If the file referenced by *file-name-1* contains fixed length records, the sizes of the records contained in the file referenced by *file-name-2* and *file-name-3* must not be greater than the largest record described for *file-name-1*.

(4) *Data-name-1* is a key data-name. Key data-names are subject to the following rules:

a. The data items identified by key data-names must be described in records associated with *file-name-1*.

b. Key *data-names* may be qualified.

c. Key data-names may not be described as USAGE POINTER.

d. The data items identified by key data-names must not be group items that contain variable occurrence data items.

e. If *file-name-1* has more than one record description, the data items identified by key data-names need be described in only one record description. The same character positions referenced by a key data-name in one record description entry are taken as the key in all records of the file.

f. None of the data items identified by key data-names can be described by an entry that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.

g. If a file referenced by *file-name-1* contains variable length records, all the data items identified by key data-names must be contained within the first x characters positions of the record, where x equals the minimum record size specified for the file referenced by *file-name-1*.

(5) *File-name-2*, *file-name-3*, and *file-name-4* must be described in a file description entry, not a sort-merge description entry, in the Data Division.

(6) File-names must not be repeated within the MERGE statement.

(7) No pair of file-names in a MERGE statement may be specified in the same SAME AREA, SAME SORT AREA, or SAME SORT-MERGE AREA clause. The only file-names in a MERGE statement that can be specified in the SAME RECORD AREA clause are those associated with the GIVING phrase.

(8) The words THRU and THROUGH are equivalent.

(9) *File-name-4* is subject to the following rules:

a. If *file-name-4* references an indexed file, the first specification of *data-name-1* and the data item referenced by that *data-name-1* must occupy the same character positions in its record as the data item associated with the prime record key for that file. For **ANSI 74 and ANSI 85**, the first specification of *data-name-1* must be associated with the ASCENDING phrase if *file-name-4* has a primary record key described explicitly or implicitly as VALUES ARE ASCENDING. If the key is described as VALUES ARE DESCENDING, *data-name-1* must be associated with the DESCENDING phrase. For ***VXCOBOL***, the first specification of *data-name-1* must be associated with the ASCENDING phrase.

b. For ***VXCOBOL***, if *file-name-4* references an INFOS file, it must not allow subindexing and the first specification of *data-name-1* must be associated with an ASCENDING phrase. The data-item referenced by *data-name-1* must occupy the same character positions in its record as the data item associated with the first RECORD KEY in the select for *file-name-4*, i.e., the RECORD KEY and sort key must be internal to the record.

(10) If the GIVING phrase is specified and the file referenced by *file-name-4* contains variable length records, the size of the records contained in the file referenced by *file-name-1* must not be less that the smallest record nor greater that the largest record described for *file-name-4*. If the file referenced by *file-name-4* contains fixed length records, the size of the records contained in the file referenced by *file-name-1* must not be greater that the largest record described for *file-name-4*.

(11) For ***VXCOBOL***, if *file-name-2* or *file-name-3* references INFOS files, they must not allow subindexing.

(12) *Alphabet-name* shall reference an aplhabet defined in the SPECIAL-NAMES paragraph which defines an alphnumeric collating sequence.

(13) If *file-name-2* or *file-name-3* references an indexed, INFOS, or relative file, its access mode shall be sequential or dynamic.

E.34.5 General Rules

(1) The MERGE statement merges all records contained on the file referenced by *file-name-2* and *file-name-3*.

(2) If the file referenced by *file-name-1* contains only fixed length records, any record in the file referenced by *file-name-2* or *file-name-3* containing fewer character positions that fixed length is space filled on the right beginning with the first character position after the last character in the record when that record is released to the file referenced by *file-name-1*.

(3) The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to how they are divided into KEY phrases. The leftmost data-name is the major key, the next data-name is the next most significant key, etc.

a. When the ASCENDING phrase is specified, the merged sequence will be from the lowest value of the contents of the data items identified by the key data-names to the highest value, according to the rules for comparison of operands in a relation condition.

b. When the DESCENDING phrase is specified, the merged sequence will be from the highest value of the contents of the data items identified by the key data-names to the lowest value, according to the rules for comparison of operands in a relation condition (see Relation Condition, starting on page 299).

(4) When, according to the rules for the comparison of operands in a relation condition, the contents of all key data items of one data record are equal to the corresponding key data items of one or more other data records, the order of return of these records:

a. Follows the order of the associated input files as specified in the MERGE statement.

b. Is such that all records associated with one input file are returned prior to the return of records from another input file.

(5) The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined at the beginning of the execution of the MERGE statement in the following order of precedence:

a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in that MERGE statement.

b. Second, the collating sequence established as the program collating sequence. In **ICOBOL**, this is always ASCII since the program collating sequence is ignored.

(6) The results of the merge operation are undefined unless the records in the files referenced by *file-name-2* and *file-name-3* are ordered as described in the ASCENDING or DESCENDING KEY phrases associated with the MERGE statement.

(7) All the records in the files referenced by *file-name-2* and *file-name-3* are transferred to the file referenced by *file-name-1*. At the start of the execution of the MERGE statement, the files referenced by *file-name-2* and *file-name-3* must not be in the open mode. For each of the files referenced by *file-name-2* and *file-name-3* the execution of the MERGE statement causes the following actions to be taken:

a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the INPUT phrase had been executed. If an output procedure is specified, this initiation is performed before control passes to the output procedure.

b. The logical records are obtained and released to the merge operation. Each record is obtained as if a READ statement with the NEXT and the AT END phrases had been executed.

c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. If an output procedure passes the last statement in the output procedure.

These implicit functions are performed such that any associated USE AFTER STANDARD EXCEPTION procedures are executed.

(8) The output procedure may consist of any procedure needed to select, modify, or copy records that are made available one at a time by the RETURN statement in merged order from the file referenced by *file-name-1*. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the output procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the output procedure. The range of the output procedure must not cause the execution on any MERGE, RELEASE, or SORT statement. See page 253, 306, Explicit and Implicit specifications.

(9) If an output procedure is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism at the end of the last statement in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the merge, and then passes control to the next executable statement after the MERGE statement. Before entering the output procedure, the merge procedure reaches a point at which it can select the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.

(10) During the execution of the output procedure, no statement may be executed manipulating the file referenced by or accessing the record area associated with *file-name-2* or *file-name-3*. During the execution of any USE AFTER STANDARD EXCEPTION procedure implicitly invoked while executing the MERGE statement, no statement may be executed manipulating the file referenced by, or accessing the record area associated with, *file-name-2*, *file-name-3*, or *file-name-4*.

(11) If the GIVING phrase is specified, all the merged records are written on the file referenced by *file-name-4* as the implied output procedure for the MERGE statement. At the start of execution of the MERGE statement, the file referenced by *file-name-4* must not be in the open mode. For each of the files referenced by *file-name-4*, the execution of the MERGE statement causes the following actions to be taken:

a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed.

b. The merged logical records are returned and written onto the file. Each record is written as if a WRITE statement without any optional phrases had been executed. If the file referenced by *file-name-4* is described with variable length records, the size of any record written to *file-name-4 is the size of that record when it was read from* file-name-1 , regardless of the content of the data-item referenced by the DEPENDING ON phrase of either a RECORD IS VARYING or an OCCURS clause specifiel in the file description entry for *file-name-4*.

For a relative file, the relative key date for the first record returned contains the value '1'; for the second record returned, the value '2', etc. After execution of the MERGE statement, the content of the relative key data item indicates the last record returned to the file.

c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

These implicit functions are performed such that any associated USE AFTER STANDARD EXCEPTION procedures are executed; however, the execution of such a USE procedure must not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, *file-name-4*. On the first attempt to write beyond the externally defined boundaries of the file, any USE AFTER STANDARD EXCEPTION procedure specified for that file is executed; if control is returned from that USE procedure or if no USE procedure is specified, the processing of the file is terminated as in paragraph 11c above.

(12) If the file referenced by *file-name-4* contains only fixed length records, any record in the file referenced by *file-name-1* containing fewer character positions that fixed length is space filled on the right beginning with the first character position after the last character in the record when that record is returned to the file referenced by *file-name-4*.

E.35. MOVE

E.35.1 Function

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

E.35.2 General Format

```
MOVE { identifier-1 }          TO { identifier-2 }...
     { literal      }

MOVE { CORRESPONDING }         identifier-1 TO identifier-2
     { CORR          }
```

E.35.3 Syntax Rules

(1) *Literal* or the data item referenced by *identifier-1* represents the sending area. The data item referenced by *identifier-2* represents the receiving area.

(2) CORR is an abbreviation for CORRESPONDING.

(3) When the CORRESPONDING phrase is used, all identifiers must be group items and may not be referenced modified.

(4) Neither an index data item nor Pointer data item may appear as an operand of a MOVE statement.

E.35.4 General Rules

(1) If the CORRESPONDING phrase is used, selected items within *identifier-1* are moved to selected items within *identifier-2*, according to the rules specified under the appropriate paragraph. The results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements.

(2) *Literal* or the content of the data item referenced by *identifier-1* is moved to the data item referenced by each *identifier-2* in the order in which it is specified. The rules referring to *identifier-2* also apply to the other receiving areas. Any length evaluation or subscripting associated with *identifier-2* is evaluated immediately before the data is moved to the respective data item.

If *identifier-1* has varying length (***ISQL***), is reference modified, subscripted, or is a function-identifier, the current length, reference modifier, subscript, or function-identifier is evaluated only once, immediately before data is moved to the first of the receiving operands.

The evaluation of the length of *identifier-1* or *identifier-2* may be affected by the DEPENDING ON phrase of the OCCURS clause.

(3) Any move in which the receiving operand is an elementary item and the sending operand is either a literal or an elementary item is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, numeric edited, alphanumeric edited, (***ISQL***) date, time, timestamp, year-to-month, or day-to-time. Numeric literals belong to the category numeric; nonnumeric literals belong to the category alphanumeric; (***ISQL***) date-time and interval literals belong to their respective categories. The figurative constant ZERO (ZEROS, ZEROES), when moved to a numeric or numeric edited item, belongs to the category numeric. In all other cases, it belongs to the category alphanumeric. The figurative constant SPACE (SPACES) belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.

The following rules apply to an elementary move between these categories:

a. The figurative constant SPACE, a numeric edited, an alphanumeric edited, or alphabetic data item must not be moved to a numeric, numeric edited, (***ISQL***) date-time, or interval data item.

b. A numeric literal, the figurative constant ZERO, a numeric data item, or a numeric edited data item must not be moved to an alphabetic, (***ISQL***) date-time, or interval data item.

c. A non-integer numeric literal or a non-integer numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

d. (***ISQL***) A date-time or interval literal or data item must not be moved to a data item with a category that differs from the category of the literal or data item.

e. (***ISQL***) An alphanumeric item must not be moved to a date-time or interval data-item.

f. All other elementary moves are legal and are performed according to the rules given in General Rule 4.

(4) Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for the receiving data item:

a. When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space filling takes place as previously defined.

1) If the sending operand is described as being signed numeric, the operational sign is not moved; if the operational sign occupies a separate character position, that character is not moved and the size of the sending operand is considered to be one less than its actual size in terms of standard data format characters.

2) If the sending operand is numeric edited, no de-editing takes place.

3) If the usage of the sending operand is different from that of the receiving operand, conversion of the sending operand to the internal representation of the receiving operand takes place.

4) If the sending operand is numeric and contains the PICTURE symbol `P', all digit positions specified with this symbol are considered to have the value zero and are counted in the size of the sending operand.

b. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero filling takes place as previously defined except where zeros are replaced because of editing requirements.

1) When a signed numeric item is the receiving item, the sign of the sending operand is placed in the receiving item. Conversion of the representation of the sign takes place as necessary. If the sending operand is unsigned, a positive sign is generated for the receiving item.

2) When an unsigned numeric item is the receiving item, the absolute value of the sending operand is moved and no operational sign is generated for the receiving item.

3) When the sending operand is described as being alphanumeric, data is moved as if the sending operand were described as an unsigned numeric integer.

c. When a receiving field is described as alphabetic, justification and any necessary space filling takes place as previously defined.

d. (***ISQL***) When the sending and receiving items are of category date, time or timestamp, each sub-field is treated as a simple numeric to numeric move, with any applicable alignment, zero padding, or truncation of fractional digits.

e. (***ISQL***) When the sending and receiving items are of category year-month or day-time, the value of the sending operand is normalized and any alignment, padding with zero fields, or truncation takes place as previously described.

(5) Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as noted in the OCCURS clause.

(6) The following table summarizes the legality of the various types of MOVE statements. 'Yes' means the move is legal; 'No' means it is not legal. The General Rule reference (after the slash) indicates the rule that prohibits the move or that describes the behavior of a legal move.

CATEGORY OF SENDING OPERAND	CATEGORY OF RECEIVING DATA ITEM							
	ALPHABETIC	ALPHANUMERIC EDITED ALPHANUMERIC	NUMERIC INTEGER NUMERIC NONINTEGER NUMERIC EDITED	DATE	TIME	TIMESTAMP	YEAR-TO-MONTH	DAY-TO-MONTH
ALPHABETIC	Yes/4c	Yes/4a	No/3a	No/3a	No/3a	No/3a	No/3a	No/3a
ALPHANUMERIC	Yes/4c	Yes/4a	Yes/4b	No/3e	No/3e	No/3e	No/3e	No/3e
ALPHANUMERIC EDITED	Yes/4c	Yes/4a	No/3a	No/3a	No/3a	No/3a	No/3a	No/3a
NUMERIC INTEGER	No/3b	Yes/4a	Yes/4b	No/3b	No/3b	No/3b	No/3b	No/3b
NUMERIC NONINTEGER	No/3b	No/3c	Yes/4b	No/3b	No/3b	No/3b	No/3b	No/3b
NUMERIC EDITED	No/3b	Yes/4a	No/3a	No/3b	No/3b	No/3b	No/3b	No/3b
DATE	No/3d	No/3d	No/3d	Yes/4d	No/3d	No/3d	No/3d	No/3d
TIME	No/3d	No/3d	No/3d	No/3d	Yes/4d	No/3d	No/3d	No/3d
TIMESTAMP	No/3d	No/3d	No/3d	No/3d	No/3d	Yes/4d	No/3d	No/3d
YEAR-TO-MONTH	No/3d	No/3d	No/3d	No/3d	No/3d	No/3d	Yes/4e	No/3d
DAY-TO-TIME	No/3d	No/3d	No/3d	No/3d	No/3d	No/3d	No/3d	Yes/4e

TABLE 25. Legality of Types of MOVE Statements

E.36. MULTIPLY

E.36.1 Function

The MULTIPLY statement causes numeric data items to be multiplied and sets the values of data items equal to the results.

E.36.2 General Format

Format 1:

```
MULTIPLY { identifier-1 }                BY { identifier-2 [ ROUNDED ] }...
         { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-MULTIPLY ]
```

Format 2:

```
MULTIPLY { identifier-1 }    { identifier-2 }                GIVING { identifier-3 [ ROUNDED ] }...
         { literal-1    }    { literal-2    }BY
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-MULTIPLY ]
```

E.36.3 Syntax Rules

(1) Each identifier must refer to a numeric elementary item, except that in Format 2 the identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

(2) Each literal must be a numeric literal.

(3) The composite of operands, which is the hypothetical data item resulting from the superimposition of all receiving data items of a given statement aligned on their decimal points, must not contain more than 18 digits.

E.36.4 General Rules

(1) When Format 1 is used, *literal-1* or the value of the data item referenced by *identifier-1* is stored in a temporary data item. The value in this temporary data item is multiplied by the value of the data item referenced by *identifier-2*. The value of the multiplier (the value of the data item referenced by *identifier-2*) is replaced by this product; similarly, the temporary data item is multiplied by each successive occurrence of *identifier-2* in the left-to-right order in which *identifier-2* is specified.

(2) When Format 2 is used, *literal-1* or the value of the data item referenced by *identifier-1* is multiplied by *literal-2* or the value of the data item referenced by *identifier-2* and the result is stored in the data items referenced by *identifier-3*.

(3) Additional rules and explanations relative to this statement are given under the appropriate paragraphs, (See Scope of Statements, page 253; The ROUNDED Phrase, page 245; The ON SIZE ERROR Phrase, page 246; The Arithmetic Statements, page 249; Overlapping Operands, page 249; and Multiple Results in Arithmetic Statements, page 249.)

E.37. OPEN

E.37.1 Function

The OPEN statement initiates the processing of files.

E.37.2 General Format (***ANSI 74*** and ***ANSI 85***)

For sequential files:

```
                                  { { INPUT filename [ REVERSED ] }...
                                  {                  [ WITH NO REWIND ] }
OPEN [ EXCLUSIVE ] { { OUTPUT filename [ WITH NO REWIND ]}...  }...
                                  {                                  }
                                  {                                  }
                                  {                                  }
                                  {                                  }
                                  {                                  }
```

For relative and indexed files:

```
                                  { INPUT filename...  }
                                  { OUTPUT filename...  }
OPEN [ EXCLUSIVE ] { I-O filename...  }...
                                  { EXTEND filename...  }
```

E.37.3 General Format (***VXCOBOL***)

```
OPEN [ EXCLUSIVE ]
{ INPUT [ SEQUENTIAL ] { filename [ WITH NO REWIND ] [ ONLY ] }...
                                                     [ EXCLUDE identifier... ] }
{ OUTPUT [ INDEX ] { filename [ WITH { VERIFY } ] [ ONLY ] }...
                                     { NO REWIND } ] [ EXCLUDE identifier... ] }
{ I-O { filename [ WITH VERIFY ] [ ONLY ] }...
                                 [ EXCLUDE identifier... ] }
{ EXTEND { filename [ WITH VERIFY ] [ ONLY ] }...
                                    [ EXCLUDE identifier... ] }
}...
```

E.37.4 Syntax Rules

(1) The files referenced in the OPEN statement need not all have the same organization or access.

(2) For **ANSI 74**, the EXTEND phrase must only be used for sequential files.

(3) For **ANSI 85**, the EXTEND phrase must only be used for files in the sequential access mode.

(4) For ***VXCOBOL***, the EXTEND phrase must only be used for sequential files, INFOS files, or files in sequential access mode.

(5) The WITH NO REWIND, REVERSED, WITH VERIFY, ONLY, and EXCLUDE clauses are for documentation purposes only.

(6) Filename may not be a sort/merge file.

(7) The EXTEND phrase must only be used for files for which the LINAGE clause has not been specified.

E.37.5 General Rules

(1) The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode. The successful execution of an OPEN statement associates the file with the *filename* through the file connector.

Once the filename is processed the OPEN statement checks to see if the given file is physically present and is recognized by the input-output control system. and follows the rules as outlined in the following table.

The three tables below show the results of opening available and unavailable files for ***ANSI 74***, ***ANSI 85***, and ***VXCOBOL***.

	File is Available	File is Unavailable
INPUT	Normal open	Open is unsuccessful
I-O	Normal open	For sequential, Open is unsuccessful For relative and indexed, Open causes the file to be created, **NOT ANSI STANDARD**
OUTPUT	For sequential, Normal open; the file contains no records For relative and indexed, Normal open, **NOT ANSI STANDARD**	Open causes the file to be created
EXTEND (sequential only)	Normal open	Open causes the file to be created

TABLE 26. Availability of a File (***ANSI 74***)

	File is Available	File is Unavailable
INPUT	Normal open	Open is unsuccessful
INPUT (optional)	Normal open	Normal open; the first READ causes the at end or invalid key condition
I-O	Normal open	Open is unsuccessful
I-O (optional)	Normal open	Open causes the file to be created
OUTPUT	Normal open; the file contains no records	Open causes the file to be created
EXTEND	Normal open	Open is unsuccessful
EXTEND (optional)	Normal open	Open causes the file to be created

TABLE 27. Availability of a File (***ANSI 85***)

	File is Available	File is Unavailable
INPUT	Normal open	Open is unsuccessful
INPUT (optional)	Normal open	Normal open; the first READ causes the at end or invalid key condition
INPUT SEQUENTIAL (INFOS)	Normal open	Open is unsuccessful
I-O	Normal open	Open is unsuccessful
OUTPUT	For ICISAM and INFOS, files-Open is unsuccessful For others-Open is unsuccessful unless compiled with the ANSI switch (-G a) in which case Open is successful to an empty file	Open causes the file to be created
OUTPUT INDEX (INFOS)	Open is unsuccessful	Open causes the file to be created
EXTEND	Normal open	Open is unsuccessful

TABLE 28. Availability of a File (***VXCOBOL***)

(2) The successful execution of an OPEN statement makes the associated record area available to the program.

(3) When a file is not in an open mode, no statement may be executed which references the file, either explicitly or implicitly, except for a MERGE statement with the USING or GIVING phrase, an OPEN statement, or a SORT statement with the USING or GIVING phrase..

(4) An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In the Permissible Statements table below, `X' at an intersection indicates that the specified statement may be used with the open mode given at the top of the column.

File Access Mode	Statement	OPEN MODE			
		Input	Output	I-O	Extend
Sequential	READ	X		X	
	WRITE		X		X
	REWRITE	X		X	
	START			X	
	DELETE			X	
	UNDELETE			X	
Random	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
	UNDELETE			X	
Dynamic	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	
	UNDELETE			X	
All	(***VXCOBOL***:)				
	DEFINE SUB-INDEX		X	X	
	EXPUNGE SUB-INDEX		X	X	
	LINK SUB-INDEX		X	X	
	RETRIEVE	X		X	

TABLE 29. Permissible Statements

(5) A file may be opened with the INPUT, OUTPUT, EXTEND, and I-O phrases in the same run unit. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for that same file must be preceded by the execution of a CLOSE statement, without the LOCK phrase, for that file.

(6) Execution of the OPEN statement does not obtain or release the first data record.

(7) If during the execution of an OPEN statement a file attribute conflict condition occurs, the execution of the OPEN statement is unsuccessful.

(8) If a file opened with the INPUT phrase is an optional file which is not present, the OPEN statement sets the file position indicator to indicate that an optional input file is not present.

(9) When files are opened with the INPUT or I-O phrase, the file position indicator is set to the first record for sequential files, 1 for relative files, and to the first record using the primary key for indexed files.

(10) When the EXTEND phrase is specified, the OPEN statement positions the file immediately after the last logical record for that file. The last logical record for a sequential file is the last record written in the file. The last logical record for a relative file is the currently existing record with the highest relative record number. The last logical record for an indexed file is the currently existing record with the highest primary key.

(11) The OPEN statement with the I-O phrase must reference a file that supports the input and output operations that are permitted for a file when opened in the I-O mode. The execution of the OPEN statement with the I-O phrase places the referenced file in the open mode for both input and output operations.

(12) For ***ANSI 74***, for a file that is unavailable, the successful execution of an OPEN statement with an EXTEND or I-O phrase creates the file. This creation takes place as if the following statements were executed in the order shown:

```
OPEN OUTPUT file-name.
CLOSE file-name.
```

These statements are followed by execution of the OPEN statement specified in the source program.

The successful execution of an OPEN statement with the OUTPUT phrase creates the file. After the successful creation of a file, that file contains no data records.

(13) For ***ANSI 85***, for an optional file that is unavailable, the successful execution of an OPEN statement with an EXTEND or I-O phrase creates the file. This creation takes place as if the following statements were executed in the order shown:

```
OPEN OUTPUT file-name.
CLOSE file-name.
```

These statements are followed by execution of the OPEN statement specified in the source program.

The successful execution of an OPEN statement with the OUTPUT phrase creates the file. After the successful creation of a file, that file contains no data records.

OPTIONAL is specified in the File Control SELECT clause.

(14) For ***VXCOBOL***, for a file that is unavailable, the execution of an OPEN statement with an EXTEND or I-O phrase is unsuccessful.

(15) The execution of the OPEN statement causes the value of the I-O status (and, for ***VXCOBOL***, the INFOS status) associated with *filename* to be updated.

(16) If more than one *filename* is specified in an OPEN statement, the result of executing this OPEN statement is the same as if a separate OPEN statement had been written for each file-name in the same order as specified in the OPEN statement.

(17) The minimum and maximum record sizes for a file are established at the time the file is created and must not subsequently be changed.

(18) The EXCLUSIVE phrase is an extension to ANSI COBOL that specifies that for each file in the OPEN statement, the current program is the only program that will be allowed to open the file, and the program can have the file open on a single file connector. If any other **ICOBOL** program already has the file open, the OPEN statement will fail. On some systems, the open will fail if any other program on the system (not just COBOL programs) has the file open.

For ***VXCOBOL***:

(19) Opening an INFOS file will automatically perform a DOWN motion positioning the file position indicator before the first key in the top level index (U/FOS positions the file position indicator above the top level index.) if the access mode is sequential or dynamic. The downward motion is not done if the access mode is RANDOM.

(20) OPEN INPUT SEQUENTIAL could improve the performance of sequential reads thru INFOS II indexes, however the SEQUENTIAL phrase is ignored when using U/FOS files.

(21) OPEN OUTPUT INDEX is used to create an additional index for an INFOS file. (The additional index is frequently referred to as an inversion of the file.) The index named by the ASSIGN INDEX clause of the SELECT must not exist and the database named by the ASSIGN DATA clause (or implied) must exist.

(22) INFOS files can be created with the OPEN OUTPUT phrase, but it is recommended that they be created with an external utility. U/FOS files cab be created with the ufos_create utility. This utility provides more complete access to the options available for the file.

NOTES:

(1) Files opened for OUTPUT, EXTEND, or I-O must not have the Read-Only attribute set, else the OPEN fails with a File Status 92.

(2) **On UNIX**, for OPEN OUTPUT to a sequential file that already exists, the file is opened with the UNIX truncate option, which sets the filesize to 0. This is equivalent to the COBOL behavior of deleting and recreating the file. This method is used to properly maintain UNIX hard links to the name.

(3) **ICOBOL** supports Indexed and Relative versions 5, 6, and 7. An OPEN of a file that exists will automatically adjust for the version of the file. An OPEN of a new file will create the file version 7. A particular version can be specified under programmer control by using the "v=5|6|7" option in an extended disk open.

(4) **On UNIX**, for systems supporting symbolic links, OPEN will always open the resolution file.

(5) For **ANSI 74 and ANSI 85**, OPEN with ASSIGN TO PRINTER or PRINTER-1 including a filename with the Printer Control utility enabled in the configuration file (.cfi) will place the file in the printer control file to be printed if the given queue was enabled. If the given filename is a simple name (i.e., no path specifier), the file will be created in the printer control directory. ASSIGN TO PRINTER will place the file in the queue directed to @PCQ0 and ASSIGN TO PRINTER-1 will place the file in the queue directed to @PCQ1. If the appropriate PCQ has the AUTO option enabled, then when the file is closed by the COBOL program the file will automatically start printing using the default options specified for that PCQ.

The printer control file has a limit of 48 to 1024 files before subsequent OPENs will fail with a File Status 99 if a new file is to be added to the print queue.

(6) OPEN EXTEND does not imply EXCLUSIVE. If EXCLUSIVE access is desired, it should be explicitly specified on the OPEN statement.

E.38. PERFORM

E.38.1 Function

The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete. The PERFORM statement is also used to control execution of one or more imperative statements which are within the scope of that PERFORM statement.

E.38.2 General Format (**ANSI 74 and ANSI 85**)

Format 1: Unconditional PERFORM

Out-of-line

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ]
                           { THRU    }
```

In-line

```
PERFORM imperative-statement-1 END-PERFORM
```

Format 2: Iterative PERFORM

Out-of-line

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ] { identifier-1 } TIMES
                           { THRU    }                    { literal-1    }
```

In-line

```
PERFORM { identifier-1 } TIMES imperative-statement-1
        { literal-1    }
END-PERFORM
```

Format 3: Conditional PERFORM

Out-of-line

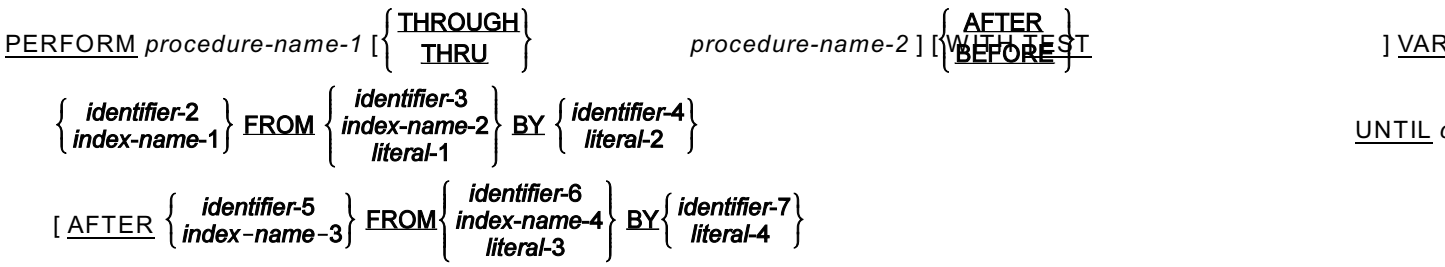
```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ]
                           { THRU    }
[ WITH TEST { AFTER  } ] UNTIL condition-1
            { BEFORE }
```

In-line

```
PERFORM [ WITH TEST { AFTER  } ] UNTIL condition-1 imperative-statement-1 END-PERFORM
                    { BEFORE }
```

Format 4: Variable PERFORM

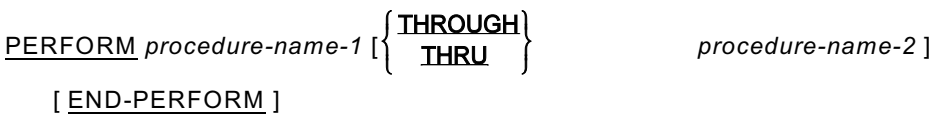
Out-of-line



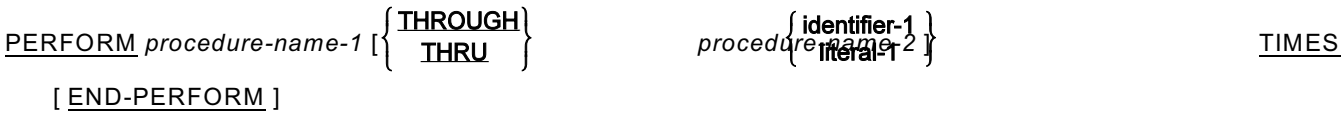
In-line

E.38.3.General Formats (***VXCOBOL***)

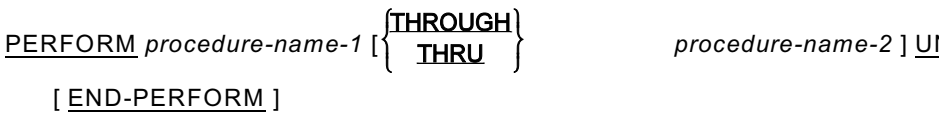
Format 1: Unconditional PERFORM



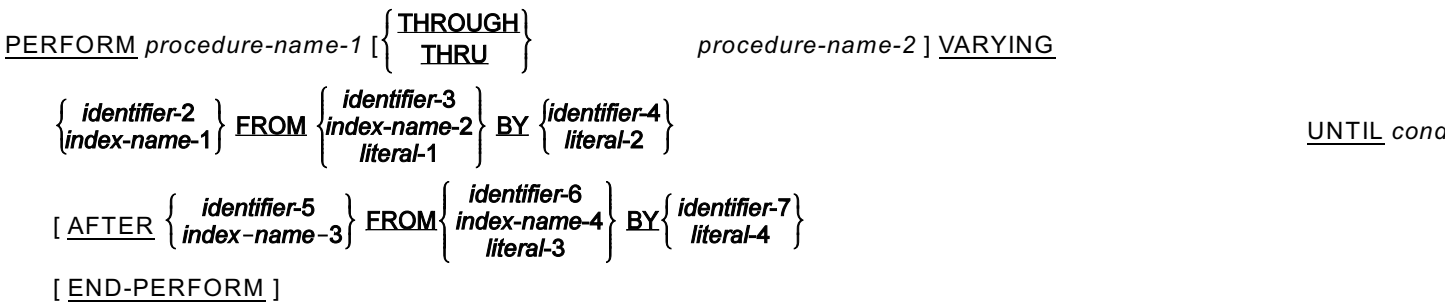
Format 2: Iterative PERFORM



Format 3: Conditional PERFORM



Format 4: Variable PERFORM



E.38.4 Syntax Rules

(1) Each identifier represents a numeric elementary item described in the Data Division. In Format 2, *identifier-1* must be described as a numeric integer.

(2) If neither the TEST BEFORE nor TEST AFTER phrase is specified, the TEST BEFORE is assumed. For ***VXCOBOL***, TEST BEFORE is always assumed.

(3) Each literal represents a numeric literal.

(4) The words THROUGH and THRU are equivalent.

(5) If an index-name is specified in the VARYING or AFTER phrase, then:

a. The identifier in the associated FROM and BY phrases must reference an integer data item.

b. The literal in the associated FROM phrase must be a positive integer.

c. The literal in the associated BY phrase must be a nonzero integer.

(6) If an index-name is specified in the FROM phrase, then:

a. The identifier in the associated VARYING or AFTER phrase must reference an integer data item.

b. The identifier in the associated BY phrase must reference an integer data item.

c. The literal in the associated BY phrase must be an integer.

(7) Literal in the BY phrase must not be zero.

(8) *Condition-1*, *condition-2*, ... , may be any conditional expression.

(9) Where *procedure-name-1* and *procedure-name-2* are both specified and either is the name of a procedure in the declaratives portion of the Procedure Division, both must be procedure-names in the same declarative section.

(10) Six AFTER phrases are permitted in Format 4 of the out-of-line PERFORM statement.

(11) For ***VXCOBOL***, the END-PERFORM is for documentation purposes only.

E.38.5 General Rules

(1) When *procedure-name-1* is specified, the PERFORM statement is referred to as an out-of-line PERFORM statement; when *procedure-name-1* is omitted, the PERFORM statement is referred to as an in-line PERFORM statement. In-line PERFORM statements are not supported for ***VXCOBOL***.

(2) The data items referenced by *identifier-4* and *identifier-7* must not have a zero value.

(3) If an index-name is specified in the VARYING or AFTER phrase, and an identifier is specified in the associated FROM phrase, the data item referenced by the identifier must have a positive value.

(4) The statements contained within the range of *procedure-name-1* (through *procedure-name-2* if specified) for an out-of-line PERFORM statement or contained within the PERFORM statement itself for an in-line PERFORM statement are referred to as the specified set of statements.

(5) The END-PERFORM phrase delimits the scope of the in-line PERFORM statement.

(6) An in-line PERFORM statement functions according to the following general rules for an otherwise identical out-of-line PERFORM statement, with the exception that the statements contained within the in-line PERFORM statement are executed in place of the statements contained within the range of *procedure-name-1* (through *procedure-name-2* if specified). Unless specially qualified by the word in-line or out-of-line, all the general rules which apply to the out-of-line PERFORM statement also apply to the in-line PERFORM statement.

(7) When the PERFORM statement is executed, control is transferred to the first statement of the specified set of statements (except as indicated in general rules 10b, 10c, and 10d). This transfer of control occurs only once for each execution of a PERFORM statement. For those cases where a transfer of control to the specified set of statements does take place, an implicit transfer of control to the end of the PERFORM statement is established as follows:

a. If *procedure-name-1* is a paragraph-name and *procedure-name-2* is not specified, the return is after the last statement of *procedure-name-1*.

b. If *procedure-name-1* is a section-name and *procedure-name-2* is not specified, the return is after the last statement of the last paragraph in *procedure-name-1*.

c. If *procedure-name-2* is specified and it is a paragraph-name, the return is after the last statement of the paragraph.

d. If *procedure-name-2* is specified and it is a section-name, the return is after the last statement of the last paragraph in the section.

e. If an in-line PERFORM statement is specified, an execution of the PERFORM statement is completed after the last statement contained within it has been executed.

(8) There is no necessary relationship between *procedure-name-1* and *procedure-name-2* except that a consecutive sequence of operations is to be executed beginning at the procedure named *procedure-name-1* and ending with the execution of the procedure named *procedure-name-2*. In particular, GO TO and PERFORM statements may occur between *procedure-name-1* and the end of *procedure-name-2*. If there are two or more logical paths to the return point, then *procedure-name-2* may be the name of a paragraph consisting of the EXIT statement, to which all of these paths must lead.

(9) If control passes to the specified set of statements by means other than a PERFORM statement, control will pass through the last statement of the set to the next executable statement as if no PERFORM statement referenced the set.

(10) The PERFORM statements operate as follows:

a. Format 1 is the basic PERFORM statement. The specified set of statements referenced by this type of PERFORM statement is executed once and then control passes to the end of the PERFORM statement.

b. Format 2 is the PERFORM ... TIMES. The specified set of statements is performed the number of times specified by *integer-1* or by the initial value of the data item referenced by *identifier-1* for that execution. If at the time of the execution of a PERFORM statement, the value of the data item referenced by *identifier-1* is equal to zero or is negative, control passes to the end of the PERFORM statement. Following the execution of the specified set of statements the specified number of times, control is transferred to the end of the PERFORM statement.

During execution of the PERFORM statement, reference to *identifier-1* cannot alter the number of times the specified set of statements is to be executed from that which was indicated by the initial value of the data item referenced by *identifier-1*.

See Appendix A, Implementation Limits on page 817, for the maximum number **ICOBOL** currently supports for an interative PERFORM (i.e., PERFORM n TIMES) and for the maximum number of active PERFORMs.

c. Format 3 is the PERFORM ... UNTIL. The specified set of statements is performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the end of the PERFORM statement. If the condition is true when the PERFORM statement is entered, and test TEST BEFORE phrase is specified or implied no transfer to *procedure-name-1* takes place, and control is passed to the end of the PERFORM statement. If the TEST AFTER phrase is specified, the PERFORM statement functions as if the TEST BEFORE phrase was specified except that the condition is tested after the specified set of statements has been executed. Any subscripting associated with the operands specified in *condition-1* is evaluated each time the condition is tested.

d. Format 4 is the PERFORM ... VARYING. This variation of the PERFORM statement is used to augment the values referenced by one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion, every reference to identifier as the object of the VARYING, AFTER, and FROM (current value) phrases also refers to index-names. If *index-name-1* or *index-name-3* is specified, the value of the associated index at the beginning of the PERFORM statement must be set to an occurrence number of an element in the table. If *index-name-2* or *index-name-4* is specified, the value of the data item referenced by *identifier-2* or *identifier-5* at the beginning of the PERFORM statement must be equal to an occurrence number of an element in a table associated with *index-name-2* or *index-name-4*. Subsequent augmentation, as described below, of *index-name-1* or *index-name-3* must not result in the associated index being set to a value outside the range of the table associated with *index-name-1* or *index-name-3*; except that, at the completion of the PERFORM statement, the index associated with *index-name-1* may contain a value that is outside the range of the associated table by one increment or decrement value. If *identifier-2* or *identifier-5* is subscripted, the subscripts are evaluated each time the content of the data item referenced by the identifier is set or augmented. If *identifier-3*, *identifier-4*, *identifier-6*, or *identifier-7* is subscripted, the subscripts are evaluated each time the content of the data item referenced by the identifier is used in a setting or augmenting operation. Any subscripting associated with the operands specified in *condition-1* or *condition-2* is evaluated each time the condition is tested.

Representation of the actions of several types of Format 4 PERFORM statements are given in figures 5 and 6 on the following pages.

1) If the TEST BEFORE phrase is specified or implied:

When the data item referenced by one identifier is varied, the content of the data item referenced by *identifier-2* is set to *literal-1* or the current value of the data item referenced by *identifier-3* at the point of initial execution of the PERFORM statement; then, if the condition of the UNTIL phrase is false, the specified set of statements is executed once. The value of the data item referenced by *identifier-2* is augmented by the specified increment or decrement value (*literal-2* or the value of the data item referenced by *identifier-4*) and *condition-1* is evaluated again. The cycle continues until this condition is true, at which point control is transferred to the end of the PERFORM statement. If *condition-1* is true at the beginning of execution of the PERFORM statement, control is transferred to the end of the PERFORM statement.

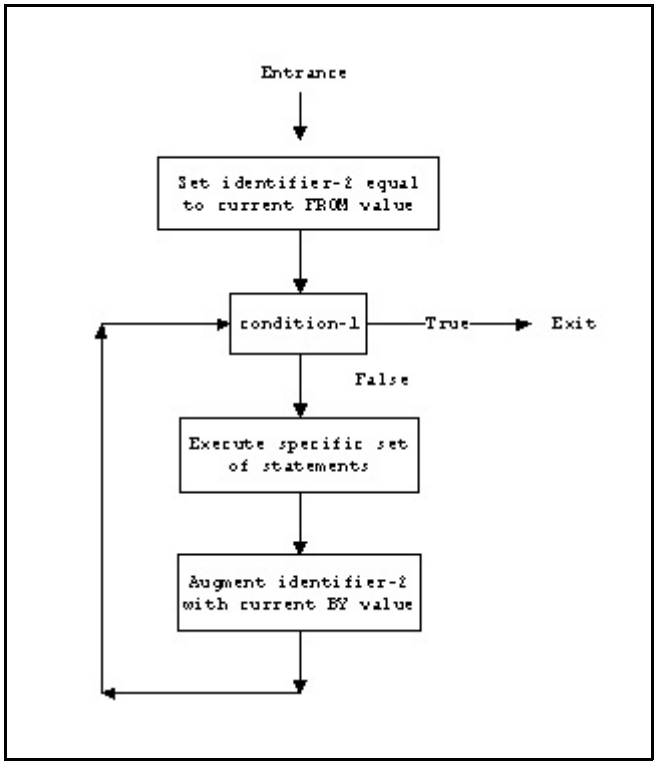


FIGURE 5. PERFORM [TEST BEFORE] VARYING with one condition

When the data items referenced by two identifiers are varied, the content of the data item referenced by *identifier-2* is set to *literal-1* or the current value of the data item referenced by *identifier-3* and then the content of the data item referenced by *identifier-5* is set to *literal-3* or the current value of the data item referenced by *identifier-6*. After the contents of the data items referenced by the identifiers have been set, *condition-1* is evaluated; if true, control is transferred to the end of the PERFORM statement; if false, *condition-2* is evaluated. If *condition-2* is false, the specified set of statements is executed once, then the content of the data item referenced by *identifier-5* is augmented by *literal-4* or the content of the data item referenced by *identifier-7* and *condition-2* is evaluated again. This cycle of evaluation and augmentation continues until this condition is true. When *condition-2* is true, the content of the data item referenced by *identifier-2* is augmented by *literal-2* or the content of the data item referenced by *identifier-4*, the content of the data item referenced by *identifier-5* is set to *literal-3* or the current value of the data item referenced by *identifier-6*, and *condition-1* is reevaluated. The PERFORM statement is completed if *condition-1* is true; if not, the cycle continues until *condition-1* is true.

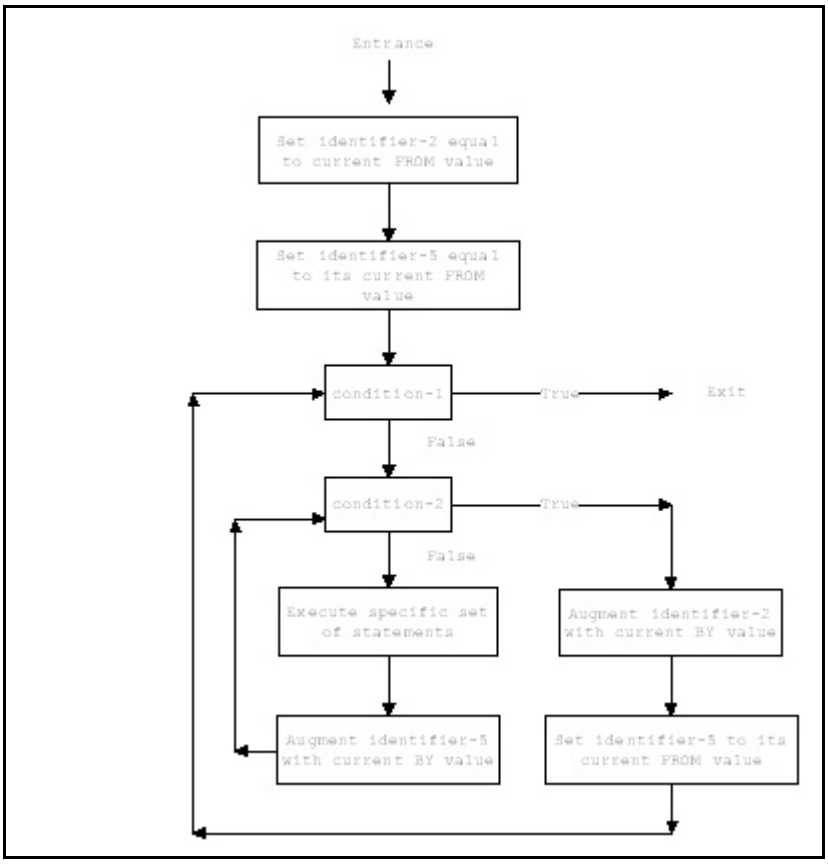


FIGURE 6. PERFORM [TEST BEFORE] VARYING with two conditions

At the termination of the PERFORM statement, the data item referenced by *identifier-5* contains *literal-3* or the current value of the data item referenced by *identifier-6*. The data item referenced by *identifier-2* contains a value that exceeds the last used setting by one increment or decrement value, unless *condition-1* was true when the PERFORM statement was entered, in which case, the data item referenced by *identifier-2* contains *literal-1* or the current value of the data item referenced by *identifier-3*.

2) For **ANSI 74 and ANSI 85**, if the TEST AFTER phrase is specified:

When the data item referenced by one identifier is varied, the content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3 at the point of execution of the PERFORM statement; then the specified set of statements is executed and condition-1 of the UNTIL phrase is tested. If the condition is false, the value of the data item referenced by identifier-2 is augmented by the specified increment or decrement value (literal-2 or the value of the data item referenced by identifier-4) and the specified set of statements is executed again. The cycle continues until condition-1 is tested and found to be true, at which point control is transferred to the end of the PERFORM statement.

When the data item referenced by two identifiers are varied, the content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3, then the content of the data item referenced by identifier-5 is set to literal-3 or the current value of the data item referenced by identifier-6 and the specified set of statements is executed. Condition-2 is then evaluated; if false, the content of the data item referenced by identifier-5 is augmented by literal-4 or the content of the data item referenced by identifier-7 and the specified set of statements is again executed. The cycle continues until condition-2 is again evaluated and found to be true, at which time condition-1 is evaluated. If the condition is false, the value of the data item referenced by identifier-2 is augmented by the specified increment or decrement value (literal-2 or the value of the data item referenced by identifier-4), the content of the data item referenced by identifier-5 is set to literal-3 or the current value of the data item referenced by identifier-6 and the specified set of statements is executed again. The cycle

continues until condition-1 is tested and found to be true, at which point control is transferred to the end of the PERFORM statement.

After the completion of the PERFORM statement, each data item varied by an AFTER or VARYING phrase contains the same value it contained at the end of the most recent execution of the specified set of statements.

During the execution of the specified set of statements associated with the PERFORM statement, any change to the VARYING variable (the data item referenced by *identifier-2* and *index-name-1*), the BY variable (the data item referenced by *identifier-4*), the AFTER variable (the data item referenced by *identifier-5* and *index-name-3*), or the FROM variable (the data item referenced by *identifier-3* and *index-name-2*) will be taken into consideration and will affect the operation of the PERFORM statement.

When the data items referenced by two identifiers are varied, the data item referenced by *identifier-5* goes through a complete cycle (FROM, BY, UNTIL) each time the content of the data item referenced by *identifier-2* is varied. When the contents of three or more data items referenced by identifiers varied, the mechanism is the same as for two identifiers except that the data item being varied by each AFTER phrase goes through a complete cycle each time the data item being varied by the preceding AFTER phrase is augmented.

(11) The range of a PERFORM statement consists logically of all those statements that are executed as a result of executing the PERFORM statement through execution of the implicit transfer of control to the end of the PERFORM statement. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the PERFORM statement, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the source program.

(12) Statements executed as the result of a transfer of control caused by executing an EXIT PROGRAM statement are not considered to be part of the range of the PERFORM statement when:

a. That EXIT PROGRAM statement is specified in the same program in which the PERFORM statement is specified, and

b. The EXIT PROGRAM statement is within the range of the PERFORM statement.

(13) Statements in other programs in the run unit may only be obeyed as a result of executing a PERFORM statement, if the range of that PERFORM statement includes CALL and EXIT PROGRAM statements.

(14) If the range of a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit. See the following illustrations for examples of legal PERFORM constructs:

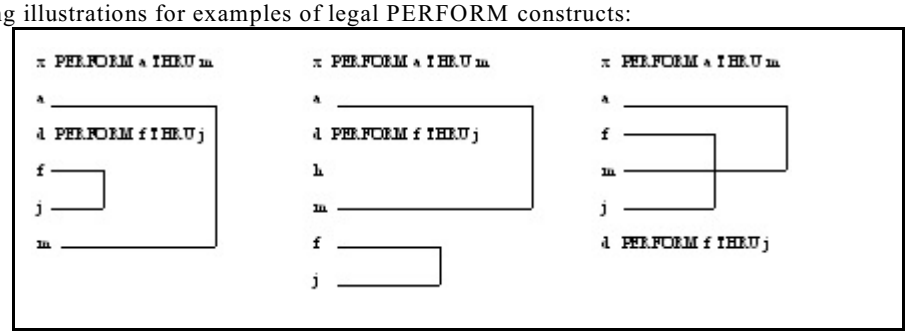


FIGURE 7. Valid PERFORM constructs

E.39. PREPARE (***ISQL***)

E.39.1 Function

The PREPARE statement prepares an SQL statement for subsequent execution by the EXECUTE statement.

E.39.2 General Format

PREPARE { *identifier-1* / *literal-1* } { *identifier-2* / *literal-2* } FROM

[ON SQLERROR *imperative-statement-1*]
[NOT ON SQLERROR *imperative-statement-2*]
[END-PREPARE]

E.39.3 Syntax Rules

(1) *Literal-1* and *literal-2* must specify a nonnumeric literal and may not specify a figurative constant.

(2) *Identifier-1* and *identifier-2* must specify an alphanumeric data item.

(3) *Literal-1* or the value represented by *identifier-1* may not exceed 30 characters in length.

E.39.4 General Rules

(1) *Literal-1* or the content of the data item represented by *identifier-1* specifies the name of a statement container at runtime. The statement container holds the result of the statement preparation process that is performed when the PREPARE statement is executed. The content of the statement container is subsequently used by an EXECUTE statement to perform the SQL operation.

(2) *Literal-2* or the content of the data item represented by *identifier-2* specifies the text of the SQL statement that is to be prepared for execution.

(3) Statement containers are considered to be local to the currently active connection, regardless of the program containing the PREPARE statement that allocates them. Therefore a statement can be prepared in one program and executed in a separate program.

(4) The following SQL statements may be specified as part of *literal-2* or the content of the data item represented by *identifier-2*:

- CREATE TABLE and CREATE INDEX
- DECLARE CURSOR
- DELETE
- DROP TABLE and DROP INDEX
- SELECT
- INSERT
- UPDATE

Additional information on the syntax for these supported statements can be found in the chapter on the ICODBC Driver found on page 771.

(5) If there is no currently active connection in the run unit, the execution of the PREPARE statement will result in an error with a SQLSTATE of "HY010", which is a "Function sequence error".

(6) If a statement container by the specified name already exists in the currently active connection at the time the PREPARE statement is executed, the existing content of the statement container is discarded and the container is reused for the execution of this PREPARE statement.

(7) If a statement container by the specified name does not already exist in the currently active connection at the time the PREPARE statement is executed, a new statement container with the given name is allocated for the currently active connection.

(8) Upon completion of the PREPARE statement, the following occurs in the order specified:

a. The value of the SQLSTATE data item is updated with the status of the operation.

b. If the value of the SQLSTATE class field is “00" or “01", the statement is successful. Control is transferred to the end of the PREPARE statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the PREPARE statement.

c. If the value of the SQLSTATE class field is not “00" or “01", the statement is unsuccessful. The statement container is deallocated and no statement container of the specified name will exist in the current program. Control is transferred to the end of the PREPARE statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the PREPARE statement.

(9) The END-PREPARE phrase delimits the scope of the PREPARE statement.

(10) More on SQLSTATE can be found on page 133.

E.40. READ (***ANSI 74*** and ***ANSI 85***)

E.40.1 Function

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a mass storage file. LOCK and IGNORE LOCK are extensions to ANSI COBOL. TIME-OUT is an extension to ANSI COBOL.

E.40.2 General Format

Format 1:

For sequential files:

READ *file-name* [NEXT] RECORD [INTO *identifier-1*] [TIME-OUT AFTER { *identifier-2* / *literal-1* }]

[AT END *imperative-statement-1*]
[NOT AT END *imperative-statement-2*]
[END-READ]

For indexed and relative files:

READ *file-name* [NEXT / PREVIOUS] [WITH { LOCK / IGNORE LOCK }] RECORD [INTO *ident*

[AT END *imperative-statement-1*]
[NOT AT END *imperative-statement-2*]
[END-READ]

Format 2:

For relative files:

READ *file-name* RECORD [WITH { LOCK / IGNORE LOCK }] [INTO *identifier-1*]

[INVALID KEY *imperative-statement-1*]
[NOT INVALID KEY *imperative-statement-2*]
[END-READ]

For indexed files:

READ *file-name* RECORD [WITH { LOCK / IGNORE LOCK }] [INTO *identifier-1*] [KEY IS *key-name*]

[INVALID KEY *imperative-statement-1*]
[NOT INVALID KEY *imperative-statement-2*]
[END-READ]

E.40.3 Syntax Rules

(1) The storage area associated with *identifier-1* and the record area associated with *file-name* must not be the same storage area.

(2) Format 1 must be used for all files in sequential access mode.

(3) *Identifier-2* may represent any elementary numeric data item. *Literal-1* may be any numeric literal.

(4) In Format 1, the NEXT or PREVIOUS phrase must be specified for files in dynamic access mode when records are to be retrieved sequentially.

(5) Format 2 is used for indexed and relative files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

(6) The INVALID KEY phrase or the AT END phrase must be specified, if no applicable USE AFTER STANDARD EXCEPTION procedure is specified for *file-name*.

For indexed files:

(7) The KEY IS phrase of the READ statement must reference a key-name (*id-1* in the formats of the RECORD KEY or ALTERNATE RECORD KEY) associated with *file-name*.

(8) *Key-name* may be qualified if *id-1* is a simple data item. *Key-name* may be qualified by the filename if it is a composite data item.

E.40.4 General Rules

(1) The file referenced by *file-name* must be open in the input or I-O mode at the time this statement is executed.

(2) In format 1, if neither the NEXT phrase nor the PREVIOUS phrase is specified, then NEXT is implied for files in sequential access mode.

(3) The execution of the READ statement causes the value of the I-O status associated with *file-name* to be updated.

(4) The setting of the file position indicator at the start of the execution of a Format 1 READ statement is used in determining the record to be made available according to the following rules. Comparisons for records in sequential files relate to the record number. Comparisons for records in relative files relate to the relative key number. Comparisons for records in indexed files relate to the value of the current key of reference. For indexed files, the comparisons are made according to the collating sequence of the file.

a. If the file position indicator indicates that no valid next record has been established, execution of the READ statement is unsuccessful.

b. If the file position indicator was established by a previous OPEN or START statement, the first existing record that is selected is either:

1. If NEXT is specified or implied, the first existing record in the file whose record number or key value is greater than or equal to the file position indicator, or

2. If PREVIOUS is specified, the first existing record in the file whose record number or key value is less than or equal to the file position indicator.

NOTE: For OPEN, this means that you normally get the first record in the file for sequential or relative and normally get an at end condition for indexed.

c. If the file position indicator was established by a previous READ statement and the file is sequential or relative, or an indexed file whose current key of reference does not allow duplicates, the first existing record in the

file whose record number (or relative record number) or key value is greater than the file position indicator if NEXT is specified or implied or is less than the file position indicator if PREVIOUS is specified is selected.

d. For indexed files, if the file position indicator was established by a previous READ statement, and the current key of reference does allow duplicates, the record that is selected is one of the following:

1. If NEXT is specified or implied, the first record in the file whose key value is either equal to the file position indicator and whose logical position within the set of duplicates is immediately after the record that was made available by that previous READ statement, or whose key value is greater that the file position indicator.

2. If PREVIOUS is specified, the first record in the file whose key value is either equal to the file position indicator and whose logical position within the set of duplicates is immediately prior to the record that was made available by that previous READ statement, or whose key value is less than the file position indicator.

If a record is found which satisfies the above rules, it is made available in the record area associated with *file-name*, unless the RELATIVE KEY phrase is specified for *file-name* and the number of significant digits in the relative record number of the selected record is larger than the size of the relative key data item, in which case, the file position indicator is set to indicate this condition and execution proceeds as specified in General Rule 10.

If no record is found which satisfies the above rules, the file position indicator is set to indicate that no next logical record exists and execution proceeds as specified in General Rule 9.

If a record is made available, the file position indicator is set to the record number of the record made available.

(5) Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged; a record is available to the object program prior to the execution of *imperative-statement-2*, if specified, or prior to the execution of any statement following the READ statement, if *imperative-statement-2* is not specified.

(6) When the logical records of a file are described with more than one record description, these records automatically share the same record area in storage; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record ate undefined at the completion of the execution of the READ statement.

(7) The INTO phrase may be specified in a READ statement:

a. If only one record description is subordinate to the file description entry, or

b. If all record-names associated with file-name and the data item referenced by *identifier-1* describe a group item or an elementary alphanumeric item.

(8) The result of the execution of a READ statement with the INTO phrase is equivalent to the application of the following rules in the order specified:

a. The execution of the same READ statement without the INTO phrase.

b. The current record is moved from the record area to the area specified by *identifier-1* according to the rules for the MOVE statement without the CORRESPONDING phrase. The size of the current record is specified in the RECORD clause. If the file description entry contains a RECORD IS VARYING clause, the implied move is a group move. The implied MOVE statement does not occur if the execution of the READ statement was unsuccessful. Any subscripting associated with *identifier-1* is evaluated after the record has been read and immediately before it is moved to the data item. The record is available in both the record area and the data item referenced by *identifier-1*.

(9) For **ANSI 85**, if at the time of execution of a format 2 READ statement, the file position indicator indicates that an optional input file is not present, the invalid key condition exists and execution of the READ statement is unsuccessful.

(10) For a Format 1 READ statement, if the file position indicator indicates that no next logical record exists, or that the number of significant digits in the relative record number is larger that the size of the relative key data item, the following occurs in the order specified:

a. A value, derived from the setting of the file position indicator, is placed into the I-O status associated with *file-name* to indicate the at end condition.

b. If the AT END phrase is specified in the statement causing the condition, control is transferred to *imperative-statement-1* in the AT END phrase. Any USE AFTER STANDARD EXCEPTION procedure associated with *file-name* is not executed.

c. If the AT END phrase is not specified, a USE AFTER STANDARD EXCEPTION procedure must be associated with this *file-name*, and that procedure is executed. Return from that procedure is to the next executable statement following the end of the READ statement.

When the at end condition occurs, execution of the READ statement is unsuccessful.

(11) If neither an at end nor an invalid key condition occurs during the execution of a READ statement, the AT END phrase or INVALID KEY phrase is ignored, if specified, and the following actions occur:

a. The file position indicator is set and the I-O status associated with *file-name* is updated.

b. If an exception condition which is not an at end or invalid key condition exists, control is transferred according to rules of the USE statement following the execution of any USE AFTER STANDARD EXCEPTION procedure applicable to *file-name*.

c. If no exception condition exists, the record is made available in the record area and any implicit move resulting from the presence of an INTO phrase is executed. Control is transferred to the end of the READ statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the READ statement.

(12) Following the unsuccessful execution of a READ statement, the content of the associated record area is undefined, and the file position indicator is set to indicate that no valid next record has been established. If the READ statement is unsuccessful due to the end-of-file condition or because the record which would have been returned is locked, the file position indicator remains unchanged. In these cases, the program can loop for the condition to be released.

(13) If the number of character positions in the record that is read is less than the minimum size specified by the record description entries for *file-name*, the portion of the record area which is to the right of the last valid character read is undefined. If the number of character positions in the record that is read is greater than the maximum specified by the record description entries for *file-name*, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and an I-O status is set, indicating that a record length conflict has occurred.

For relative files:

(14) For a relative file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file.

(15) For a relative file, if the RELATIVE KEY phrase is specified for *file-name*, the execution of a Format 1 READ statement moves the relative record number of the record made available to the relative key data item according to the rules for the MOVE statement.

(16) For a relative file, execution of a Format 2 READ statement sets the file position indicator to the value contained in the data item referenced by the RELATIVE KEY phrase for the file, and the record whose relative record number equals the file position indicator is made available in the record area associated with *file-name*. If the file does not contain such a record, the invalid key condition exists and execution of the READ statement is unsuccessful.

For indexed files:

(17) For an indexed file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file.

(18) For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key which is the key of reference are made available in the same order in which they are released by execution of WRITE statements, or by execution of REWRITE statements which create such duplicate values.

(19) For an indexed file, if the KEY phrase is specified in a Format 2 READ statement, *key-name* is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements for the file until a different key of reference is established for the file.

(20) For an indexed file, if the KEY phrase is not specified in a Format 2 READ statement, the primary record key is established as the key of reference for this statement. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent execution of Format 1 READ statements for the file until a different key of reference is established for the file.

(21) For an indexed file, execution of a Format 2 READ statement sets the file position indicator to the value in the key of reference. If the RECORD KEY or ALTERNATE RECORD KEY clause of the file control entry includes the OCCURS phrase, the file position indicator is set to the value in the first occurrence. Similarly, if the clause contains the ALSO phrase, the value of the root key (*id-2* in the format for the RECORD KEY and ALTERNATE RECORD clauses) is used to set the file position indicator. This value is compared with the value contained in the corresponding data item of the stored records in the file until the first record having an equal value is found. In the case of an alternate key with duplicate values, the first record found is the first record of a sequence of duplicates which was released to the file system. The record so found is made available in the record area associated with *file-name*. If no record can be so identified, the invalid key condition exists and execution of the READ statement is unsuccessful.

For relative and indexed files:

(22) If the LOCK phrase is specified, the system attempts to lock the record for the exclusive use of the currently executing program. If the lock operation and the read operation are successful, the record may not be read, deleted, or rewritten by another user, with one exception: a READ statement executed on a file open in the INPUT mode or a READ with the IGNORE LOCK clause will ignore the lock and the record can be read.

(23) If IGNORE LOCK is specified, READ will successfully read the data from a locked record. (This behaves similarly to reading locked records in a file open for INPUT.)

(24) If the record cannot be locked, either because it is already locked by another user or because of system limitations on the number of locks, the I-O status is set to indicate the lock violation and the READ statement is unsuccessful.

(25) If the conditions in General Rule 22 cause the READ statement to be unsuccessful, the current record position is not modified, rather than being set as specified in General Rule 12.

(26) A record lock can be removed by the successful execution of an UNLOCK or CLOSE statement for the file.

(27) The END-READ phrase delimits the scope of the READ statement.

For sequential files:

(28) The TIME-OUT phrase enables a local timeout for the particular READ statement. The file specified must be capable of timing out, generally a serial line. The timeout specifies the amount of time, in seconds, that the runtime will wait for individual keystrokes (characters). If the time expires, the READ terminates with an I-O status of 9T and an exception status of 76. Valid timeout values are:

<= 0 or >= 65535	No timeout (Wait forever)
65534	Timeout immediately
> 6300	Set to 6300 seconds
1-6300	Set to n seconds

(29) If the timeout value specified by *identifier-2* or *literal-1* is not an integer, its value is rounded to the nearest tenth of a second.

(30) When using timeouts, **ICOBOL** handles them in the following order for READ statements:

a) If a local timeout was specified by the TIME-OUT clause then it is used, otherwise,

b) If a timeout was set on the OPEN with the extended open option for timeout, then it is used; otherwise,

c) The default timeout for this particular device class is used.

NOTE: Extended open options are discussed in the Developer's Guide section beginning on page 756.

(31) When performing data-sensitive reads on files whose file control entry has one of the clauses ASSIGN TO KEYBOARD, ORGANIZATION IS LINE SEQUENTIAL, or RECORD DELIMITER IS DATA-SENSITIVE then the characters null <000>, carriage-return <015>, newline <012>, form-feed <014>, or the carriage-return newline pair <015><012) are used to terminate the read. If the ASSIGN TO KEYBOARD phrase is used, the terminator (one or two bytes) is included in the record if there is sufficient room and it will be included in the length of the record. If the RECORD DELIMITER IS DATA-SENSITIVE phrase the delimiter is NOT placed into the record area. The DELIMITER INTO phrase can be used to capture the one or two byte delimiter. The size of the delimiter is NOT included in the length of the record. In this case an "empty" data-sensitive record will have a length of zero (0) which will return a file status of "04" since the smallest VARYING RECORD size is 1. We suggest using the DEPENDING on phrase to return the actual record length and then use the length to determine whether the "04" was caused by an empty record or a record that is too long.

E.41. READ (*VXCOBOL*)

E.41.1 Function

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a mass storage file.

E.41.2 General Format

Format 1: Sequential Access Mode

For sequential files:

```
READ file-name [ NEXT ] RECORD [ INTO identifier-1 ]
    [ AT END imperative-statement-1 ]
    [ NOT AT END imperative-statement-2 ]
    [ END-READ ]
```

For relative and indexed files:

```
READ file-name [ MANDATORY ] [ NEXT | BACKWARD ]        [ LOCK | UNLOCK ] RECORD        [ INTO identifier-1 ]
    [ AT END imperative-statement-1 ]
    [ NOT AT END imperative-statement-2 ]
    [ END-READ ]
```

For INFOS files:

```
READ file-name [ MANDATORY ] [ { FIX | RETAIN } POSITION ] [ NEXT | FORWARD | BACKWARD | UP FORWARD | UP BACKWARD | DOWN FORWARD ]
    [ SUPPRESS [ PARTIAL RECORD ] [ DATA RECORD ] | RECORD ]
    [ LOCK | UNLOCK ]        [ INTO identifier-1 ]
    [ AT END imperative-statement-1 ]
    [ NOT AT END imperative-statement-2 ]
    [ END-READ ]
```

Format 2:

For relative files:

```
READ file-name RECORD [ LOCK | UNLOCK ]        [ WAIT ] [ INTO identifier-1 ]
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-READ ]
```

For indexed files:

READ *file-name* RECORD [LOCK / UNLOCK] [INTO *identifier-1*] [KEY IS *data-name*]
[INVALID KEY *imperative-statement-1*]
[NOT INVALID KEY *imperative-statement-2*]
[END-READ]

For INFOS files:

READ *file-name* [MANDATORY] [{ FIX / RETAIN } POSITION] [UP / DOWN / STATIC]
[RECORD / SUPPRESS [PARTIAL RECORD] [DATA RECORD]]
[LOCK / UNLOCK] [INTO *identifier-1*] [{ KEY IS / KEYS ARE } { data-name [APPROXIMATE / GENERIC] }...]
[INVALID KEY *imperative-statement-1*]
[NOT INVALID KEY *imperative-statement-2*]
[END-READ]

E.41.3 Syntax Rules

(1) The storage area associated with *identifier-1* and the record area associated with *file-name* must not be the same storage area.

(2) Format 1 must be used for all files in sequential access mode.

(3) In Format 1, the NEXT or BACKWARD phrase must be specified for files in dynamic access mode when records are to be retrieved sequentially.

(4) Format 2 is used for indexed, relative, and INFOS files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

(5) The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE AFTER STANDARD EXCEPTION procedure is specified for *file-name*.

For indexed files:

(6) *Data-name* must be the name of a data item specified as a record key associated with *file-name*.

(7) *Data-name* may be qualified.

E.41.4 General Rules

(1) The file referenced by *file-name* must be open in the input or I-O mode at the time this statement is executed.

(2) In Format 1, if neither the NEXT phrase nor the BACKWARD phrase is specified, then NEXT is implied for files in sequential access mode.

(3) The execution of the READ statement causes the value of the I-O status and INFOS status associated with *file-name* to be updated.

(4) The setting of the file position indicator at the start of the execution of a Format 1 READ statement is used in determining the record to be made available according to the following rules. Comparisons for records in sequential files relate to the record number. Comparisons for records in relative files relate to the relative key number. Comparisons for records in indexed or INFOS files relate to the value of the current key of reference. For indexed or INFOS files, the comparisons are made according to the collating sequence of the file.

a. If the file position indicator indicates that no valid next record as been established, execution of the READ statement is unsuccessful.

b. If the file position indicator was established by a previous OPEN or START statement, the first existing record that is selected is either:

1. If NEXT is specified or implied, the first existing record in the file whose record number or key value is greater than or equal to the file position indicator, or

2. If BACKWARD is specified, the first existing record in the file whose record number or key value is less than or equal to the file position indicator.

NOTE: For OPEN, this means that you normally get the first record in the file for sequential or relative and normally get an at end condition for indexed.

c. If the file position indicator was established by a previous READ statement and the file is sequential or relative, an indexed, of INFOS file whose current key of reference does not allow duplicates, the first existing record in the file whose record number (or relative record number) or key value is greater than the file position indicator if NEXT is specified or implied or is less than the file position indicator if BACKWARD is specified is selected.

d. For indexed files or INFOS, if the file position indicator was established by a previous READ statement, and the current key of reference does allow duplicates, the record that is selected is one of the following:

1. If NEXT is specified or implied, the first record in the file whose key value is either equal to the file position indicator and whose logical position within the set of duplicates is immediately after the record that was made available by that previous READ statement, or whose key value is greater that the file position indicator.

2. If BACKWARD is specified, the first record in the file whose key value is either equal to the file position indicator and whose logical position within the set of duplicates is immediately prior to the record that was made available by that previous READ statement, or whose key value is less than the file position indicator.

If a record is found which satisfies the above rules, it is made available in the record area associated with *file-name*, unless the RELATIVE KEY phrase is specified for *file-name* and the number of significant digits in the relative record number of the selected record is larger than the size of the relative key data item, in which case, the file position indicator is set to indicate this condition and execution proceeds as specified in General Rule 9.

If no record is found which satisfies the above rules, the file position indicator is set to indicate that no next logical record exists and execution proceeds as specified in General Rule 9.

If a record is made available, the file position indicator is set to the record number of the record made available.

(5) Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged; a record is available to the object program prior to the execution of *imperative-statement-2*, if specified, or prior to the execution of any statement following the READ statement, if *imperative-statement-2* is not specified.

(6) When the logical records of a file are described with more than one record description, these records automatically share the same record area in storage; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

(7) The INTO phrase may be specified in a READ statement:

a. If only one record description is subordinate to the file description entry, or

b. If all record-names associated with *file-name* and the data item that is referenced by *identifier-1* describe a group item or an elementary alphanumeric item.

(8) The result of the execution of a READ statement with the INTO phrase is equivalent to the application of the following rules in the order specified:

a. The execution of the same READ statement without the INTO phrase.

b. The current record is moved from the record area to the area specified by *identifier-1* according to the rules for the MOVE statement without the CORRESPONDING phrase. The size of the current record is specified in the RECORD LENGTH clause. The implied MOVE statement does not occur if the execution of the READ statement was unsuccessful. Any subscripting associated with *identifier-1* is evaluated after the record has been read and immediately before it is moved to the data item. The record is available in both the record area and the data item referenced by *identifier-1*.

(9) For a Format 1 READ statement, if the file position indicator indicates that no next logical record exists, or that the number of significant digits in the relative record number is larger that the size of the relative key data item, the following occurs in the order specified:

a. A value, derived from the setting of the file position indicator, is placed into the I-O status and INFOS status associated with *file-name* to indicate the at end condition.

b. If the AT END phrase is specified in the statement causing the condition, control is transferred to *imperative-statement-1* in the AT END phrase. Any USE AFTER STANDARD EXCEPTION procedure associated with *file-name* is not executed.

c. If the AT END phrase is not specified, a USE AFTER STANDARD EXCEPTION procedure must be associated with this *file-name*, and that procedure is executed. Return from that procedure is to the next executable statement following the end of the READ statement.

When the at end condition occurs, execution of the READ statement is unsuccessful.

(10) If neither an at end nor an invalid key condition occurs during the execution of a READ statement, the AT END phrase or INVALID KEY phrase is ignored, if specified, and the following things happen:

a. The file position indicator is set and the I-O status and INFOS status associated with *file-name* is updated.

b. If an exception condition which is not an at end or invalid key condition exists, control is transferred according to rules of the USE statement following the execution of any USE AFTER STANDARD EXCEPTION procedure applicable to *file-name*. If there is no applicable USE statement and the I-O status is 96, any implicit move resulting from the presence of the INTO phrase is executed.

c. If no exception condition exists, the record is made available in the record area and any implicit move resulting from the presence of an INTO phrase is executed. Control is transferred to the end of the READ statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the READ statement.

(11) Following the unsuccessful execution of a READ statement, the content of the associated record area is undefined and the file position indicator is set to indicate that no valid next record has been established. If the

READ statement is unsuccessful due to the end-of-file condition or because the record which would have been returned is locked, the file position indicator remains unchanged.

(12) If the number of character positions in the record that is read is less than the minimum size specified by the record description entries for *file-name*, the portion of the record area which is to the right of the last valid character read is undefined. If the number of character positions in the record that is read is greater than the maximum specified by the record description entries for *file-name*, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and an I-O status is set, to indicate that a record length conflict has occurred.

(13) The END-READ phrase delimits the scope of the READ statement.

For relative files:

(14) For a relative file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file.

(15) For a relative file, if the RELATIVE KEY phrase is specified for *file-name*, the execution of a Format 1 READ statement moves the relative record number of the record made available to the relative key data item according to the rules for the MOVE statement.

(16) For a relative file, execution of a Format 2 READ statement sets the file position indicator to the value contained in the data item referenced by the RELATIVE KEY phrase for the file, and the record whose relative record number equals the file position indicator is made available in the record area associated with *file-name*. If the file does not contain such a record, the invalid key condition exists and execution of the READ statement is unsuccessful.

For indexed and INFOS files:

(17) For an indexed file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file.

(18) For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key which is the key of reference are made available in the same order in which they are released by execution of WRITE statements, or by execution of REWRITE statements which create such duplicate values.

(19) For an indexed file, if the KEY phrase is specified in a Format 2 READ statement, *data-name* is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements for the file until a different key of reference is established for the file.

(20) For an indexed file, if the KEY phrase is not specified in a Format 2 READ statement, the primary record key is established as the key of reference for this statement. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent execution of Format 1 READ statements for the file until a different key of reference is established for the file.

(21) For an indexed file, execution of a Format 2 READ statement sets the file position indicator to the value in the key of reference. This value is compared with the value contained in the corresponding data item of the stored records in the file until the first record having an equal value is found. In the case of an alternate key with duplicate values, the first record found is the first record of a sequence of duplicates which was released to the file system. The record so found is made available in the record area associated with *file-name*. If no record can be so identified, the invalid key condition exists and execution of the READ statement is unsuccessful.

For relative, indexed, and INFOS files:

(22) If the LOCK phrase is specified, the system attempts to lock the record for the exclusive use of the currently executing program. If the lock operation and the read operation are successful, the record may not be read,

deleted, or rewritten by another user, with two exceptions: a READ statement executed on a file open in the INPUT mode or a read with the MANDATORY clause will ignore the lock, and the record can be read.

(23) If the record cannot be locked, either because it is already locked by another user or because of system limitations on the number of locks, the I-O status is set to indicate the lock violation and the READ statement is unsuccessful.

(24) If the conditions in General Rule 21 cause the READ statement to be unsuccessful, the current record position is not modified, rather than being set as specified in General Rule 11.

(25) A record lock can be removed by the successful execution of an UNLOCK or CLOSE statement for the file, or by execution of an i/o statement with the UNLOCK clause.

(26) If the UNLOCK phrase is specified, the system attempts to unlock the record after completion of the READ. The record then becomes accessible to any user if it had in fact been locked.

For INFOS files:

(27) If a file is opened for input and the MANDATORY keyword is used, the program will read a record even if the record is locked.

(28) If the position phrase is omitted, FIX POSITION is the default.

(29) If the relative option and the KEY series phrase are omitted in random or dynamic access mode, the default is the first key in the SELECT clause; in sequential access mode the default is READ NEXT.

(30) For an INFOS file being sequentially accessed, records having the same duplicate value in a record key are made available in the same order in which they are released by execution of WRITE statements or by execution of REWRITE statements that create duplicate values.

(31) For an INFOS file being randomly accessed, records having the same duplicate value in a key allowing duplicates are made available as follows:

a. If the OCCURRENCE clause was specified for the key and contains a non-zero value and a key with the specified value and specified occurrence number exists, then that record will be returned.

b. If no OCCURRENCE clause was specified for the key or if the value of the occurrence data-item is zero, then the record associated with the first key of the specified value, if any, is returned.

(32) If a FEEDBACK data-item was specified for the file, its value is updated by execution of a successful READ.

(33) If a RECORD LENGTH data-item was specified for the file, it will be updated with the length of the record read.

(34) KEY LENGTH is used.

(35) The location of the entry defined is determined according to that specified in the position phrase, the relative option phrase, and/or the KEY series phrase. The specification can be implicit if the program uses the defaults or explicit if the KEY or path is specified fully.

(36) FIX POSITION causes the record pointer to move from the current position to the position specified in this statement. RETAIN position causes the record position to remain at the position it was on before the execution of this statement. RETAIN is the default.

(37) The relative motion option without the KEY series phrase allows access to the index file relative to that file's current record position.

(38) Using the KEY series phrase without the relative motion option cause the key path specified to begin with the top index in the hierarchy and follow a downward motion.

(39) If the KEY series phrase is specified, each key, *data-name*, must be included in the RECORD KEY clause of the SELECT statement for *file-name*. If the relative motion option and KEY series phrase at both specified only UP, DOWN, and STATIC are allowed. The relative motion option is processed first and the key path is used. If both are omitted, STATIC is the default.

(40) If SUPPRESS DATA RECORD is specified, all locks on the data record are ignored, and the data record associated with the referenced index entry is not read into the file's record area.

(41) If SUPPRESS PARTIAL RECORD is specified, all locks on the partial record are ignored and the partial data record associated with the index entry is not retrieved.

For sequential files:

(42) When using timeouts, **ICOBOL** handles them in the following order for READ statements:

a. If a timeout was set on the OPEN with the extended open option for timeout, then it is used,

b. The default timeout for the particular device class is used.

If a timeout occurs, the I-O status is set to 9T with an exception status of 76.

(43) When performing data-sensitive reads on files whose file control entry has the clause ASSIGN TO KEYBOARD or whose file description entry specifies RECORDING MODE IS DATA-SENSITIVE the characters null <000>, carriage-return <015>, newline <012>, and form-feed <014> are used to terminate the read. The terminator is placed into the record if there is sufficient room. If the record area is too small, the maximum amount of data is stored in the record and file status 99 (exception status 40) is returned.

E.42. RELEASE

E.42.1 Function

The RELEASE statement transfers records to the initial phrase of a sort operation.

E.42.2 General Format

RELEASE *record-name* [FROM *identifier*]

E.42.3 Syntax Rules

(1) *Record-name* must be the name of a logical record in a sort-merge file description, and it may be qualified.

(2) A RELEASE statement may be used only within the range of an input procedure associated with a SORT statement for the file-name whose sort-merge file description entry contains *record-name*.

(3) *Record-name* and *identifier* must not refer to the same storage.

(4) If *identifier* is a function identifier, it shall reference an alphanumeric function.

E.42.4 General Rules

(1) The execution of a Release statement causes the record named by *record-name* to be released to the initial phrase of a sort operation.

(2) The logical record released by the execution of the RELEASE statement is no longer available in the record area unless the sort-merge file-name associated with *record-name* is specified in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as the file associated with *record-name*.

(3) The result of the execution of a RELEASE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:

a. The statement:

MOVE identifier TO *record-name*

according to the rules specified for the MOVE statement.

b. The same RELEASE statement without the FROM phrase.

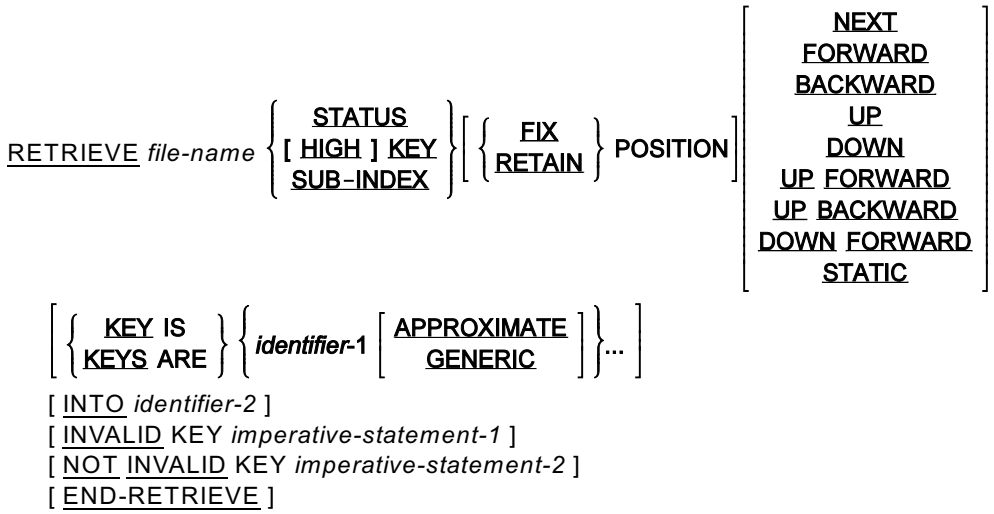
(4) After the execution of the RELEASE statement is complete, the information in the area referenced by *identifier* is available, even though the information in the area referenced by *record-name* is not available except as specified by the SAME RECORD AREA clause.

E.43. RETRIEVE (***VXCOBOL***)

E.43.1 Function

The RETRIEVE statement obtains information about an INFOS file.

E.43.2 General Format



E.43.3 Syntax Rules

(1) *File-name* is a filename that specifies an INFOS file opened for OUTPUT or I/O and selected for ALLOW SUB-INDEX.

(2) *Identifier-1* is an alphanumeric data item that specifies a record key associated with *file-name*.

(3) *Identifier-2* is any data item specifying a destination that receives record, key, or index status information.

E.43.4 General Rules

(1) If the relative option and the KEY series phrase are omitted, the default is STATIC.

(2) The occurrence number is not used and is not updated.

(3) FEEDBACK is not used and is not updated.

(4) KEY LENGTH is used.

(5) The location from which to retrieve information is determined according to what is specified in the relative option phrase and/or the KEY series phrase.

(6) FIX POSITION causes the record pointer to move from the current position to the position specified in this statement. RETAIN position causes the record position to remain at the position it was on before the execution of this statement. If the position phrase is omitted, RETRIEVE KEY and RETRIEVE HIGH KEY default to FIX POSITION and RETRIEVE STATUS and RETRIEVE SUBINDEX default to RETAIN.

(7) The relative motion option without the KEY series phrase allows access to the index file relative to that file's current record position.

(8) Using the KEY series phrase without the relative motion option cause the key path specified to begin with the top index in the hierarchy and follow a downward motion.

(9) If the KEY series phrase is specified, each key, *identifier-1*, must be declared in the SELECT statement for file-name. If the relative motion option and KEY series phrase at both specified only UP, DOWN, and STATIC are allowed. The relative motion option is processed first and the key path is used. If both are omitted, STATIC is the default.

(10) Transfer of control following the successful or unsuccessful execution of the RETRIEVE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the RETRIEVE statement.

(11) INVALID KEY clauses on I/O statements are ONLY invoked when an Invalid Key error, as determined by a File Status of 2*x* where *x* can be any character 0 - 9 or A - Z, is generated. All other error conditions will cause the associated USE procedure, if present, as defined in the DECLARATIVES section to be executed. (See The Invalid Key Condition, page 271, for more a more comprehensive discussion.).

(12) If STATUS is specified, *identifier-2* is interpreted as a 4-character data item. **ICOBOL** will store either a "1" or "0" in each byte as follows:

Character 1:	"1" if partial record for target key is logically deleted.
Character 2:	"1" if target key is a duplicate.
Character 3:	always "0"
Character 4:	"1" if data record for target key is logically deleted.

(13) If KEY is specified, **ICOBOL** moves the value of the target key to identifier-2. If HIGH KEY is specified, **ICOBOL** moves the value of the highest key present in the subindex associated with the target key to identifier-2. In either case, the following also occur:

a. The key length of the retrieved key is returned to the KEY LENGTH data-item associated with the last key in the key series phrase or the first key in the SELECT statement if there is no key series phrase. If no KEY LENGTH data item exits, the length is not accessible.

b. If the key is a duplicate, the occurrence number of the retrieved key is returned to the OCCURRENCE data-item associated with the last key in the key series phrase or the first key in the SELECT statement if there is no key series phrase. If the retrieved key is not a duplicate, a zero is returned to the OCCURRENCE data-item. If no OCCURRENCE data-item exists, the occurrence number is not accessible.

(14) If SUB-INDEX is specified, **ICOBOL** returns a 16-byte data item to identifier-2. This 16-byte item is the 16-bit AOS INFOS subindex definition packet. It has the following format:

```
01 PACKET.
   03 FILLER            PIC XX.
   03 NODE-SIZE         PIC 9(4) COMP.
   03 FILLER            PIC X.
   03 MAX-KEYLEN        PIC 9(2) COMP.
   03 FILLER            PIC X.
   03 PARTIAL-REC-LEN   PIC 9(2) COMP.
   03 FILLER            PIC XX.
   03 FLAGS             PIC 9(4) COMP.
   03 FILLER            PIC X(4).
```

FLAGS values are: 2048 allow duplicates, 16384 Disallow subindex.

Under U/FOS, the partial record length will be either zero (no partial records) or 255 (partial records allowed). Under INFOS II, the actual length was returned.

E.44. RETURN

E.44.1 Function

The RETURN statement obtains either sorted records from the final phrase of a sort operation or merged records during a merge operation.

E.44.2 General Format

```
RETURN file-name RECORD [ INTO identifier ]
    AT END imperative-statement-1
    [ NOT AT END imperative-statement-2 ]
    [ END-RETURN ]
```

E.44.3 Syntax Rules

(1) The storage area associated with *identifier* and the record area associated with *file-name* must not be the same storage area.

(2) *File-name* must be described by a sort-merge file description entry in the Data Division.

(3) A RETURN statement may only be used within the range of an output procedure associated with a SORT or MERGE statement for *file-name*.

E.44.4 General Rules

(1) When the logical records in a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the RETURN statement.

(2) The execution of the RETURN statement causes the next existing record in the file referenced by *file-name*, as determined by the keys listed in the SORT or MERGE statement, to be made available in the record area associated with *file-name*. If no next logical record exists in the file referenced by *file-name*, the at end conditions exists and control is transferred to *imperative-statement-1* of the AT END phrase. Execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred according to the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the RETURN statement and the NOT AT END phrase is ignored, if specified. When the at end condition occurs, execution of the RETURN statement is unsuccessful and the contents of the record area associated with *file-name* are undefined. After the execution of *imperative-statement-1* in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.

(3) If an at end condition does not occur during the execution of a RETURN statement, then after the record is made available and after executing any implicit move resulting from the presence of an INTO phrase, control is transferred to *imperative-statement-2*, if specified; otherwise, control is transferred to the end of the RETURN statement.

(4) The END-RETURN phrase delimits the scope of the RETURN statement. (See page 253, Scope of Statements.)

(5) The INTO phrase may be specified in a RETURN statement:

a. If only one record description is subordinate to the sort-merge file description entry, or

b. If all record-names associated with *file-name* and the data items referenced by *identifier* describe a group item or an elementary alphanumeric item.

(6) The result of the execution of a RETURN statement with the INTO phrase is equivalent to the application of the following rules in the order specified:

a. The execution of the same RETURN statement without the INTO phrase.

b. The current record is moved from the record area to the area specified by *identifier* according to the rules for the MOVE statement without the CORRESPONDING phrase. The size of the current record is determined by rules specified for the RECORD clause. If the file description entry contains a RECORD IS VARYING clause, the implied move is a group move. The implied MOVE statement does not occur if the execution of the RETURN statement was unsuccessful. Any subscript or reference modification associated with *identifier* is evaluated after the record has been read and immediately before it is moved to the data item. The record is available in both the record area and the data item referenced by *identifier*.

E.45. REWRITE

E.45.1 Function

The REWRITE statement logically replaces a record existing in a mass storage file. IMMEDIATE is an extension to ANSI COBOL.

E.45.2 General Format

Sequential Files:

REWRITE *record-name* [IMMEDIATE] [FROM *identifier*]
[END-REWRITE]

Relative and Indexed Files: (***ANSI 74*** and ***ANSI 85***)

REWRITE *record-name* [IMMEDIATE] [FROM *identifier*]
[INVALID KEY *imperative-statement-1*]
[NOT INVALID KEY *imperative-statement-2*]
[END-REWRITE]

Relative and Indexed: (***VXCOBOL***)

REWRITE *record-name* [IMMEDIATE] [FROM *identifier*] [KEY IS *identifier-2*]
[INVALID KEY *imperative-statement-1*]
[NOT INVALID KEY *imperative-statement-2*]
[END-REWRITE]

INFOS: (***VXCOBOL***)

REWRITE [INVERTED] *record-name* [IMMEDIATE]

```
                                  [ NEXT          ]
                                  [ FORWARD       ]
                                  [ BACKWARD      ]
                                  [ UP            ]
[ { FIX    } POSITION ]           [ DOWN          ]
[ { RETAIN }          ]           [ UP FORWARD    ]
                                  [ UP BACKWARD   ]
                                  [ DOWN FORWARD  ]
                                  [ STATIC        ]
```

[SUPPRESS [PARTIAL RECORD] [DATA RECORD]]

```
[ LOCK   ]          { identifier }
[ UNLOCK ]          { literal    }                    ]
```

```
[ { KEY IS  } { identifier-2 [ APPROXIMATE ] } ... ]
[ { KEYS ARE } {             [ GENERIC     ] }     ]
```

[INVALID KEY *imperative-statement-1*]
[NOT INVALID KEY *imperative-statement-2*]
[END-REWRITE]

E.45.3 Syntax Rules

(1) *Record-name* and *identifier* must not refer to the same storage area.

(2) *Record-name* is the name of a logical record in the File Section of the Data Division and may be qualified.

(3) The INVALID KEY and NOT INVALID KEY phrases must not be specified for a REWRITE statement which references a relative, indexed, or INFOS file in sequential access mode.

(4) The INVALID KEY phrase must be specified in the REWRITE statement for relative, indexed, or INFOS files in the random or dynamic access mode, and for which an appropriate USE AFTER STANDARD EXCEPTION procedure is not specified.

(5) For ***VXCOBOL***, for an indexed file, *identifier-2* must reference the RECORD KEY data-item for the file.

E.45.4 General Rules

(1) The file referenced by the *file-name* associated with *record-name* must be a mass storage file and must be open in the I-O mode at the time of execution of this statement.

(2) For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The file system logically replaces the record that was accessed by the READ statement.

(3) The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the *file-name* associated with *record-name* is specified in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as the file associated with *record-name*.

(4) The result of the execution of a REWRITE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:

a. The statement:

MOVE *identifier* TO *record-name*

according to the rules specified for the MOVE statement.

b. The same REWRITE statement without the FROM phrase.

(5) After the execution of the REWRITE statement is complete, the information in the area referenced by *identifier* is available, even though the information in the area referenced by *record-name* is not available except as specified by the SAME RECORD AREA clause.

(6) The file position indicator is not affected by the execution of a REWRITE statement.

(7) The execution of the REWRITE statement causes the value of the I-O status (and for ***VXCOBOL***, the INFOS status) of the file-name associated with *record-name* to be updated.

(8) The execution of the REWRITE statement releases a logical record to the operating system.

(9) For **ANSI 74 and ANSI 85**, when using indexed or relative files, the number of character positions in the record referenced by *record-name* must not be larger than the largest or smaller than the smallest number of character positions allowed by the RECORD IS VARYING clause associated with the file-name associated with *record-name*. For ***VXCOBOL***, when using indexed, relative, or INFOS files with RECORDING MODE IS

VARIABLE, the number of character positions in the record referenced by *record-name* must not be larger than or smaller than the maximum and minimum record lengths established for the file.

In either of these cases the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the contents of the record area are unaffected and the I-O status of the file associated with *record-name* is set to a value indicating the cause of the condition.

(10) Transfer of control following the successful or unsuccessful execution of the REWRITE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the REWRITE statement.

(11) The END-REWRITE phrase delimits the scope of the REWRITE statement.

(12) For sequential, relative, and indexed files, the IMMEDIATE option causes the REWRITE to immediately flush the new information to disk. Normally this information could be held in internal buffers before being flushed to disk. This option increases file security at the expense of performance. For INFOS files the IMMEDIATE option is ignored.

For relative files:

(13) For a relative file, for a file accessed in either random or dynamic access mode, the file system logically replaces the record specified by the content of the relative key data of the file-name associated with *record-name*. If the file does not contain the record specified by the key, the invalid key condition exists. When the invalid key condition is recognized, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the contents of the record area are unaffected and the I-O status of the file-name associated with *record-name* is set to a value indicating the cause of the condition.

For sequential files:

(14) If the number of character positions specified in the record referenced by *record-name* is not equal to the number of character positions in the record being replaced, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the content of the record area is unaffected and the I-O status of the file associated with *record-name* is set to a value indicating the cause of the condition.

For indexed files:

(15) For a file in the sequential access mode, the record to be replaced is specified by the value of the primary record key. When the REWRITE statement is executed, the value of the primary record key of the record to be replaced must be equal to the value of the primary record key of the last record read from this file.

(16) For a file in the random or dynamic access mode, the record to be replaced is specified by the primary record key.

(17) Execution of the REWRITE statement for a record which has an alternate key occurs as follows:

a. When the value of a specific alternate record key is not changed, the order of retrieval when that key is the key of reference remains unchanged.

b. When the value of a specific alternate record key is changed, the subsequent order of retrieval of that record may be changed when that specific alternate record key is the key of reference. When duplicate key values are permitted, the record is logically positioned as though this new alternate key was written when the record itself was added to the file for ICISAM version 5 and 6) or last within the set of duplicate records containing the same alternate record key value as the one that was placed in the record. (ICISAM version 7)..

(18) The invalid key condition exists under the following circumstances:

a. When the file is open in the sequential access mode, and the value of the primary record key of the record to be replaced is not equal to the value of the primary record key of the last record read from the file, or

b. When the file is open in the dynamic or random access mode, and the value of the primary record key of the record to be replaced is not equal to the value of the primary record key of any record existing in the file, or

c. When the value of an alternate record key of the record to be replaced, for which duplicates are not allowed, equals the value of the corresponding data item of a record already existing in the file.

(19) When the invalid key condition is recognized, the execution of the REWRITE statement is unsuccessful and the I-O status (and for ***VXCOBOL***, the INFOS status) of the filename associated with *record-name* is set to a value indicating the cause of the condition.

For INFOS files:

(20) If the position phrase is omitted, RETAIN POSITION is the default.

(21) If the relative option and the KEY series phrase are omitted, the default is the first key in the SELECT clause.

(22) FEEDBACK is used if you specify INVERTED. REWRITE updates the FEEDBACK data item.

(23) KEY LENGTH is unused.

(24) If INVERTED is not specified, a record is written in a location that is determined according to what is specified in the relative option phrase and/or the KEY series phrase. The specification can be implicit if the program uses the defaults or explicit if the KEY or path is fully specified. If INVERTED is specified, the REWRITE statement does not write a data record but links an existing data record to an index entry with no data record. A FEEDBACK data-item must have been specified. It contains the record location REWRITE INVERTED links to the index entry.

(25) FIX POSITION causes the record pointer to move from the current position to the position specified in this statement. RETAIN position causes the record position to remain at the position it was on before the execution of this statement. RETAIN is the default.

(26) The relative motion option without the KEY series phrase allows access to the index file relative to that file's current record position.

(27) Using the KEY series phrase without the relative motion option cause the key path specified to begin with the top index in the hierarchy and follow a downward motion.

(28) If the KEY series phrase is specified, each key, *identifier-2*, must be declared in the SELECT statement for file-name. If the relative motion option and KEY series phrase at both specified only UP, DOWN, and STATIC are allowed. The relative motion option is processed first and the key path is used. If both are omitted, STATIC is the default.

(29) If DUPLICATE and OCCURRENCE IS was specified in this file's SELECT clause, and the occurrence number is not equal to zero, REWRITE uses the occurrence number to determine which record to rewrite. Zero indicates that the key is not a duplicate.

(30) If SUPPRESS DATA RECORD is specified, all locks on the data record are ignored and the data record associated with the index entry is not output.

(31) If SUPPRESS PARTIAL RECORD is specified, the partial data record associated with the index entry is not output.

E.46. ROLLBACK (***ISQL***)

E.46.1 Function

The ROLLBACK statement allows the program to rollback an SQL database connection or connections..

E.46.2 General Format

```
ROLLBACK [ ALL ]
    [ ON SQLERROR imperative-statement-1 ]
    [ NOT ON SQLERROR imperative-statement-2 ]
    [ END-ROLLBACK ]
```

E.46.3 Syntax Rules

E.47.4 General Rules

(1) The ALL phrase specifies that all connections in the run unit will be rolled back. (if there are any). If not specified, only the current connection is rolled back.

(2) Upon completion of the ROLLBACK statement, the following occurs in the order specified:

a. The value of the SQLSTATE data item is updated with the status of the operation.

b. If the value of the SQLSTATE class field is “00" or “01", the statement is successful. Control is transferred to the end of the ROLLBACK statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the ROLLBACK statement.

c. If the value of the SQLSTATE class field is not “00" or “01", the statement is unsuccessful. The statement container is deallocated and no statement container of the specified name will exist in the current program. Control is transferred to the end of the ROLLBACK statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the ROLLBACK statement.

(3) The END-ROLLBACK phrase delimits the scope of the ROLLBACK statement.

(4) More on SQLSTATE can be found on page 133.

E.47. SEARCH

E.47.1 Function

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the value of the associated index to indicate that table element.

E.47.2 General Format

Format 1:

```
SEARCH identifier-1 [ VARYING { identifier-2 } ]                    [ AT END imperative-statement-1 ]
                              { index-name-1 }
    { WHEN condition-1 { imperative-statement-2 } } ...
                       { NEXT SENTENCE          }
    [ END-SEARCH ]
```

Format 2:

```
SEARCH ALL identifier-1 [ AT END imperative-statement-1 ]
    WHEN { data-name-1      { IS EQUAL TO } { identifier-3           } }
         { condition-name-1 { IS =         } { literal-1             }
                           {               { arithmetic-expression-1 } }
      [ AND { data-name-2      { IS EQUAL TO } { identifier-4           } } ] ...
            { condition-name-2 { IS =         } { literal-2             }
                              {               { arithmetic-expression-2 } }
    { imperative-statement-2 }
    { NEXT SENTENCE          }
    [ END-SEARCH ]
```

E.47.3 Syntax Rules

(1) For ***VXCOBOL***, format 1 may include an optional keyword ALL immediately after the keyword SEARCH unless the -G h compiler switch is used. In that case the use of the keyword ALL must conform to format 2 and all the supporting rules which follow.

(2) In both formats 1 and 2, *identifier-1* must not be subscripted, but its description must contain an OCCURS clause including an INDEXED BY phrase. The description of *identifier-1* in Format 2 must also contain the KEY IS phrase in its OCCURS clause.

(3) *Identifier-2* must reference a data item described as USAGE IS INDEX or numeric elementary data item without any positions to the right of the decimal point. *Identifier-2* may not be subscripted by the first (or only) index-name specified in the INDEXED BY phrase in the OCCURS clause associated with *identifier-1*.

(4) In Format 1, *condition-1* may be any conditional expression.

(5) In Format 2, all referenced condition-names must be defined as having only a single value. The data-name associated with a condition-name must appear in the KEY IS phrase in the OCCURS clause referenced by *identifier-1*. Each *data-name-1*, *data-name-2* may be qualified. Each *data-name-1*, *data-name-2* must be subscripted by the first index-name associated with *identifier-1* along with other subscripts as required, and must be referenced in the KEY IS phrase in the OCCURS clause referenced by *identifier-1*. *Identifier-3*, *identifier-4*, or identifiers specified in *arithmetic-expression-1*, *arithmetic-expression-2* must not be referenced in the KEY IS

phrase in the OCCURS clause referenced by *identifier-1* or be subscripted by the first index-name associated with *identifier-1*.

In Format 2, when a data-name in the KEY IS phrase in the OCCURS clause referenced by *identifier-1* is referenced, or when a condition-name associated with a data-name in the KEY IS phrase in the OCCURS clause referenced by *identifier-1* is referenced, all preceding data-names in the KEY IS phrase in the OCCURS clause referenced by *identifier-1* or their associated condition-names must also be referenced.

(6) If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase must not be specified.

E.47.4 General Rules

(1) The scope of a SEARCH statement may be terminated by any of the following:

a. An END-SEARCH phrase at the same level of nesting.

b. A separator period.

c. An ELSE or END-IF phrase associated with a previous IF statement.

(2) If Format 1 of the SEARCH statement is used, a serial type of search operation takes place, starting with the current index setting.

a. If, at the start of execution of the SEARCH statement, the index-name associated with *identifier-1* contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for *identifier-1*, the search is terminated immediately. The number of occurrences of *identifier-1*, the last of which is the highest permissible, is discussed in the OCCURS clause. Then, if the AT END phrase is specified, *imperative-statement-1* is executed; if the AT END phrase is not specified, control passes to the end of the SEARCH statement.

b. If, at the start of execution of the SEARCH statement, the index-name associated with *identifier-1* contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for *identifier-1* (the number of occurrences of *identifier-1*, the last of which is the highest permissible, is discussed in the OCCURS clause), the SEARCH statement operates by evaluating the conditions in the order that they are written, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested. If none of the conditions is satisfied, the index-name for *identifier-1* is incremented to obtain reference to the next occurrence. The process is then repeated using the new index-name settings unless the new value of the index-name settings for *identifier-1* corresponds to a table element outside the permissible range of occurrence values, in which case the search terminates as indicated in 2a above. If one of the conditions is satisfied upon its evaluation, the search terminates immediately, and control passes to the imperative statement associated with that condition, if present, or if the NEXT SENTENCE phrase is associated with that condition, to the next executable sentence; the index-name remains set at the occurrence which caused the condition to be satisfied.

(3) In a Format 2 SEARCH statement, the results of the SEARCH ALL operation are predictable only when:

a. The data in the table is ordered in the same manner as described in the KEY IS phrase of the OCCURS clause referenced by *identifier-1*, and

b. The contents of the key(s) referenced in the WHEN phrase are sufficient to identify a unique table element.

(4) If Format 2 of the SEARCH statement is used, a nonserial type of search operation may take place; the initial setting of the index-name for *identifier-1* is ignored and its setting is varied during the search operation to conduct a binary search, with the restriction that at no time is it set to a value that exceeds the value which corresponds to the last element of the table, or that is less than the value that corresponds to the first element of the table. The length of the table is discussed in the OCCURS clause. If any of the conditions specified in the WHEN

phrase cannot be satisfied for any setting of the index within the permitted range, control is passed to *imperative-statement-1* of the AT END phrase, when specified, or to the end of the SEARCH statement when this phrase is not specified; in either case, the final setting of the index is not predictable. If all the conditions can be satisfied, the index indicates an occurrence that allows the conditions to be satisfied, and control passes to *imperative-statement-2*, if specified, or to the next executable sentence if the NEXT SENTENCE phrase is specified.

(5) After execution of *imperative-statement-1* or *imperative-statement-2*, that does not terminate with a GO TO statement, control passes to the end of the SEARCH statement.

(6) In Format 2, the index-name that is used for the search operation is the first (or only) index-name specified in the INDEXED BY phrase in the OCCURS clause associated with *identifier-1*. Any other index-names for *identifier-1* remain unchanged.

(7) In Format 1, if the VARYING phrase is not used, the index-name that is used for the search operation is the first (or only) index-name specified in the INDEXED BY phrase in the OCCURS clause associated with *identifier-1*. Any other index-names for *identifier-1* remain unchanged.

(8) In Format 1, if the VARYING *index-name-1* phrase is specified, and if *index-name-1* appears in the INDEXED BY phrase in the OCCURS clause referenced by *identifier-1*, that index-name is used for this search. If this is not the case or if the VARYING *identifier-2* phrase is specified, the first (or only) index-name given in the INDEXED BY phrase in the OCCURS clause referenced by *identifier-1* is used for the search. In addition, the following operations will occur:

a. If the VARYING *index-name-1* phrase is used, and if *index-name-1* appears in the INDEXED BY phrase in the OCCURS clause referenced by another table entry, the occurrence number represented by *index-name-1* is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with *identifier-1* is incremented.

b. If the VARYING *identifier-2* phrase is specified, and *identifier-2* is an index data item, then the data item referenced by *identifier-2* is incremented by the same amount as, and at the same time as, the index associated with *identifier-1* is incremented. If *identifier-2* is not an index data item, the data item referenced by *identifier-2* is incremented by the value one at the same time as the index referenced by the index-name associated with *identifier-1* is terminated.

(9) The END-SEARCH phrase delimits the scope of the SEARCH statement.

(10) A representation of the action of a Format 1 SEARCH statement containing two WHEN phrases is shown in the figure that follows. This figure is not intended to indicate the underlying implementation.

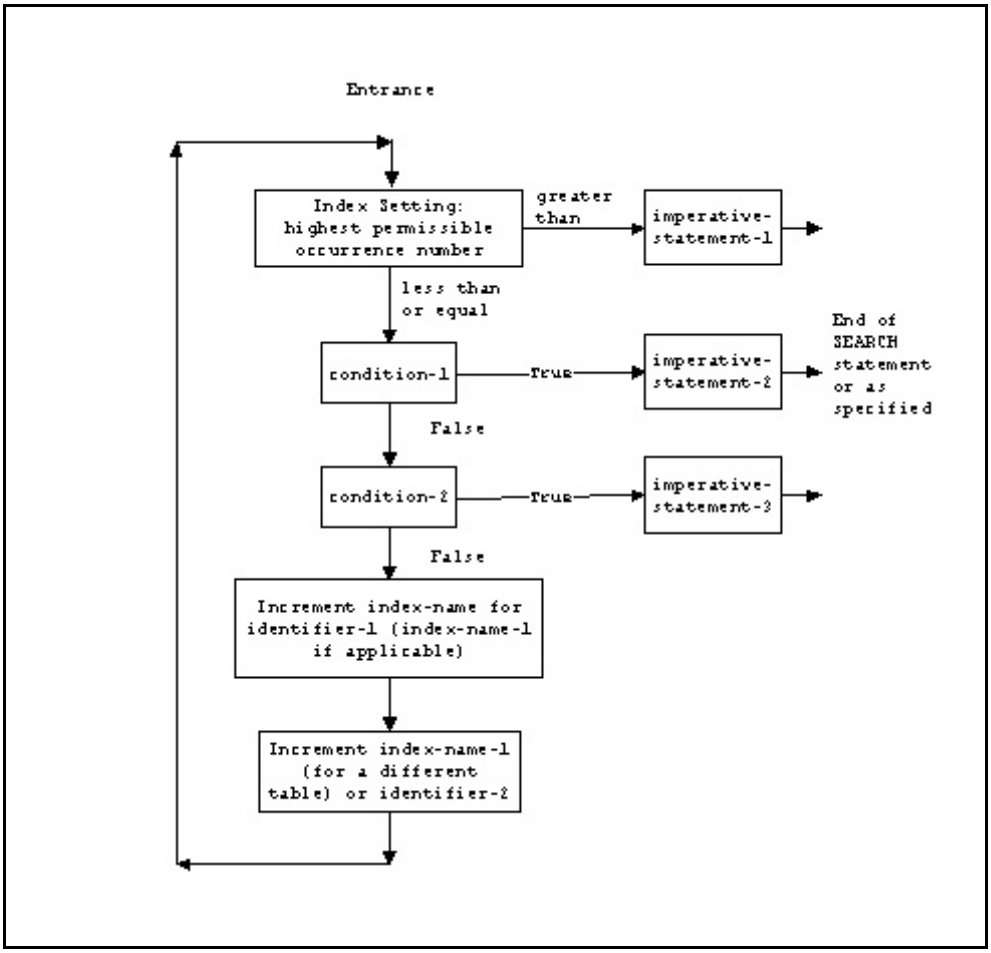


FIGURE 8. Format 1 SEARCH statement having two WHEN phrases

E.48. SET (***ANSI 74*** and ***ANSI 85***)

E.48.1 Function

The SET statement establishes reference points for table handling operations by setting indices associated with table elements. It also sets the values of condition names, mnemonic names, and pointer data items; and, (***ISQL***) the value of an indicator data item or the current database connection.

E.48.2 General Format

Format 1:

```
SET { index-name-1   } ... TO { index-name-2 }
    { identifier-1   }        { identifier-2 }
                              { integer-1    }
```

Format 2:

```
SET { index-name-3 }... { UP BY   } { identifier-3 }
                        { DOWN BY } { integer-2    }
```

Format 3:

```
SET { { mnemonic-name-1 }... TO { ON  }          }...
                               { OFF }
```

Format 4:

```
SET { condition-name-1 }... TO TRUE
```

Format 5:

```
SET { identifier-4 }... TO { ADDRESS OF identifier-5 }
                           { identifier-6            }
                           { NULL                    }
```

Format 6: (*ISQL*)

```
SET { identifier-7 }... TO { identifier-8 }
                           { NULL         }
                           { VALID        }
                           { OVERFLOW     }
```

E.48.3 Syntax Rules

(1) All references to *index-name-1*, *identifier-1*, and *index-name-3* apply equally to all recursions thereof.

(2) *Identifier-1* and *identifier-2* must each reference an index data item or an elementary item described as an integer.

(3) *Identifier-3* must reference an elementary numeric integer.

(4) *Integer-1* and *integer-2* may be signed. *Integer-1* must be positive.

(5) *Mnemonic-name-1* must be associated with an external switch, the status of which can be altered.

(6) *Condition-name-1* must be associated with a conditional variable.

(7) Every occurrence of *identifier-4* and *identifier-6* must reference a data item described as USAGE IS POINTER.

(8) *Identifier-5* may reference any data item defined in the Data Division.

(9) (***ISQL***) Every occurrence of *identifier-7* and *identifier-8* must reference a data item described as USAGE IS INDICATOR.

E.48.4 General Rules

Format 1 and 2:

(1) Index-names are associated with a given table by being specified in the INDEXED BY phrase of the OCCURS clause for that table.

(2) If *index-name-1* is specified, the value of the index after the execution of the SET statement must correspond to an occurrence number of an element in the table associated with *index-name-1*. The value of the index associated with an index-name after the execution of a PERFORM statement may be set to an occurrence number that is outside the range of its associated table.

If *index-name-2* is specified, the value of the index before the execution of the SET statement must correspond to an occurrence number of an element in the table associated with *index-name-1*.

If *index-name-3* is specified, the value of the index both before and after the execution of the SET statement must correspond to an occurrence number of an element in the table associated with *index-name-3*.

Note:	Rule 2 is not currently enforced by **ICOBOL**.

(3) In Format 1, the following action occurs:

a. *Index-name-1* is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by *index-name-2*, *identifier-2*, or *integer-1*. If *identifier-2* references an index data item, or if *index-name-2* is related to the same table as *index-name-1*, no conversion takes place.

b. If *identifier-1* references an index data item, it may be set equal to either the content of *index-name-2* or *identifier-2* where *identifier-2* also references an index data item; no conversion takes place in either case.

c. If *identifier-1* does not reference an index data item, it may be set only to an occurrence number that corresponds to the value of *index-name-2*. Neither *identifier-2* nor *integer-1* can be used in this case.

d. The process is repeated for each recurrence of *index-name-1* or *identifier-1*, if specified. Each time, the value of *index-name-2* or the data item referenced by *identifier-2* is used as it was at the beginning of the execution of the statement. Any subscripting associated with *identifier-1* is evaluated immediately before the value of the respective data item is changed.

(4) In Format 2, the content of *index-name-3* is incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of *integer-2* or the data item referenced by *identifier-3*; thereafter, the process is repeated for each recurrence of *index-name-3*. For each repetition the value of the data item referenced by *identifier-3* is used as it was at the beginning of the execution of the statement.

(5) Data in the following table represents the validity of various operand combinations in Format 1 of the SET statement. The general rule reference (after the slash) indicates the applicable general rule.

SENDING ITEM	RECEIVING DATA ITEM		
	INTEGER DATA ITEM	INDEX	INDEX DATA ITEM
Integer literal	No/3c	Valid/3a	No/3b
Integer data item	No/3c	Valid/3a	No/3b
Index	Valid/3c	Valid/3a	Valid/3b*
Index data item	No/3c	Valid/3a*	Valid/3b*

TABLE 30. Validity of Operand Combinations in Format 1 SET Statements

* No conversion takes place

Format 3:

(6) The status of each external switch associated with the specified *mnemonic-name-1* is modified such that the truth value resultant from evaluation of a condition-name associated with that switch will reflect an on status if the ON phrase is specified, or an off status if the OFF phrase is specified. See the Switch-Status condition on page 238

Format 4:

(7) The literal in the VALUE clause associated with *condition-name-1* is placed in the conditional variable according to the rules of the VALUE clause, see The VALUE Clause on page 195). If more than one literal is specified in the VALUE clause, the conditional variable is set to the value of the first literal that appears in the VALUE clause.

(8) If multiple condition-names are specified, the results are the same as if a separate SET statement had been written for each *condition-name-1* in the same order as specified in the SET statement.

Format 5:

(9) The address specified by the TO phrase is moved into *identifier-4*. This address will be valid until the program terminates or returns control to the calling program.

(10) The address specified by the TO phrase is the address contained in *identifier-6*, the address of *identifier-5*, or if NULL is specified, an address which points to no data-item.

Format 6:

(11) The value of the indicator specified by the TO phrase is moved into *identifier-7*.

E.49. SET (***VXCOBOL***)

E.49.1 Function

Sets one or more data items equal to another data item, or adds an operand to or subtracts an operand from one or more operands.

E.49.2 General Format

Format 1:

```
SET { index-name-1 } ... TO { index-name-2 }
    { identifier-1 }        { identifier-2 }
                            { literal-1    }
```

Format 2:

```
SET { identifier-4 }   { UP BY   }  { identifier-3 }
    { index-name-3 }   { DOWN BY } ... { literal-2 }
```

E.49.3 Syntax Rules

(1) All references to *index-name-1*, *identifier-1*, *index-name-3*, and *identifier-4* apply equally to all recursions thereof.

(2) *Identifier-3* and *identifier-4* must reference numeric data-items.

(3) *Literal-2* must be a numeric literal.

E.49.4 General Rules

Format 1 and 2:

(1) Index-names are associated with a given table by being specified in the INDEXED BY phrase of the OCCURS clause for that table.

(2) If *index-name-1* is specified, the value of the index after the execution of the SET statement must correspond to an occurrence number of an element in the table associated with *index-name-l.* The value of the index associated with an index-name after the execution of a PERFORM statement may be set to an occurrence number that is outside the range of its associated table.

If *index-name-2* is specified, the value of the index before the execution of the SET statement must correspond to an occurrence number of an element in the table associated with *index-name-1*.

If *index-name-3* is specified, the value of the index both before and after the execution of the SET statement must correspond to an occurrence number of an element in the table associated with *index-name-3*.

> **Note:** Rule 2 is not currently enforced by **ICOBOL**.

(3) In Format 1, the following action occurs:

a. *Index-name-1* or *identifier-1* is set to a value equal to the content of *index-name-2* or *identifier-2* or to the value specified by *literal-1*.

b. The value is set using the MOVE rules. Format 1 of the SET statement is equivalent to:

```
MOVE { index-name-2 } TO { index-name-1 } ...
     { identifier-2 }    { identifier-1 }
     { literal-1    }
```

c. The process is repeated for each recurrence of *index-name-1* or *identifier-1*. Each time, the value of *index-name-2* or *identifier-2* is used as it was at the beginning of the statement. Any subscripting associated with *identifier-1* is evaluated immediately before the value of the respective data-item is changed.

(4) In format 2, the following action occurs:

a. A SET UP statement adds the contents of *identifier-3* or the value of *literal-2* to *index-name-3* or *identifier-4* and stores it back in *index-name-3* or *identifier-4*, respectively, according to MOVE rules. The SET UP statement is equivalent to:

```
ADD { identifier-3 } TO { index-name-3 } ...
    { literal-2    }    { identifier-4 }
```

b. A SET DOWN statement subtracts the contents of *identifier-3* or the value of *literal-2* from *index-name-3* or *identifier-4* and stores it back in *index-name-3* or *identifier-4*, respectively, according to MOVE rules. The SET DOWN statement is equivalent to:

```
SUBTRACT { identifier-3 } FROM { index-name-3 } ...
         { literal-2    }     { identifier-4 }
```

c. The process is repeated for each recurrence of *index-name-3* or *identifier-4*. Each time, the value of *identifier-3* is used as it was at the beginning of the statement. Any subscripting associated with *identifier-4* is evaluated immediately before the value of the respective data-item is changed.

NOTE: The *VXCOBOL* implementation of the SET statement differs substantially from ANSI COBOL. ANSI COBOL uses SET to establish reference points for table handling operations by setting indices associated with table elements. As such, *identifier-1* and *identifier-2* must reference index data items or elementary integer items. *Identifier-3* must be an elementary numeric integer. *Literal-1* must be a positive integer, and *literal-2* must be an integer.

E.50. SET CONNECTION (*ISQL*)

E.50.1 Function

Set the currently active SQL database connection.

E.50.2 General Format

```
SET CONNECTION { DEFAULT
               { literal-1
               { identifier-1 }
    [ ON SQLERROR imperative-statement-1 ]
    [ NOT ON SQLERROR imperative-statement-2 ]
    [ END-SET ]
```

E.50.3 Syntax Rules

(1) *Literal-1* must be an alphanumeric literal and may not specify a figurative constant.

(2) *Identifier-1* must be an alphanumeric data item.

E.50.4 General Rules

(1) If the DEFAULT phrase is used, it specifies that the default connection (which has the name “default”) is to be made the currently active connection.

(2) The value of *literal-1* or the content of the data item represented by *identifier-1* specifies the name of the connection that is to be made the currently active connection.

(3) Connections are kept on a run unit basis, i.e., the scope of the connection name is the entire run unit, not just the program containing the SET CONNECTION statement. If the specified connection does not exist, it is an error and SQLSTATE will be set to “08003", which is “Connection does not exist”.

(4) If there is a currently active connection and it differs from the connection specified by the SET CONNECTION statement, it is made the most recent dormant connection.

(5) Upon completion of the SET CONNECTION statement, the following occurs in the order specified:

a. The value of the SQLSTATE data item is updated with the status of the operation.

b. If the value of the SQLSTATE class field is “00", the statement is successful. Control is transferred to the end of the SET CONNECTION statement or to *imperative-statement-2*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the SET CONNECTION statement.

c. If the value of the SQLSTATE class field is not “00", the statement is unsuccessful. The statement container is deallocated and no statement container of the specified name will exist in the current program. Control is transferred to the end of the SET CONNECTION statement or to *imperative-statement-1*, if specified. In the latter case, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for the statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the SET CONNECTION statement.

(6) The END-SET phrase delimits the scope of the SET CONNECTION statement.

(7) More on SQLSTATE can be found on page 133.

E.51. SORT

E.51.1 Function

The SORT statement creates a sort file by executing an input procedure or by transferring records from another file, sorts the records in the sort file on a set of specified keys; and in the final phrase of the sort operation, makes available each record from the sort file, in sorted order, to an output procedure or to an output file.

E.51.2 General Format (***ANSI 74*** and ***ANSI 85***)

```
SORT file-name-1 { ON { ASCENDING  }                KEY { data-name-1 }... }...
                     { DESCENDING }
    [ WITH DUPLICATES IN ORDER ]
    [ COLLATING SEQUENCE IS alphabet-name ]
    { INPUT  PROCEDURE IS procedure-name-1 [ { THROUGH } procedure-name-2 ] }
                                           { THRU    }
                          USING { file-name-2 }...
    { OUTPUT PROCEDURE IS procedure-name-3 [ { THROUGH } procedure-name-4 ] }
                                           { THRU    }
                          GIVING { file-name-3 }...
```

E.51.3 General Format (***VXCOBOL***)

```
SORT file-name-1
d   [ literal [ CREATE MAXIMUM RECORDS { identifier } ] [ SAVE ] ]
                                       { literal    }
    [ ON { ASCENDING  }                KEY [ data-name-1 ]... ]...
         { DESCENDING }
    [ WITH DUPLICATES IN ORDER ]
    [ COLLATING SEQUENCE IS { ASCII      } ]
                            { NATIVE     }
                            { STANDARD-1 }
                            { EBCDIC     }
                            { alphabet-name }
    { INPUT  PROCEDURE IS procedure-name [ { THROUGH } procedure-name ] }
                                         { THRU    }
                          USING { file-name-2 } ...
    { OUTPUT PROCEDURE IS procedure-name [ { THROUGH } procedure-name ] }
                                         { THRU    }
                          GIVING { file-name-3 } ...
```

E.51.4 Syntax Rules

(1) A SORT statement may appear anywhere in the Procedure Division except in the declaratives portion.

(2) *File-name-1* must be described in a sort-merge file description entry in the Data Division.

(3) If the USING phrase is specified and the file referenced by *file-name-1* contains variable length records, the size of the records contained in the files referenced by *file-name-2* must not be less than the smallest record nor greater than the largest record described for *file-name-1*. If the file referenced by *file-name-1* contains fixed length records, the sizes of the records contained in the file referenced by *file-name-2* must not be greater than the largest record described for *file-name-1*.

(4) *Data-name-1* is a key data-name. Key data-names are subject to the following rules:

a. The data items identified by key data-names must be described in records associated with *file-name-1*.

b. Key data-names may be qualified.

c. Key data-names may not be described as USAGE POINTER.

d. The data items identified by key data-names must not be group items that contain variable occurrence data items.

e. If *file-name-1* has more than one record description, the data items identified by key data-names need be described in only one record description. The same character positions referenced by a key data-name in one record description entry are taken as the key in all records of the file.

f. None of the data items identified by key data-names can be described by an entry that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.

g. If a file referenced by *file-name-1* contains variable length records, all the data items identified by key data-names must be contained within the first x characters positions of the record, where x equals the minimum record size specified for the file referenced by *file-name-1*.

h. For ***VXCOBOL***, if no *data-name-1* is specified the entire record is used as the key.

(5) The words THRU and THROUGH are equivalent.

(6) *File-name-2* and *file-name-3* must be described in a file description entry, not a sort-merge description entry, in the Data Division.

(7) No pair of file-names in a SORT statement may be specified in the same SAME SORT AREA or SAME SORT-MERGE AREA clause. File-names associated with the GIVING phrase may not be specified in the same SAME clause.

(8) If *file-name-3* is specified it is subject to the following rules:

a. If *file-name-3* references an indexed file, the first specification of *data-name-1* and the data item referenced by that *data-name-1* must occupy the same character positions in its record as the data item associated with the prime record key for that file. For **ANSI 74** and **ANSI 85**, the first specification of *data-name-1* must be associated with the ASCENDING phrase if *file-name-3* has a primary record key described explicitly or implicitly as VALUES ARE ASCENDING. If the key is described as VALUES ARE DESCENDING, *data-name-1* must be associated with the DESCENDING phrase. For ***VXCOBOL***, the first specification of *data-name-1* must be associated with the ASCENDING phrase.

b. For ***VXCOBOL***, If *file-name-3* references an INFOS file, it must not allow subindexing, and the first specification of *data-name-1* must be associated with an ASCENDING phrase. The data-item referenced by

data-name-1 must occupy the same character positions in its record as the data item associated with the first RECORD KEY in the SELECT for *file-name-3*, i.e., the RECORD KEY and sort key must be internal to the record.

(9) For ***VXCOBOL***, if *file-name-2* references an INFOS file, it must not all subindexing.

(10) If the GIVING phrase is specified and the file referenced by *file-name-3* contains variable length records, the size of the records contained in the file referenced by *file-name-1* must not be less that the smallest record nor greater that the largest record described for *file-name-3*. If the file referenced by *file-name-3* contains fixed length records, the size of the records contained in the file referenced by *file-name-1* must not be greater that the largest record described for *file-name-3*.

(11) Alphabet-name shall reference an alphabet defined in the SPECIAL-NAMES paragraph which defines an alphanumeric collating sequence.

(12) For ***VXCOBOL***, the CREATE clause is for documentation purposes only.

(13) If *file-name-2* references an indexed, relative, or INFOS file its access mode shall be sequential or dynamic.

(14) For ***VXCOBOL***, if the ASCENDING or DESCENDING clause is not specified then ASCENDING is assumed, and the entire record is used as the key.

E.51.5 General Rules

(1) If the file referenced by *file-name-1* contains only fixed length records, any record in the file referenced by *file-name-2* containing fewer character positions that fixed length is space filled on the right beginning with the first character position after the last character in the record when that record is released to the file referenced by *file-name-1*.

(2) The data-names following the word KEY are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY phrases. The leftmost data-name is the major key, the next data-name is the next most significant key, etc.

a. When the ASCENDING phrase is specified, the sorted sequence will be from the lowest value of the contents of the data items identified by the key data-names to the highest value, according to the rules for comparison of operands in a relation condition.

b. When the DESCENDING phrase is specified, the sorted sequence will be from the highest value of the contents of the data items identified by the key data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

(3) If the DUPLICATES phrase is specified and the contents of all the key data items associated with one data record are equal to the contents of the corresponding key data items associated with one or more other data records, then the order of return of these records is:

a. The order of the associated input files as specified in the SORT statement. Within a given input file the order is that in which the records are accessed from that file.

b. The order in which these records are released by an input procedure, when an input procedure is specified.

(4) If the DUPLICATES phrase is not specified and the contents of all the key data items associated with one data record are equal to the contents of the corresponding key data items associated with one or more data records, then the order of return of these records is undefined.

(5) The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined at the beginning of the execution of the SORT statement in the following order of precedence:

a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in the SORT statement.

b. Second, the collating sequence established as the program collating sequence. In **ICOBOL** this is always ASCII since the program collating sequence is ignored.

(6) The execution of a SORT statement consists of three distinct phases as follows:

a. Records are made available to the file referenced by *file-name-1*. This is achieved either by the execution of RELEASE statements in the input procedure or by the implicit execution of READ statements for *file-name-2*. When this phrase commences, the file referenced by *file-name-2* must not be in the open mode. When this phrase terminates, the file referenced by *file-name-2* is not in the open mode.

b. The file referenced by *file-name-1* is sequenced. No processing of the files referenced by *file-name-2* and *file-name-3* takes places during this phase.

c. The records of the file referenced by *file-name-1* are made available in sorted order. The sorted records are either written to the file referenced by *file-name-3* or, by execution of a RETURN statement, are made available for processing by the output procedure. When this phase commences, the file referenced by *file-name-3* must not be in the open mode. When this phase terminates, the file referenced by *file-name-3* is not in the open mode.

(7) The input procedure may consist of any procedure needed to select, modify, or copy the records that are made available one at a time by the RELEASE statement to the file referenced by *file-name-1*. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the input procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the input procedure. The range of the input procedure must not cause the execution of any MERGE, RETURN, or SORT statement.

(8) If an input procedure is specified, control is passed to the input procedure before the file referenced by *file-name-1* is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last statement in the input procedure and when control passes the last statement in the input procedure, the records that have been released to the file referenced by *file-name-1* are sorted.

(9) If the USING phrase is specified, all the records in the file(s) referenced by *file-name-2* are transferred to the file referenced by *file-name-1*. For each of the files referenced by *file-name-2* the execution of the SORT statement causes the following actions to be taken:

a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the INPUT phrase had been executed.

b. The logical records are obtained and released to the sort operation. Each record is obtained as if a READ statement with the NEXT and the AT END phrases had been executed. When the at end condition exists for *file-name-1*, the processing for that file connector is terminated. If the file referenced by *file-name-1* is described with variable-length records, the size of any record released to *file-name-1* is the size of that record when it was read from *file-name-2*, regardless of the content of the data item referenced by the DEPENDING ON phrase of either a RECORD IS VARYING clause or an OCCURS clause specified in the sort-merge file description entry for *file-name-1*. If the size of the record read from the file referenced by *file-name-2* is larger than the largest record allowed in the file description entry for *file-name-1* or is smaller than the smallest record allowed in the file description entry for *file-name-1*, an exception condition exists and the execution of the SORT statement is terminated.

For a relative file, the content of the relative key data items is undefined after the execution of the SORT statement if *file-name-2* is not referenced in the GIVING phrase.

c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. This termination is performed before the file referenced by *file-name-1* is sequenced by the SORT statement.

These implicit functions are performed such that any associated USE AFTER STANDARD EXCEPTION procedures are executed; however, the execution of such a USE procedure must not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, *file-name-2*.

(10) The output procedure may consist of any procedure needed to select, modify, or copy records that are made available one at a time by the RETURN statement in sorted order from the file referenced by *file-name-1*. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the output procedure, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the output procedure. The range of the output procedure must not cause the execution on any MERGE, RELEASE, or SORT statement. See page 253, 306, Explicit and Implicit specifications.

(11) If an output procedure is specified, control passes to it after the file referenced by *file-name-1* has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last statement in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the merge, and then passes control to the next executable statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

(12) If the GIVING phrase is specified, all the sorted records are written on the file referenced by *file-name-3* as the implied output procedure for the SORT statement. For each of the files referenced by *file-name-3*, the execution of the SORT statement causes the following actions to be taken:

a. The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed. This initiation is performed after the execution of any input procedure.

b. The sorted logical records are returned and written onto the file. Each record is written as if a WRITE statement without any optional phrases had been executed. If the file referenced by *file-name-3* is described with variable length records, the size of any record written to *file-name-3* is the size of that record when it was read from *file-name-1*, regardless of the content of the data item referenced by the DEPENDING ON phrase of either a RECORD IS VARYING clause or an OCCURS clause specified in the file description entry for *file-name-3*.

For a relative file, the relative key date for the first record returned contains the value '1'; for the second record returned, the value '2', etc. After execution of the SORT statement, the content of the relative key data item indicates the last record returned to the file.

c. The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

These implicit functions are performed such that any associated USE AFTER STANDARD EXCEPTION procedures are executed; however, the execution of such a USE procedure must not cause the execution of any statement manipulating the file referenced by, or accessing the record area associated with, *file-name-3*. On the first attempt to write beyond the externally defined boundaries of the file, any USE AFTER STANDARD EXCEPTION procedure specified for that file is executed; if control is returned from that USE procedure or if no USE procedure is specified, the processing of the file is terminated as in paragraph 12c above.

(13) If the file referenced by *file-name-3* contains only fixed length records, any record in the file referenced by *file-name-1* containing fewer character positions that fixed length is space filled on the right beginning with the first character position after the last character in the record when that record is returned to the file referenced by *file-name-3*.

(14) The environment entry ICTMPDIR is used for temporary files.

(15) An ACCEPT FROM EXCEPTION should be done after this operation to ensure no errors.

E.52. START

E.52.1 Function

The START statement provides a basis for logical positioning within a relative, indexed, or INFOS file, or for a fixed sequential file for subsequent sequential retrieval of records.

E.52.2 General Format

Format 1: (Relative or Indexed)

ANSI 74 and ANSI 85:

```
START file-name [ KEY IS { =                                  } key-name ]
                         { EQUAL TO                           }
                         { >                                  }
                         { GREATER THAN                       }
                         { >=                                 }
                         { GREATER THAN OR EQUAL TO           }
                         { NOT <                              }
                         { NOT LESS THAN                      }
                         { <                                  }
                         { LESS THAN                          }
                         { <=                                 }
                         { LESS THAN OR EQUAL TO              }
                         { NOT >                              }
                         { NOT GREATER THAN                   }
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-START ]
```

VXCOBOL

```
START file-name [ KEY IS { =                  } data-name ]
                         { EQUAL TO           }
                         { >                  }
                         { GREATER THAN       }
                         { NOT <              }
                         { NOT LESS THAN      }
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-START ]
```

Format 2: (Relative or Indexed)

ANSI 74 and ANSI 85

```
START file-name { FIRST } [ KEY IS key-name ]
                { LAST  }
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-START ]
```

Format 3: (Sequential)

ANSI 74 and ANSI 85

```
                  { RECORD { identifier-1 } [ CHARACTER { identifier-2 } ] }
                  {        { integer-1    }             { integer-2    }   }
START file-name   {                                                       }
                  {                CHARACTER { identifier-2 }               }
                  {                          { integer-2    }               }
                  {                AFTER LAST CHARACTER                     }
                  {                  FIRST CHARACTER                        }
```

VXCOBOL

```
START file-name RECORD { identifier-1 }          [ CHARACTER { identifier-2 }          ] [ END-START ]
                       { integer-1    }                     { integer-2    }
```

E.52.3 Syntax Rules

(1) *File-name* must be the name of a file with a sequential or dynamic access.

(2) *Key-name* may be qualified if *id-1* is a simple data item. *Key-name* may be qualified by the filename if it is a composite data item.

(3) The INVALID KEY phrase must be specified if no applicable USE AFTER STANDARD EXCEPTION procedure is specified for *file-name*.

For relative files:

(4) *Key-name*, if specified, must be the data item specified in the RELATIVE KEY phrase in the ACCESS MODE clause of the associated file control entry.

For indexed and INFOS Files:

(5) If the KEY IS phrase is specified, *key-name* must reference

a. A key associated with *file-name* (in *id-1* in the formats of the RECORD KEY or ALTERNATE RECORD KEY).

b. Any data-item of category alphanumeric whose leftmost character position within a record of the file corresponds to the leftmost character position of a record key or the root segment of a record key and whose length is not greater than the length of that key or root segment.

(6) In the case where multiple alternate keys start at the same position, **ICOBOL** matches it up with the first key whose size is larger than key-name. NOT ANSI STANDARD.

For sequential files:

(7) *Identifier-1*, *identifier-2*, *integer-1* and *integer-2* must be integers that are greater than or equal to zero.

(8) If RECORD is specified, the file description entry for *file-name* may not contain the RECORD IS VARYING or RECORDING MODE IS VARIABLE clause. The file control entry must explicitly or implicitly be ASSIGN TO DISK.

(9) If RECORD is specified, *identifier-2* or *integer-2* must be less than or equal to the maximum record size.

(10) For ***VXCOBOL***, the file description entry associated with *file-name* must have the RECORDING MODE IS FIXED.

E.52.4 General Rules

(1) The file referenced by *file-name* must be open in the input or I-O mode at the time that the START statement is executed. (See The OPEN Statement, page 399.)

(2) For Format 1, if the KEY phrase is not specified, the relational operator `IS EQUAL TO' is implied.

(3) The execution of the START statement does not alter the content of the record area.

(4) The execution of the START statement causes the value of the I-O status associated with file-name to be updated.

(5) Transfer of control following the successful or unsuccessful execution of the START operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the START statement.

(6) Following the unsuccessful execution of a START statement, the file position indicator is set to indicate that no valid next record has been established.

(7) The END-START phrase delimits the scope of the START statement.

For relative files:

(8) The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name and a data item as specified in general rule 8. Numeric comparison rules apply. (See Comparison of Numeric Operands, page 234.)

a. The file position indicator is set to the relative record number of the first logical record in the file whose key satisfies the comparison.

b. If the comparison is not satisfied by any record in the file, the invalid key condition exists and the execution of the START statement is unsuccessful.

(9) The comparison described in general rule 8 uses the data item referenced by the RELATIVE KEY phrase of the ACCESS MODE clause associated with file-name.

For indexed files:

(10) The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by *file-name-1* and a data item as specified in general rules 12 and 13. The comparison is made on the ascending key of reference according to the collating sequence of the file. If the operands are of unequal size, comparison proceeds as though the longer one was truncated on the right such that its length is equal to that of the shorter. All other nonnumeric comparison rules apply. (See Comparison of Nonnumeric Operands, page 234.)

a. The file position indicator is set to the value of the key of reference in the first logical record whose key satisfies the comparison.

b. If the comparison is not satisfied by any record in the file, the invalid key condition exists and the execution of the START statement is unsuccessful,

(11) A key of reference is established as follows:

a. If the KEY phrase is not specified, the primary record key specified for file-name becomes the key of reference.

b. If the KEY phrase is specified, and data-name is specified as a record key for file-name, that record key becomes the key of reference.

c. If the KEY phrase is specified, and data-name is not specified as a record key for file-name, the record key whose left-most character position corresponds to the left-most character position of the data item specified by data-name, becomes the key of reference.

d. In the case where multiple alternate keys start at the same position **ICOBOL** matches it up with the first key whose size is larger than data-name. **NOT ANSI STANDARD.**

This key of reference is used to establish the ordering of records for the purpose of this START statement, see general rule 10; and, if the execution of the START statement is successful, the key of reference is also used for subsequent sequential READ statements. (See The READ Statement, page 412, 414, 420.)

(12) If the KEY phrase is specified, the comparison described in general rule 5 uses:

a. The *data-item* specified by *key-name,* if the RECORD KEY or ALTERNATE RECORD KEY clause of the file control entry for *file-name* does not include the equal sign (=).

b. The composite key specified by *key-name*, if the RECORD KEY or ALTERNATE RECORD KEY clause of the file control entry for *file-name* includes the PLUS phrase.

c. The first occurrence of the data-item or composite key specified by *key-name* if the ALTERNATE RECORD KEY clause of the file control entry for *file-name* includes the OCCURS phrase.

d. The root key of *key_name* if the ALTERNATE RECORD KEY clause of the file control entry for *file-name* includes the ALSO phrase. (Ths is *id-2* in the ALTERNATE RECORD KEY format.)

(13) If the KEY phrase is not specified, the comparison described in general rule 5 uses the data item or composite key referenced in the RECORD KEY clause associated with file-name.

(14) The keyword FIRST is used to position to the first record in the file for the key of reference.

(15) The keyword LAST is used to position to the last record in the file for the key of reference.

For sequential files:

(16) If the RECORD phrase is specified. The file position indicator will be positioned to the character position computed by:

(record length * record number) + character offset

where record length is the fixed length of the records associated with *file-name*, record number is *integer-1* or the contents of *identifier-1*, and character offset is *integer-2* or the contents of *identifier-2*. If the CHARACTER phrase is omitted, *integer-2* is assumed to be zero.

(17) If the CHARACTER phrase is specified without the RECORD phrase, positioning occurs as in general rule 16 where *integer-1* is assumed to be zero.

(18) The first record in the file is assumed to be record zero, the second record is record 1. Etc. Thus to position to the Nth record, *integer-1* or *identifier-1* should be set to N-1.

(19) The AFTER LAST phrase set the file position indicator following the last character of the file (i.e. end-of-file), This is equivalent to the position immediately after an OPEN EXTEND.

(20) The FIRST CHARACTER phrase set the file position indicator to the first character of the file. This is equivalent to the position immediately after an OPEN INPUT. The FIRST CHARACTER phrase is equivalent to CHARACTER 0.

For INFOS Files:

(21) The type of of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by *file-name-1* and a data item as specified in general rules 22 and 23. The comparison is made on the ascending key of reference according to the collating sequence of the file. If the operands are of unequal size, comparison proceeds as though the longer one was truncated on the right such that its length is equal to that of the shorter. All other nonnumeric comparison rules apply. (See Comparison of Nonnumeric Operands, page 234.)

a. The file position indicator is set to the value of the key of reference in the first logical record whose key satisfies the comparison.

b. If the comparison is not satisfied by any record in the file, the invalid key condition exists and the execution of the START statement is unsuccessful,

(22) If the KEY phrase is not specified, the comparison described in general rule 5 uses the first *data-item* referenced in the RECORD KEY clause associated with *file-name*.

(23) If the KEY phrase is specified, the key refers to the file's top level. If you have specified DUPLICATES and the OCCURRENCE options in the file's SELECT clause, and the occurrence number is not zero, then START uses the occurrence number to refer to the key.

(24) After a START statement, **ICOBOL** treats a READ NEXT as a READ STATIC.

(25) The FEEDBACK *data-item* is not updated by a START.

E.53. STOP

E.53.1 Function

The STOP statement causes a permanent or temporary suspension of the execution of the run unit. The literal variation of the STOP statement is an obsolete element in Standard COBOL because it is to be deleted from the next revision of Standard COBOL.

E.53.2 General Format

```
STOP { RUN [ literal ] }
     { literal         }
```

E.53.3 Syntax Rules

(1) *Literal* must not be a figurative constant that begins with the word ALL.

(2) If a STOP RUN statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

(3) If *literal* is numeric, then it must be an unsigned integer.

E.53.4 General Rules

(1) If the RUN phrase is specified, execution of the run unit ceases and control is transferred to the operating system. If the optional *literal* is specified, it is displayed before the run unit ceases.

(2) During the execution of a STOP RUN statement, an implicit CLOSE statement without any optional phrases is executed for each file that is in the open mode in the run unit. Any USE procedures associated with any of these files are not executed.

(3) A STOP RUN *literal* will cause a value to be returned as the exit code from ICRUN. If *literal* is numeric, and 10 < *literal* < 255, then the integer portion of *literal* is returned; otherwise, the value 10 is returned. Exit codes 0 through 9 are reserved for the standard exit codes of the runtime system.

(4) If STOP *literal* is specified, the execution of the run unit is suspended and *literal* is communicated to the operator. Continuation of the execution of the run unit begins with the next executable statement when a newline has been entered or a STOP RUN is executed if an ESCAPE is entered.

(5) When using timeouts, **ICOBOL** handles them in the following order:

a. If a timeout had been set with the IC_SET_TIMEOUT builtin, then it is used; otherwise,

b. The global timeout as set with ICTIMEOUT will be used. The default case for global timeout is to wait forever.

NOTE:

Using an extended open option to set timeout on your console does NOT affect an ACCEPT or STOP statement. Extended open options are discussed in the Interactive COBOL Developer's Guide Section.

IC_SET_TIMEOUT is discussed in this document beginning on page 558, 559.

E.54. STRING

E.54.1 Function

The STRING statement provides juxtaposition of the partial or complete contents of one or more data items into a single data item.

E.54.2 General Format

STRING { { *identifier-1* / *literal-1* }... DELIMITED BY { *identifier-2* / *literal-2* / SIZE } }... INTO *identifier-3*

[WITH POINTER *identifier-4*]
[ON OVERFLOW *imperative-statement-1*]
[NOT ON OVERFLOW *imperative-statement-2*]
[END-STRING]

E.54.3 Syntax Rules

(1) *Literal-1* or *literal-2* must not be a figurative constant that begins with the word ALL.

(2) All literals must be described as nonnumeric literals, and all identifiers, except *identifier-4*, must be described implicitly or explicitly as USAGE IS DISPLAY.

(3) *Identifier-3* must not be reference modified.

(4) *Identifier-3* must not represent an edited data item and must not be described with the JUSTIFIED clause.

(5) *Identifier-4* must be described as an elementary numeric integer data item of sufficient size to contain a value equal to 1 plus the size of the data item referenced by *identifier-3*. The symbol `P' may not be used in the PICTURE character-string of *identifier-4*.

(6) Where *identifier-1* or *identifier-2* is an elementary numeric data item, it must be described as an integer without the symbol `P' in its PICTURE character-string.

(7) *Identifier-1* may not be a function identifier.

E.54.4 General Rules

(1) *Identifier-1* or *literal-1* represents the sending item. *Identifier-3* represents the receiving item.

(2) *Literal-2* or the content of the data item referenced by *identifier-2* indicates the character(s) delimiting the move. If the SIZE phrase is used, the content of the complete data item defined by *identifier-1* or *literal-1* is moved. When a figurative constant is used as the delimiter, it is a single character nonnumeric literal.

(3) When a figurative constant is specified as *literal-1* or *literal-2*, it refers to an implicit one character data item whose usage is DISPLAY.

(4) When the STRING statement is executed, the transfer of data is governed by the following rules:

a. Those characters from *literal-1* or from the content of the data item referenced by *identifier-1* are transferred to the data item referenced by *identifier-3* in accordance with the rules for alphanumeric to alphanumeric moves, except that no space filling will be provided.

b. If the DELIMITED phrase is specified without the SIZE phrase, the content of the data item referenced by *identifier-1*, or the value of *literal-1*, is transferred to the receiving data item in the sequence specified in the STRING statement beginning with the left-most character and continuing from left to right until the end of the sending data item is reached or the end of the receiving data item is reached or until the character(s) specified by *literal-2*, or by the content of the data item referenced by *identifier-2*, are encountered. The character(s) specified by *literal-2* or by the data item referenced by *identifier-2* are not transferred.

c. If the DELIMITED phrase is specified with the SIZE phrase, the entire content of *literal-1*, or the content of the data item referenced by *identifier-1*, is transferred, in the sequence specified in the STRING statement, to the data item referenced by *identifier-3* until all data has been transferred or the end of the data item referenced by *identifier-3* has been reached.

This behavior is repeated until all occurrences of *literal-1* or data items referenced by *identifier-1* have been processed.

(5) If the POINTER phrase is specified, the data item referenced by *identifier-4* must be set to an initial value greater than zero prior to the execution of the STRING statement.

(6) If the POINTER phrase is not specified, the following general rules apply as if the user had specified *identifier-4* referencing a data item with an initial value of 1.

(7) When characters are transferred to the data item referenced by *identifier-3*, the moves behave as though the characters were moved one at a time from the source into the character positions of the data item referenced by *identifier-3* designated by the value of the data item referenced by *identifier-1* (provided the value of the data item referenced by *identifier-4* does not exceed the length of the data item referenced by *identifier-3*), and then the data item referenced by *identifier-4* was increased by one prior to the move of the next character or prior to the end of execution of the STRING statement. The value of the data item referenced by *identifier-4* is changed during execution of the STRING statement only by the behavior specified above.

(8) At the end of execution of the STRING statement, only the portion of the data item referenced by *identifier-3* that was referenced during the execution of the STRING statement is changed. All other portions of the data item referenced by *identifier-3* will contain data that was present before this execution of the STRING statement.

(9) Before each move of a character to the data item referenced by *identifier-3*, if the value associated with the data item referenced by *identifier-4* is either less than one or exceeds the number of character positions in the data item referenced by *identifier-3*, no (further) data is transferred to the data item referenced by *identifier-3*, and the NOT ON OVERFLOW phrase, if specified, is ignored, and control is transferred to the end of the STRING statement or, if the ON OVERFLOW phrase is specified, to *imperative-statement-1*. If control is transferred to *imperative-statement-1*, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the STRING statement.

(10) If, at the time of execution of a STRING statement with the NOT ON OVERFLOW phrase, the conditions described in General Rule 9 are not encountered, after completion of the transfer of data according to the other general rules, the ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the STRING statement or, if the NOT ON OVERFLOW phrase is specified to *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the STRING statement.

(11) If *identifier-1* or *identifier-2* occupies the same storage area as identifier-3, or *identifier-4*, or if *identifier-3* and *identifier-4* occupy the same storage area, the result of the execution of this statement is undefined, even if they are defined by the same data description entry.

(12) The END-STRING phrase delimits the scope of the STRING statement.

E.55. SUBTRACT

E.55.1 Function

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from an item, and set the value of an item equal to the results.

E.55.2 General Format

Format 1:

```
SUBTRACT { identifier-1 }...        FROM { identifier-2 [ ROUNDED ] }...
         { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-SUBTRACT ]
```

Format 2:

```
SUBTRACT { identifier-1 }...  { identifier-2 }  FROM  GIVING { identifier-3 [ ROUNDED ] }...
         { literal-1    }     { literal-2    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-SUBTRACT ]
```

Format 3:

```
SUBTRACT { CORRESPONDING } identifier-1 FROM identifier-2 [ ROUNDED ]
         { CORR          }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ]
    [ END-SUBTRACT ]
```

E.55.3 Syntax Rules

(1) Each identifier must refer to a numeric elementary item except that:

a. In Format 2, each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

b. In Format 3, each identifier must refer to a group item.

(2) Each literal must be a numeric literal.

(3) The composite of operands must not contain more than 18 digits.

a. In Format 1, the composite of operands is determined by using all of the operands in a given statement.

b. In Format 2, the composite of operands is determined by using all of the operands in a given statement excluding the data item that follows the word GIVING.

c. In Format 3, the composite of operands is determined separately for each pair of corresponding data items.

(4) CORR is an abbreviation for CORRESPONDING.

E.55.4 General Rules

(1) When Format 1 is used, the values of the operands preceding the word FROM are added together and the sum is stored in a temporary data item. The value in this temporary data item is subtracted from the value of the data item referenced by *identifier-2*, storing the result into the data item referenced by *identifier-2*, and repeating this process for each successive occurrence of *identifier-2* in the left-to-right order in which *identifier-2* is specified.

(2) In Format 2, all literals and the values of the data items referenced by the identifiers preceding the word FROM are added together, the sum is subtracted from *literal-2* or the value of the data item referenced by *identifier-2*, and the result of the subtraction is stored as the new content of each data item referenced by *identifier-3*.

(3) If Format 3 is used, data items in *identifier-1* are subtracted from and stored into corresponding data items in *identifier-2*.

(4) The compiler insures enough places are carried so as not to lose significant digits during execution.

(5) Additional rules and explanations relative to this statement are given under the appropriate paragraphs. (See Scope of Statements, page 253; The ROUNDED Phrase, page 245; The ON SIZE ERROR Phrase, page 246; The Arithmetic Statements, page 249; Overlapping Operands, page 249; Multiple Results in Arithmetic Statements, page 249, and The CORRESPONDING Phrase, page 246.)

E.56. UNDELETE (***ANSI 74*** and ***ANSI 85***)

E.56.1 Function

The UNDELETE statement restores a logically deleted record to a relative or indexed file. UNDELETE is an extension to ANSI COBOL.

E.56.2 General Format

UNDELETE *file-name* RECORD
[INVALID KEY *imperative-statement-1*]
[NOT INVALID KEY *imperative-statement-2*]
[END-UNDELETE]

E.56.3 Syntax Rules

(1) *File-name* must be the name of a file with dynamic or random access.

(2) The INVALID KEY phrase must be specified for an UNDELETE statement which references a file for which an applicable USE AFTER STANDARD EXCEPTION procedure is not specified.

E.56.4 General Rules

(1) The file referenced by *file-name* must be a mass storage file and must be open in the I-O mode at the time the UNDELETE statement is executed.

(2) For a relative file, the file system logically restores to the file that record identified by the content of the relative key data item associated with *file-name*. If the file does not contain the logically deleted record specified by the key, the invalid key condition exists.

(2) For an indexed file, the file system logically restores to the file that record identified by the content of the RECORD KEY data item associated with the *file-name*. If the file does not contain the logically deleted record specified by the key, the invalid key condition exists.

(3) After the successful execution of an UNDELETE statement, the identified record has been logically restored to the file and can now be accessed.

(4) The execution of a UNDELETE statement does not affect the content of the record area.

(5) The file position indicator is not affected by the execution of a UNDELETE statement.

(6) The execution of the UNDELETE statement causes the value of the I-O status associated with *file-name* to be updated.

(7) Transfer of control following the successful or unsuccessful execution of the UNDELETE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the UNDELETE statement.

E.57. UNDELETE (***VXCOBOL***)

E.57.1 Function

The UNDELETE statement restores a logically deleted record to an indexed or INFOS file. UNDELETE is an extension to ANSI COBOL.

E.57.2 General Format

For indexed files:

```
UNDELETE file-name RECORD [ LOGICAL { LOCAL          } ]                    [ KEY IS identifier-1 ]
                                    { GLOBAL         }
                                    { LOCAL GLOBAL   }
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-UNDELETE ]
```

For INFOS files:

```
UNDELETE file-name [ { FIX    } POSITION [ NEXT          ] ]
                   { RETAIN }            [ FORWARD       ]
                                         [ BACKWARD      ]
                                         [ UP            ]
                                         [ DOWN          ]
                                         [ UP FORWARD    ]
                                         [ UP BACKWARD   ]
                                         [ DOWN FORWARD  ]
                                         [ STATIC        ]

    RECORD [ LOGICAL { LOCAL          } ] [ { KEY IS  } { identifier-2 [ APPROXIMATE ] }... ]
                     { GLOBAL         }     { KEYS ARE }               [ GENERIC     ]
                     { LOCAL GLOBAL   }
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-UNDELETE ]
```

E.57.3 Syntax Rules

(1) File-name must be the name of a file with dynamic or random access.

(2) The INVALID KEY phrase must be specified for an UNDELETE statement which references a file for which an applicable USE AFTER STANDARD EXCEPTION procedure is not specified.

(3) Identifier-1 must be the data-name specified as the RECORD KEY for the SELECT clause for file-name.

E.57.4 General Rules

(1) The file referenced by file-name must be a mass storage file and must be open in the I-O mode at the time the UNDELETE statement is executed.

(2) The execution of a UNDELETE statement does not affect the content of the record area.

(3) The execution of the UNDELETE statement causes the value of the I-O status associated with file-name to be updated.

(4) Transfer of control following the successful or unsuccessful execution of the UNDELETE operation depends on the presence or absence of the optional INVALID KEY and NOT INVALID KEY phrases in the UNDELETE statement.

For indexed files:

(5) For an indexed file, the file system logically restores to the file that record identified by the content of the RECORD KEY data-item associated with the file-name. If the file does not contain the logically deleted record specified by the key, the invalid key condition exists.

(6) After the successful execution of an UNDELETE LOGICAL GLOBAL the data record identified has been logically restored to the file.

(7) If LOCAL is specified, it is ignored. If LOCAL GLOBAL is specified, it is treated as GLOBAL.

(8) If no type of restoration is specified, LOGICAL GLOBAL is assumed.

(9) The file position indicator is not affected by the execution of a UNDELETE statement.

For INFOS files:

(10) If the relative option and the KEY series phrase are omitted, the default is STATIC.

(11) The occurrence number is used.

(12) FEEDBACK is not used and is not updated.

(13) KEY LENGTH is used.

(14) The record to UNDELETE is determined according to what is specified in the relative option phrase and/or the KEY series phrase. The specification can be implicit if the program uses the defaults or explicit if the KEY or path is fully specified.

(15) FIX POSITION causes the record pointer to move from the current position to the position specified in this statement. RETAIN position causes the record position to remain at the position it was on before the execution of this statement. RETAIN is the default.

(16) The relative motion option without the KEY series phrase allows access to the index file relative to that file's current record position.

(17) Using the KEY series phrase without the relative motion option cause the key path specified to begin with the top index in the hierarchy and follow a downward motion.

(18) If the KEY series phrase is specified, each key, identifier-2, must be declared in the SELECT statement for file-name. If the relative motion option and KEY series phrase at both specified only UP, DOWN, and STATIC are allowed. The relative motion option is processed first and the key path is used. If both are omitted, STATIC is the default.

(19) If LOGICAL LOCAL is specified, the key and partial record are logically restored.

(20) If LOGICAL GLOBAL is specified, the data record is logically restored.

(21) If LOGICAL LOCAL GLOBAL is specified, the key, partial record, and the data record are logically restored.

(22) If no type of restoration is specified, LOGICAL LOCAL GLOBAL is the default.

E.58. UNLOCK

E.58.1 Function

The UNLOCK statement unlocks all records that have been locked by the program on a specified file connector. UNLOCK is an extension to ANSI COBOL.

E.58.2 General Format

UNLOCK *file-name* { RECORD / RECORDS }

E.58.3 Syntax Rules

(1) *File-name* must be the name of a file with random or dynamic access.

E.58.4 General Rules

(1) The file referenced by *file-name* must be a mass storage file and must be open in the input or I-O mode at the time of the execution of this statement.

(2) After the successful execution of an UNLOCK statement, all records that were locked in the file specified by *file-name* by this program are now released.

(3) The execution of a UNLOCK statement does not affect the content of the record area.

(4) The file position indicator is not affected by the execution of an UNLOCK statement.

(5) If no records are locked no error is detected and the UNLOCK is successful.

(6) The execution of the UNLOCK statement causes the value of the I-O status associated with *file-name* to be updated.

E.59. UNSTRING

E.59.1 Function

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

E.59.2 General Format

```
UNSTRING identifier-1 [ DELIMITED BY [ ALL ] { identifier-2 } { identifier-3 }
                                             { literal-1   } { literal-2   } ]... ]
    INTO { identifier-4
         [ DELIMITER IN identifier-5 ]
         [ COUNT IN identifier-6 ] }...
    [ WITH POINTER identifier-7 ]
    [ TALLYING IN identifier-8 ]
    [ ON OVERFLOW imperative-statement-1 ]
    [ NOT ON OVERFLOW imperative-statement-2 ]
    [ END-UNSTRING ]
```

E.59.3 Syntax Rules

(1) *Literal-1* and *literal-2* must be nonnumeric literals and neither can be a figurative constant that begins with the word ALL.

(2) *Identifier-1*, *identifier-2*, *identifier-3*, and *identifier-5* must reference data items described, implicitly or explicitly, as category alphanumeric.

(3) *Identifier-4* may be described as either the category alphabetic, alphanumeric, or numeric (except that the symbol `P' may not be used in the PICTURE character-string), and must be described implicitly or explicitly, as USAGE IS DISPLAY.

(4) *Identifier-6* and *identifier-8* must reference integer data items (except that the symbol `P' may not be used in the PICTURE character-string).

(5) *Identifier-7* must be described as an elementary numeric integer data item of sufficient size to contain a value equal to 1 plus the size of the data item referenced by *identifier-1*. The symbol `P' may not be used in the PICTURE character-string of *identifier-7*.

(6) The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

(7) *Identifier-1* must not be reference modified.

E.59.4 General Rules

(1) All references to *identifier-2* and *literal-1* apply equally to *identifier-3* and *literal-2*, respectively, and all recursions thereof.

(2) The data item referenced by *identifier-1* represents the sending area.

(3) The data item referenced by *identifier-4* represents the data receiving area. The data item referenced by *identifier-5* represents the receiving area for delimiters.

(4) *Literal-1* or the data item referenced by *identifier-2* specifies a delimiter.

(5) The data item referenced by *identifier-6* represents the count of the number of characters within the data item referenced by *identifier-1* isolated by the delimiters for the move to the data item referenced by *identifier-4*. This value does not include a count of the delimiter character(s).

(6) The data item referenced by *identifier-7* contains a value that indicates a relative character position within the area referenced by *identifier-1*.

(7) The data item referenced by *identifier-8* is a counter which is incremented by 1 for each occurrence of the data item referenced by *identifier-4* accessed during the UNSTRING operation.

(8) When a figurative constant is used as the delimiter, it stands for a single character nonnumeric literal.

When the ALL phrase is specified, one occurrence or two or more contiguous occurrences of *literal-1* (figurative constant or not) or the content of the data item referenced by *identifier-2* are treated as if they were only one occurrence, and one occurrence of *literal-1* or the data item referenced by *identifier-2* is moved to the receiving data item according to the rules in General Rule 13d.

(9) When any examination encounters two contiguous delimiters, the current receiving area is space filled if it is described as alphabetic or alphanumeric, or zero filled if it is described as numeric.

(10) *Literal-1* or the content of the data item referenced by *identifier-2* can contain any character in the computer's character set.

(11) Each *literal-1* or the data item referenced by *identifier-2* represents one delimiter. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item, and in the order given, to be recognized as a delimiter.

(12) When two or more delimiters are specified in the DELIMITED BY phrase, an OR condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending field can be considered a part of more than one delimiter.

Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

(13) When the UNSTRING statement is initiated, the current receiving area is the data item referenced by *identifier-4*. Data is transferred from the data item referenced by *identifier-1* to the data item referenced by *identifier-4* according to the following rules:

a. If the POINTER phrase is specified, the string of characters referenced by *identifier-1* is examined beginning with the relative character position indicated by the content of the data item referenced by *identifier-7*. If the POINTER phrase is not specified, the string of characters is examined beginning with the left-most character position.

b. If the DELIMITED BY phrase is specified, the examination proceeds left to right until either a delimiter specified by *literal-1* or the value of the data item referenced by *identifier-2* is encountered. (See General Rule 11.) If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area. However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

If the end of the data item referenced by *identifier-1* is encountered before the delimiting condition is met, the examination terminates with the last character examined.

c. The characters thus examined (excluding the delimiting character(s), if any) are treated as an elementary alphanumeric data item, and are moved into the current receiving area according to the rules for the MOVE statement.

d. If the DELIMITER IN phrase is specified the delimiting character(s) are treated as an elementary alphanumeric data item and are moved into the data item referenced by *identifier-5* according to the rules for the

MOVE statement. If the delimiting condition is the end of the data item referenced by *identifier-1*, then the data item referenced by *identifier-5* is space filled.

e. If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter character(s), if any) is moved into the area referenced by *identifier-6* according to the rules for an elementary move.

f. If the DELIMITED BY phrase is specified the string of characters is further examined beginning with the first character to the right of the delimiter. If the DELIMITED BY phrase is not specified the string of characters is further examined beginning with the character to the right of the last character transferred.

g. After data is transferred to the data item referenced by *identifier-4*, the current receiving area is the data item referenced by the next recurrence of *identifier-4*. The behavior described in paragraphs 13b through 13f is repeated until either all the characters are exhausted in the data item referenced by *identifier-1*, or until there are no more receiving areas.

(14) The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

(15) The content of the data item referenced by *identifier-7* will be incremented by one for each character examined in the data item referenced by *identifier-1*. When the execution of an UNSTRING statement with a POINTER phrase is completed, the content of the data item referenced by *identifier-7* will contain a value equal to the initial value plus the number of characters examined in the data item referenced by *identifier-1*.

(16) When the execution of an UNSTRING statement with a TALLYING phrase is completed, the content of the data item referenced by *identifier-8* contains a value equal to its value at the beginning of the execution of the statement plus a value equal to the number of *identifier-4* receiving data items accessed during execution of the statement.

(17) Either of the following situations causes an overflow condition:

a. An UNSTRING is initiated, and the value in the data item referenced by *identifier-7* is less than 1 or greater than the size of the data item referenced by *identifier-1*.

b. If, during execution of an UNSTRING statement, all receiving areas have been acted upon, and the data item referenced by *identifier-1* contains characters that have not been examined.

(18) When an overflow condition exists, the UNSTRING operation is terminated, the NOT ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the UNSTRING statement or, if the ON OVERFLOW phrase is specified, to *imperative-statement-1*. If control is transferred to *imperative-statement-1*, execution continues according to the rules for each statement specified in *imperative-statement-1*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-1*, control is transferred to the end of the UNSTRING statement.

(19) If, at the time of execution of an UNSTRING statement, the conditions described in General Rule 17 are not encountered, after completion of the transfer of data according to the other general rules, the ON OVERFLOW phrase, if specified, is ignored and control is transferred to the end of the UNSTRING statement or, if the NOT ON OVERFLOW phrase is specified to *imperative-statement-2*. If control is transferred to *imperative-statement-2*, execution continues according to the rules for each statement specified in *imperative-statement-2*. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of *imperative-statement-2*, control is transferred to the end of the UNSTRING statement.

(20) If *identifier-1*, *identifier-2*, or *identifier-3*, occupies the same storage area as *identifier-4*, *identifier-5*, *identifier-6*, *identifier-7*, or *identifier-8*, or if *identifier-4*, *identifier-5*, or *identifier-6*, occupies the same storage area as *identifier-7* or *identifier-8*, or if *identifier-7* and *identifier-8* occupy the same storage area, the result of the execution of this statement is undefined, even if they are defined by the same data description entry.

(21) The END-UNSTRING phrase delimits the scope of the UNSTRING statement.

E.60. USE

E.60.1 Function

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.

E.60.2 General Format

E.60.3 Syntax Rules

(1) A USE statement, when present, must immediately follow a section header in the declaratives portion of the Procedure Division and must appear in a sentence by itself. The remainder of the section must consist of zero, one, or more procedural paragraphs that define the procedures to be used.

(2) The USE statement is never executed; it merely defines the conditions calling for the execution of the USE procedures.

(3) Appearance of *file-name* in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

(4) The words ERROR and EXCEPTION are synonymous and may be used interchangeably.

(5) The files implicitly or explicitly referenced in the USE statement need not all have the same organization or access.

(6) The INPUT, OUTPUT, I-O, or EXTEND phrases may each be specified only once in the declaratives portion of a given Procedure Division.

E.60.4 General Rules

(1) A declarative is invoked when any of the conditions described in the USE statement which prefaces the declarative occurs while the program is being executed.

(2) Within a declarative procedure, there must be no reference to any nondeclarative procedures.

(3) Procedure-names associated with a USE statement may be referenced in a different declarative section or in a nondeclarative procedure only with a PERFORM statement. (A GO TO statement may be used if the -G g compiler switch is used, but this is not recommended.)

(4) When *file-name* is specified explicitly, no other USE statement applies to *file-name*.

(5) The procedures associated with a USE statement are executed by the input-output control system after completion of the standard input-output exception routine upon the unsuccessful execution of an input-output operation unless an AT END or INVALID KEY phrase takes precedence. The rules concerning when the procedures are executed are as follows:

a. If *file-name* is specified, the associated procedure is executed when the condition described in the USE statement occurs.

b. If INPUT is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the input mode or in the process of being opened in the input mode, except those files referenced by *file-name* in another USE statement specifying the same condition.

c. If OUTPUT is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the output mode or in the process of being opened in the output mode, except those files referenced by *file-name* in another USE statement specifying the same condition.

d. If I-O is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the I-O mode or in the process of being opened in the I-O mode, except those files referenced by *file-name* in another USE statement specifying the same condition.

e. If EXTEND is specified, the associated procedure is executed when the condition described in the USE statement occurs for any file open in the extend mode or in the process of being opened in the extend mode, except those files referenced by *file-name* in another USE statement specifying the same condition.

(6) After execution of the USE procedure, control is transferred to the invoking routine in the input-output control system and the input-output control system returns control to the next executable statement following the input-output statement whose execution caused the exception.

(7) Within a USE procedure, there must not be the execution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.

E.60.5 Example

The following example illustrates how USE statements define the conditions under which a declarative procedure is to be executed. The first USE statement says to execute that declarative procedure for any I-O error encountered with the file PATIENT-FILE. The other four are for any file *except* PATIENT-FILE, because PATIENT-FILE is named in another USE statement.

```
DECLARATIVES.
PATIENT-ERROR SECTION. USE AFTER ERROR PROCEDURE ON PATIENT-FILE.
************************************************
*** Any I-O error on PATIENT-FILE.
************************************************
PROCESS-PATIENT-FILE-ERROR.
       IF PATIENT-FILE-STATUS = OPEN-ERROR
              MOVE PATIENT-FILE-STATUS TO FILE-ERROR-STATUS,
              MOVE "PATIENTFILE" TO FILE-ERROR-NAME,
              MOVE "PATIENT FILE MAINTENANCE" TO PROGRAM-NAME,
              DISPLAY FILE-ACCESS-ERROR-SCREEN,
              STOP RUN.

INPUT-ERROR-FILE SECTION.  USE AFTER ERROR PROCEDURE ON INPUT.
*****************************************************************************
*** Any INPUT error for any file except PATIENT-FILE.
*****************************************************************************
PROCESS-INPUT-ERROR.
       ACCEPT DECL-EXCEPT-CODE FROM EXCEPTION STATUS.
       PERFORM DISPLAY-ERROR-SCREEN.
       STOP RUN.

OUTPUT-ERROR-FILE SECTION.  USE AFTER ERROR PROCEDURE ON OUTPUT.
*****************************************************************************
*** Any OUTPUT error for any file except PATIENT-FILE.
*****************************************************************************
PROCESS-OUTPUT-ERROR.
       MOVE 5  TO SCR-LINE.
       MOVE 10 TO SCR-COL.
       MOVE 15 TO SCR-HEIGHT.
       MOVE 60 TO SCR-WIDTH.
       MOVE "DECLARE ERROR" TO SCR-LABEL.
       CALL "SD_NEW_WINDOW" USING SCR-LINE, SCR-COL, SCR-HEIGHT,
                                  SCR-WIDTH, SCR-LABEL.
       DISPLAY ERROR-SCREEN.
       ACCEPT CLRSCR.
       CALL "SD_REMOVE_WINDOW".

INPUT-OUTPUT-FILE SECTION. USE AFTER ERROR PROCEDURE ON I-O.
*****************************************************************************
*** Any OUTPUT error for any file except PATIENT-FILE.
*****************************************************************************
PROCESS-IO-ERROR.
       ACCEPT DECL-EXCEPT-CODE FROM EXCEPTION STATUS.
       PERFORM DISPLAY-ERROR-SCREEN.
       STOP RUN.

EXTEND-FILE SECTION. USE AFTER ERROR PROCEDURE ON EXTEND.
*****************************************************************************
*** Any OUTPUT error for any file except PATIENT-FILE.
*****************************************************************************
       ACCEPT DECL-EXCEPT-CODE FROM EXCEPTION STATUS.
       DISPLAY "Please inform the database manager"
       DISPLAY " of the following error:".
       DISPLAY "Status = " DECL-EXCEPT-CODE.
       STOP RUN.

END DECLARATIVES.
```

EXAMPLE 28. Using Declaratives

E.61. WRITE

E.61.1 Function

The WRITE statement releases a logical record for an output or input-output (in random or dynamic access mode) file. It can also be used for vertical positioning of lines within a logical page (on a sequential file). IMMEDIATE is an extension to ANSI COBOL.

E.61.2 General Format (**ANSI 74 and ANSI 85**)

For sequential files:

```
WRITE record-name-1 [ IMMEDIATE ] [ FROM identifier-1 ]
    [ { BEFORE } ADVANCING { { { identifier-2 } [ LINES ] } } ]
      { AFTER  }           { { { integer-1    } [ LINE  ] } }
                           {   PAGE                       }
    [ AT { END-OF-PAGE } imperative-statement-1 ]
         { EOP         }
    [ NOT AT { END-OF-PAGE } imperative-statement-2 ]
             { EOP         }
    [ END-WRITE ]
```

For relative and indexed files:

```
WRITE record-name-1 [ IMMEDIATE ] [ FROM identifier-1 ]
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
    [ END-WRITE ]
```

E.61.3 General Format (***VXCOBOL***)

For sequential files:

```
WRITE record-name-1 [ IMMEDIATE ] [ FROM { identifier-1 } ]
                                         { literal-1    }
    [ { BEFORE } ADVANCING { { { identifier-2 } [ LINES ] } } ]
      { AFTER  }           { { { integer-1    } [ LINE  ] } }
                           { { mnemonic-name              } }
                           {   PAGE                       }
    [ AT { END-OF-PAGE } imperative-statement-1 ]
         { EOP         }
    [ NOT AT { END-OF-PAGE } imperative-statement-2 ]
             { EOP         }
    [ END-WRITE ]
```

For relative files:

```
WRITE record-name-1 [ IMMEDIATE ] [ FROM { identifier-1 } ]
                                         { literal-1    }
    [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ]
```

[END-WRITE]

For indexed files:

WRITE *record-name-1* [IMMEDIATE] [FROM { *identifier-1* / *literal-1* }] [KEY IS *identifier-3*]

[INVALID KEY *imperative-statement-1*]
[NOT INVALID KEY *imperative-statement-2*]
[END-WRITE]

For INFOS files:

WRITE [INVERTED] *record-name-1* [IMMEDIATE] [{ FIX / RETAIN } POSITION] [UP / DOWN / STATIC]

[SUPPRESS [PARTIAL RECORD] [DATA RECORD]]
[LOCK / UNLOCK] [FROM { *identifier-1* / *literal-1* }] [{ KEY IS / KEYS ARE } { *identifier-3* }...]
[INVALID KEY *imperative-statement-1*]
[NOT INVALID KEY *imperative-statement-2*]
[END-WRITE]

E.61.4 Syntax Rules

(1) *Record-name-1* and *identifier-1* must not refer to the same storage area.

(2) *Record-name-1* is the name of a logical record in the File Section of the Data Division and may be qualified.

(3) *Identifier-2* must reference an integer data item.

(4) *Integer-1* must be positive or zero.

(5) The ADVANCING phrase may only be specified for files whose file control entry specifies ASSIGN TO PRINTER, PRINTER-1, or DISPLAY.

(6) The phrases ADVANCING PAGE and END-OF-PAGE must not both be specified in a single WRITE statement.

(7) If the END-OF-PAGE or the NOT END-OF-PAGE phrase is specified, the LINAGE clause must be specified in the file description entry for the associated file.

(8) The words END-OF-PAGE and EOP are equivalent.

For ***VXCOBOL***

(9) If *mnemonic-name* is specified, it must reference a line printer control channel as specified in the SPECIAL-NAMES paragraph of the Environment Division.

(10) For an indexed file, *identifier-3* must be the RECORD KEY specified in the SELECT for file-name. For an INFOS file, *identifier-3* must be a data-name specified as a RECORD KEY in the SELECT for file-name.

E.61.5 General Rules

(1) The file referenced by the file-name associated with *record-name-1* must be open in the output, I-O (for random or dynamic access), or extend mode at the time of the execution of this statement.

(2) The logical record released by the successful execution of the WRITE statement is no longer available in the record area unless the file-name associated with *record-name-1* is specified in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the SAME RECORD AREA clause as the associated output file, as well as the file associated with *record-name-1*.

(3) The result of the execution of a WRITE statement with the FROM phrase is equivalent to the execution of the following statements in the order specified:

a. The statement:

```
MOVE identifier-1 TO record-name-1
```

according to the rules specified for the MOVE statement.

b. The same WRITE statement without the FROM phrase.

(4) After the execution of the WRITE statement is complete, the information in the area referenced by *identifier-1* is available, even though the information in the area referenced by *record-name-1* is not available except as specified by the SAME RECORD AREA clause.

(5) The file position indicator is not affected by the execution of a WRITE statement, except for INFOS files with the FIX POSITION phrase.

(6) The execution of the WRITE statement causes the value of the I-O status of the file-name associated with *record-name-1* to be updated.

(7) The execution of the WRITE statement releases a logical record to the operating system.

(8) The number of character positions in the record referenced by *record-name-1* must not be larger than the largest or smaller than the smallest number of character positions allowed by the RECORD IS VARYING clause associated with the file-name associated with *record-name-1*. In either of these cases the execution of the WRITE statement is unsuccessful, the WRITE operation does not take place, the content of the record area is unaffected and the I-O status of the file associated with *record-name-1* is set to a value indicating the cause of the condition.

(9) If, during the execution of the WRITE statement with the NOT END-OF-PAGE or NOT INVALID KEY phrase, and the end-of-page or invalid key condition does not occur, control is transferred to *imperative-statement-2* at the appropriate time as follows:

a. If the execution of the WRITE statement is successful, after the record is written and after updating the I-O status of the file-name associate with *record-name-1*.

b. If the execution of the WRITE statement is unsuccessful, after updating the I-O status of the file-name associated with *record-name-1*, and after executing the procedure, if any, specified by a USE AFTER STANDARD EXCEPTION PROCEDURE statement applicable to the file-name associated with *file-name-1*.

(10) The IMMEDIATE option causes the WRITE to immediately flush the new information to disk. Normally this information could be held in internal buffers before being flushed to disk. This option increases file security at the expense of performance. The IMMEDIATE option is ignored for INFOS files.

(11) The END-WRITE phrase delimits the scope of the WRITE statement.

For sequential files:

(12) The successor relationship of a sequential file is established by the order of execution of WRITE statements when the file is created. The relationship does not change except when records are added to the end of a file.

(13) When a sequential file is open in the extend mode, the execution of the WRITE statement will add records to the end of the file as though the file were open in the output mode. If there are records in the file, the first record written after the execution of the OPEN statement with the EXTEND phrase is the successor of the last record in the file.

(14) For a sequential file, when an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists and the contents of the record area are unaffected. The following actions take place:

a. The value of the I-O status of the file-name associated with *record-name-1* is set to a value indicating a boundary violation. (See I-O Status, page 237.)

b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file-name associated with *record-name-1*, that declarative procedure will then be executed.

c. If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file-name associated with *record-name-1*, the run unit will be terminated with a "Fatal I/O Error".

(15) For a sequential file, both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing will be provided to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

a. If *integer-1* or the value of the data item referenced by *identifier-2* is positive, the representation of the printed page is advanced the number of lines equal to that value.

b. If the value of the data item referenced by *identifier-2* is negative, the results are undefined.

c. If *integer-1* or the value of the data item referenced by *identifier-2* is zero, no repositioning of the representation of the printed page is performed.

d. For ***VXCOBOL***, if *mnemonic-name* is specified the representation of the printed page is advanced according to the rules of the line printer control channel.

e. If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced according to rules a, b, and c above.

f. If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced according to rules a, b, and c above.

g. If PAGE is specified and the LINAGE clause is specified in the associated file description entry, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page as specified in the LINAGE clause.

h. If PAGE is specified and the LINAGE clause is not specified in the associated file description entry, the record is presented on the physical page before or after (depending on the phrase used) the device is repositioned to the next physical page. The repositioning to the next physical page is accomplished in accordance with an implementor-defined technique. If physical page has no meaning in conjunction with a specific device, advancing will be provided by the implementor to act as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

(16) If the logical end of the representation of the printed page is reached during the execution of a WRITE statement with the END-OF-PAGE phrase, *imperative-statement-1* specified in the END-OF-PAGE phrase is executed. The logical end is specified in the LINAGE clause associated with *record-name-1*.

(17) An end-of-page condition occurs when the execution of a given WRITE statement with the END-OF-PAGE phrase causes printing or spacing within the footing area of a page body. This occurs when the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by *integer-2* or the data item referenced by *data-name-2* of the LINKAGE clause if specified. In this case, the WRITE statement is executed and then *imperative-statement-1* in the END-OF-PAGE phrase is executed.

An automatic page overflow condition occurs when the execution of a given WRITE statement (with or without an END-OF-PAGE phrase) cannot be fully accommodated within the current page body.

This occurs when a WRITE statement, if executed, would cause the LINAGE-COUNTER to exceed the value specified by *integer-1* or the data item referenced by *data-name-1* of the LINAGE clause. In this case, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the first line that can be written on the next logical page as specified in the LINAGE clause. *Imperative-statement-1* in the END-OF-PAGE phrase, if specified, is executed after the record is written and the device has been repositioned.

A page overflow condition occurs when the execution of a given WRITE statement would cause LINAGE-COUNTER to simultaneously exceed the value of both *integer-2* or the data item referenced by *data-name-2* of the LINAGE clause and *integer-1* or the data item referenced by *data-name-1* of the LINAGE clause.

(18) The runtime treats all WRITE statements for ASSIGN TO PRINTER or ASSIGN TO DISPLAY files which are opened on the current console as if they are going to a DG terminal. See the DISPLAY statement for more information.

(19) WRITE's to data-sensitive files generate the following kind of advancing information assuming "data" represents the record to be written:

	ANSI 74		***ANSI 85***	
BEFORE ADVANCING	On Windows	On UNIX	On Windows	On UNIX
PAGE	data<cr><ff>	data<ff>	data<cr><ff>	data<ff>
0 LINES	data<cr><cr>	data<cr>	data	data
1 LINES	data<cr><lf>	data<lf>	data<cr><lf>	data<lf>
2 LINES	data<cr><lf><cr><lf>	data<lf><lf>	data<cr><lf><cr><lf>	data<lf><lf>
AFTER ADVANCING	On Windows	On UNIX	On Windows	On UNIX
PAGE	<cr><ff>data<cr><cr>	<ff>data<cr>	<cr><ff>data	<ff>data
0 LINES	<cr><cr>data<cr><cr>	<cr>data<cr>	data	data
1 LINES	<cr><lf>data<cr><cr>	<lf>data<cr>	<cr><lf>data	<lf>data
2 LINES	<cr><lf><cr><lf>data<cr><cr>	<lf><lf>data<cr>	<cr><lf><cr><lf>data	<lf><lf>data
NOTE: Newline <nl> and linefeed <lf> are equivalent				

TABLE 31. ***ANSI 74*** and ***ANSI 85*** ADVANCING Definitions.

	VXCOBOL	
BEFORE ADVANCING	On Windows	On UNIX
PAGE	data<cr><ff>	<cr>data<ff>
0 LINES	data	<cr>data
1 LINES	data<cr><lf>	<cr>data<lf>
2 LINES	data<cr><lf><cr><lf>	<cr>data<lf><lf>
AFTER ADVANCING	On Windows	On UNIX
PAGE	<cr><ff>data	<cr><ff>data
0 LINES	data	<cr>data
1 LINES	<cr><lf>data	<cr><lf>data
2 LINES	<cr><lf><cr><lf>data	<cr><lf><lf>data
NOTE: Newline <nl> and linefeed <lf> are equivalent		

TABLE 32. ***VXCOBOL*** ADVANCING Definitions.

(20) In addition ***VXCOBOL*** supports WRITE with a CHANNEL option. The CHANNEL option causes the following advancing information to be generated:

BEFORE ADVANCING	
CHANNEL	<cr>data<^R><channel-code>
AFTER ADVANCING	
CHANNEL	<cr><^R><channel-code>data

TABLE 33. ***VXCOBOL*** CHANNEL ADVANCING Definitions.

Channels are the values 1 through 12 generating ASCII codes of "@" through "K":

Channel	ASCII code	Channel	ASCII code	Channel	ASCII code
1	@	5	D	9	H
2	A	6	E	10	I
3	B	7	F	11	J
4	C	8	G	12	K

(21) When using timeouts, **ICOBOL** handles them in the following order for WRITE statements:

a. If a timeout was set on the OPEN with the extended open option for timeout, then it is used; otherwise,

b. The default timeout for this particular device class is used.

NOTE: Extended open options are discussed in the Developer's Guide section beginning on page 756.

For relative files:

(22) When a relative file is opened in the output mode, records may be placed into the file by one of the following:

a. If the access mode is sequential, the WRITE statement causes a record to be released to the file system. The first record has a relative record number of 1, and subsequent records released have relative record numbers of 2, 3, 4, ... If the RELATIVE KEY phrase is specified for the file-name associated with *record-name-1*, the relative record number of the record being released is moved into the relative key data item by the file system during execution of the WRITE statement according to the rules for the MOVE statement. (See The MOVE statement, page 393.)

b. If the access mode is random or dynamic prior to the execution of the WRITE statement the value of the relative key data item must be initialized by the program with the relative record number to be associated with the record in the record area. That record is then released to the file system by execution of the WRITE statement.

(23) When a relative file is opened in the I-O mode and the access mode is random or dynamic records are to be inserted in the associated file. The value of the relative key data item must be initialized by the program with the relative record number to be associated with the record in the record area. Execution of the WRITE statement then causes the content of the record area to be released to the file system.

(24) When a relative file is open in extend mode, records are inserted into the file. The first record released to the file system has a relative record number one greater than the highest relative record number existing on the file. Subsequent records released to the file system have consecutively higher relative record numbers. If the RELATIVE KEY phrase is specified for the file-name associated with *record-name-1*, the relative record number of the record being released is moved into the relative key data item during the execution of the WRITE statement according to the rules for the MOVE statement.

(25) The invalid key condition exists under the following circumstances:

a. When the access mode is random or dynamic and the relative key data item specifies a record which already exists in the file, or

b. When an attempt is made to write beyond the externally defined boundaries of the file, or

c. When the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file.

(26) When the invalid key condition is recognized, the execution of the WRITE statement is unsuccessful, the content of the record area is unaffected, and the I-O status of the file-name associated with *record-name-1* is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules for an invalid key condition.

For indexed files:

(27) Execution of a WRITE statement causes the content of the record area to be released. The file system utilizes the contents of the record keys in such a way that subsequent access of the record may be made based upon any of these specified record keys.

(28) The value of the primary record key must be unique within the records in the file.

(29) The data item or for a composite key, the data-items specified as the primary record key must be set by the program to the desired value prior to the execution of the WRITE statement.

(30) If the file is open in the sequential access mode, records must be released to the file system in ascending order of primary record key values according to the collating sequence of the file. When the file is open in the extend mode, the first record released to the file system must have a primary record key whose value is greater than the highest primary record key value existing in the file.

(31) If the file is open in the random or dynamic access mode, records may be released to the file system in any program-specified order.

(32) When the ALTERNATE RECORD KEY clause is specified in the file control entry for an indexed file, the value of the alternate record key may be non-unique only if the DUPLICATES phrase is specified for that data item. In this case the file system provides storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which they are released to the file system.

Notes:

1. The **ICOBOL** file system always allows duplicates when using ICISAM revision 5 and 6 files.

(33) The invalid key condition exists under the following circumstances:

a. When the file is open in the sequential access mode, and the file also is open in the output or extend mode, and the value of the primary record key is not greater than the value of the primary record key of the previous record, or

b. When the file is open in the output or I-O mode, and the value of the primary record key equals the value of the primary record key of a record already existing in the file, or

c. When the file is open in the output, extend, or I-O mode, and the value of an alternate record key for which duplicates are not allowed equals the value of the corresponding data item of a record already existing in the file, or

d. When an attempt is made to write beyond the externally defined boundaries of the file.

(34) When the invalid key condition is recognized, the execution of the WRITE statement is unsuccessful, the content of the record area is unaffected and the I-O status of the file-name associated with *record-name-1* is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules for an invalid key condition.

For INFOS files:

(35) If the relative option and the KEY series phrase are omitted, the default is the first key in the SELECT clause.

(36) The occurrence number is updated.

(37) FEEDBACK is used if you specify INVERTED. WRITE updates the FEEDBACK data item if INVERTED is not specified.

(38) KEY LENGTH is unused.

(39) If INVERTED is not specified, a record is written in a location that is determined according to what is specified in the relative option phrase and/or the KEY series phrase. The specification can be implicit if the program uses the defaults or explicit if the KEY or path is fully specified. If INVERTED is specified, an inversion of an existing record is written. Two keys will now point to the same data record. The FEEDBACK phrase must be specified in the FD to use INVERTED.

(40) FIX POSITION causes the record pointer to move from the current position to the position specified in this statement. RETAIN position causes the record position to remain at the position it was on before the execution of this statement. RETAIN is the default.

(41) The relative motion option without the KEY series phrase allows access to the index file relative to that file's current record position.

(42) Using the KEY series phrase without the relative motion option cause the key path specified to begin with the top index in the hierarchy and follow a downward motion.

(43) If the KEY series phrase is specified, each key, *identifier-1*, must be declared in the SELECT statement for file-name. The relative motion option is processed first and the key path is used. If both are omitted, STATIC is the default.

(44) If DUPLICATE and OCCURRENCE IS was specified in this file's SELECT clause, the occurrence number is updated for the last key in the key series phrases or the first key in the SELECT if there is no key series phrase. Zero indicates that no duplicate has occurred.

(45) If SUPPRESS DATA RECORD is specified, all locks on the data record are ignored and the data record associated with the index entry is not output. If FEEDBACK was defined, a zero is returned.

(46) If SUPPRESS PARTIAL RECORD is specified, the partial data record associated with the index entry is not output.

(47) If a FEEDBACK data item was declared for a file, it contains the location of the record that you just read, wrote, or rewrote. This location is used to link a key to an existing record in a WRITE INVERTED statement. If you intend to use the FEEDBACK data item for an inversion later in the program it must be saved in another location and restored to the FEEDBACK data item when needed.

(48) The IMMEDIATE option is ignored for INFOS files.

(49) If LOCK is specified, this program is the only one who can access the locked record until an UNLOCK of some form is done on that record. Closing the file automatically unlocks all locked records in the file.

VII. BUILTINS

A. Introduction

A.1. Overview

ICOBOL uses the CALL statement to access a set of builtins, all but four starting with the three-character sequence, "IC_". When the CALL statement is executed, the **ICOBOL** runtime looks first for a user-defined subroutine, then a builtin, and finally a callable COBOL program. All builtins, unless otherwise documented, return to the calling program with the appropriate Exception Status. User-defined subroutines or COBOL programs should not have names that begin with "IC_" as it could create a conflict with a builtin.

The following builtins are supported on Windows only:

IC_WINDOWS_SHELLEXECUTE

The following builtins are supported on Windows or on Unix with ThinClient (icrunrs) surrogates:

IC_WINDOW_TITLE
IC_WINDOWS_MSG_BOX
IC_WINDOWS_SETFONT
IC_WINDOWS_SHOW_CONSOLE

The following table lists all the **ICOBOL** builtins. Following the table is a description for each supported builtin. If the builtin provides user interaction through a menu, that interface is documented in the appropriate chapter in the Utilities Manual.

NOTE: The syntax descriptions of the various calls include type information. For numeric items, the traditional picture and USAGE information is provided, but it is not strictly enforced. You may use any numeric description in its place as long as it conforms to the tenant that of an integer was called for an integer is provided. (This does NOT apply to numeric values with packets (records). These must be coded as shown.) Also note that **ICOBOL2** required that these parameters match.

?CBADDR
?CBBADDR
?CBSYS
CLI
IC_ABORT_TERM
IC_CHANGE_DIR
IC_CHANGE_PRIV
IC_CHECK_DATA
IC_COMPRESS_OFF
IC_COMPRESS_ON
IC_CREATE_DIR
IC_CURRENT_DIR
IC_DECODE_URL
IC_DELAY
IC_DETACH_PROGRAM
IC_DIR_LIST
IC_DISABLE_HOTKEY
IC_DISABLE_INTS
IC_ENABLE_HOTKEY
IC_ENABLE_INTS
IC_EXTRACT_STRING
IC_FULL_DATE
IC_GET_DISK_SPACE
IC_GET_ENV
IC_GET_FILE_IND
IC_GET_KEY
IC_HANGUP
IC_HEX_TO_NUM
IC_INFOS_STATUS_TEXT
IC_INSERT_STRING
IC_KILL_TERM
IC_LOGON
IC_LOWER
IC_MOVE_FILE_DATA
IC_MOVE_STRING
IC_MSG_TEXT
IC_NUM_TO_HEX
IC_PID_EXISTS
IC_PRINT_STAT
IC_QUEUE_STATUS
IC_REMOVE_DIR
IC_RENAME
IC_RESOLVE_FILE
IC_SEND_MSG
IC_SEND_MAIL
IC_SERIAL_NUMBER
IC_SET_ENV
IC_SET_TIMEOUT
IC_SET_USERNAME
IC_SHUTDOWN
IC_SYS_INFO
IC_TERM_CTRL
IC_TERM_STAT
IC_TRIM
IC_UPPER
IC_VERSION
IC_WINDOW_TITLE
IC_WINDOWS_MSG_BOX
IC_WINDOWS_SETFONT
IC_WINDOWS_SHELLEXECUTE
IC_WINDOWS_SHOW_CONSOLE

TABLE 34. List of BUILTINS

B. Builtins

B.1. ?CBADDR (Added in 3.00)

The ?CBADDR function returns the address of any word-aligned COBOL data item or structure member. If the record or structure member is not word aligned, an error value of zero is returned as the word address.

The syntax is:

```
CALL "?CBADDR" USING id-1, id-2
```

Where

id-1
: is a record-level data-item whose word address you want to know. This item is not modified.

id-2
: is a data item of type PIC S9(9) USAGE IS COMP to which the address of the data item is returned.

This subroutine also works for structure members if the item is on a word boundary. If you request the word address of a byte-aligned item, id-2 is set to zero.

Level 01 and Level 77 data items always start on a word boundary, unless the -B compiler switch was specified.

B.2. ?CBBADDR (Added in 3.00)

The ?CBBADDR function returns the byte address of any COBOL data item.

The syntax is:

```
CALL "?CBBADDR" USING id-1, id-2
```

Where

id-1
: is a data-item whose byte address you want to know. This item is not modified.

id-2
: is a data item of type PIC S9(9) USAGE IS COMP to which the address of the data item is returned.

B.3. ?CBSYS (Added in 3.00)

The ?CBSYS function allows certain AOS/VS system calls to be executed. The COBOL file COBSYSID.IN included in the release shows the system calls that can be used.

The syntax is:

```
CALL "?CBSYS" USING id-1, id-2, id-3, id-4, id-5
```

Where

id-1
: is a data item of type PIC S9(4) USAGE IS COMP that holds one of the symbolic names from COBSYSID.IN for the system call that you wish to invoke. This item is not modified.

id-2
: is a data item of type PIC S9(9) USAGE IS COMP that holds both the value you pass to the system call in AC0 and the value returned.

id-3
: is a data item of type PIC S9(9) USAGE IS COMP that holds both the value you pass to the system call in AC1 and the value returned.

id-4
: is a data item of type PIC S9(9) USAGE IS COMP that holds both the value you pass to the system call in AC2 and the value returned.

id-5
: is a data item of type PIC S9(9) USAGE IS COMP to which the system returns the number of any system error that occurs during the system call. If none occurs, id5 contains zero.

Refer to the *AOS/VS, AOS/VSII, and AOS/RT32 System Call Dictionary* from Data General for a full discussion of system calls.

B.4. CLI (Added in 3.00)

The CLI function allows the COBOL program to call the Bourne shell (on UNIX) or the command processor defined by the COMSPEC.environment variable (on Windows).

The syntax is:

```
CALL "CLI" [ USING id-1 [,id-2] ]
```

Where

id-1

is defined as PIC X(n) and has the value of a particular Bourne shell (UNIX) or command processor (Windows) command that you wish to execute and return. For ***VXCOBOL***, n = 250. For **ANSI 74 and ANSI 85**, 0 < n < 256.

id-2

is defined as PIC X(n) and receives output from the command. The contents of *id-2* are either spaces or "*Error*" depending whether the shell returns a zero or non-zero exit code. For ***VXCOBOL***, n = 250. For **ANSI 74 and ANSI 85**, 6 < n < 256.

If no arguments are specified the shell is called in interactive mode. Only when the shell is exited (entering an exit command or Ctrl-D (UNIX)) will you return to the COBOL program.

On UNIX, if *id-1* is specified, it may contain a shell script, or one or more shell commands separated by semicolons. On Windows, if *id-1* is specified it may contain a .BAT file name or a command.

If the shell cannot be started the Exception Status is set and the ON EXCEPTION clause, if specified, is executed. If the shell exits with an error, the exception status is set, but the ON EXCEPTION clause is not taken, even if present.

B.5. IC_ABORT_TERM

The IC_ABORT_TERM builtin allows active terminals to be aborted either to facilitate a system shutdown or for other reasons.

The IC_ABORT_TERM builtin is enabled with the Abort terminal privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 221 "This operation is not permitted.".

On UNIX, the runtime requests that ICEXEC issue a UNIX SIGUSR1 to the process corresponding to the console number selected.

On Windows, the runtime passes the request to ICEXEC.

Two modes are available.

Mode 1 (Interactive Mode)

For mode 1, the syntax is:

```
CALL "IC_ABORT_TERM"
```

Upon invocation, a terminal status window of all logged-on terminals will be displayed. You are then prompted as to which terminal you wish to abort. Once that terminal is aborted you will see the confirmation in the status window. Aborting a terminal will not remove it from the terminal status window but will mark the terminal as `Stopped' in the terminal status window.

For more on IC_ABORT_TERM in mode 1 see the Abort Terminal utility in the Utilities Manual.

Mode 2 (Program Mode)

For mode 2, the syntax is:

```
CALL "IC_ABORT_TERM" USING term-number
```

Where

term-number
: is a PIC 9(4) COMP that holds the terminal number to abort.

If an invalid terminal number or a terminal that is not currently active is given, an Exception Status 228 "The terminal is not logged on" is returned. If the terminal is not enabled, Exception Status 229 "The terminal is not configured into the system" is returned.

B.6. IC_CHANGE_DIR

The IC_CHANGE_DIR builtin allows the program to change the working directory.

The syntax is:

```
CALL "IC_CHANGE_DIR" USING name
```

Where

name

is a PIC X(n) and holds the new working directory name.

Any error will result in a non-zero exception status and the ON EXCEPTION clause, if present, to be executed.

> Changing the working directory does not change any paths which have been resolved at initialization time. In particular, ICPCQDIR, ICCODEPATH, and ICDATAPATH are not resolved again.
>
> If this builtin is used, then the directory specifiers "." and " .." as well as an empty path entry (::), should not be included in these paths. Full pathnames should be used for all entries.

B.7. IC_CHANGE_PRIV

The IC_CHANGE_PRIV builtin allows a program to change the privileges associated with its own console or any other specified console. The changed privileges are only in effect while the runtime system process assigned to that console is active. The privileges revert to those configured for the line when the runtime system is started again.

The syntax is:

```
CALL "IC_CHANGE_PRIV" USING operation, privileges, [ term-id ]
```

Where

operation

is a PIC X(1) data item containing a code for the operation to be performed. The values are as follows:

Value	Operation
S	Set the privilege(s)
C	Clear the privilege(s)

privilege

is a PIC X(n) string (1 <= n <= 16) which contains the characters indicating which privileges are to be set or cleared. These characters are as follows:

Value	Privilege
A	Abort terminal privilege
C	Printer control management privilege
D	Program debugging privilege
I	System Information privilege
M	Message sending privilege
O	Detach/Host program privilege
P	Printer control privilege
S	System shutdown privilege
T	Terminal status privilege
W	Watch other terminals privilege
X	eXclude terminal from being watched

These characters correspond to those returned by ACCEPT id FROM ENVIRONMENT. More than one of these characters may be specified in the string. The B=Console interrupt privilege, cannot be changed with this call.

term-id

Is a PIC 9(4) COMP item containing the console number whose privileges are to be changed. This terminal must be logged on. If terminal number is omitted, then the program's own console number is used.

The privileges indicated in the privilege string are either added to or removed from those available to the specified console.

If the operation code or privilege string contains any character other than those specified above, an Exception Status 13 "Invalid Data", is returned. (The codes are case insensitive, and the privileges string may include SPACES).

Exception Status 228 "The terminal is not logged on", is returned when the term-id variable does not represent a currently active console.

Exception Status 221 "This operation is not permitted", is returned when:

(1) the watch privilege is to be set but the watch option is not present on the runtime license,
(2) Printer control privilege or Printer control management privilege is to be set and printer control is not enabled, or
(3) the Console Interrupt Privilege is given as an argument.

B.8. IC_CHECK_DATA

The IC_CHECK_DATA builtin allows data to be verified via CRC, LRC (XOR), or checksum.

The syntax is:

```
CALL "IC_CHECK_DATA" USING option, polynomial, length, buffer, result
```

Where

option
specifies a 1-byte binary, PIC 99 COMP, that holds the calculation option. Valid options are:

Value	Calculation Option
0	Normal CRC using the supplied polynomial
1	Reverse CRC using the supplied polynomial
2	LRC (XOR) 8-bit calculation
3	Checksum calculation
Note: Adding 64 to one of the above calculation options says to use the passed-in result value as the base to start the calculation, otherwise zero is used.	

polynomial
is a 2-byte binary, PIC 9(4) COMP, that holds the binary value for the CRC generator polynomial.

length
is a 2-byte binary, PIC 9(4) COMP, which holds the length of data in the buffer on which to perform the calculation. This cannot be larger than buffer.

buffer
is a PIC X(n) that holds the data on which the check is to be calculated.

result
is a 2-byte binary, PIC 9(4) COMP, that holds the calculated value as a binary number.

Some common crc polynomials are:

```
CRC-CCITT                       1021h or  4129 (base 10)
CRC-16                          8005h or 32773

reverse CRC-CCITT               8408h or 33800
reverse CRC-16                  A001h or 40961
```

The CRC-CCITT polynomial is used for XMODEM-CRC protocol.

For example, calculation option 64 would be used to calculate a CRC on a block (or file) that is larger than the buffer by making repeated calls.

B.9. IC_COMPRESS_OFF (Added in 3.30)

The IC_COMPRESS_OFF builtin causes screen compression to be disabled if the current screen allows compression and is currently in compressed mode.

The syntax is:

```
CALL "IC_COMPRESS_OFF"
```

Errors include "Invalid operation" if compressed mode support is not enabled and "Parameter mismatch" if any parameters are passed.

When compressed mode is switched from one mode to the other, the screen is completely erased. All information must be re-displayed by the program. The screen buffer is erased also.

B.10. IC_COMPRESS_ON (Added in 3.30)

The IC_COMPRESS_ON builtin causes screen compression to be enabled if the current screen allows compression and is currently not in compressed mode.

The syntax is:

```
CALL "IC_COMPRESS_ON"
```

Errors include "Invalid operation" if compressed mode support is not enabled and "Parameter mismatch" if any parameters are passed.

When compressed mode is switched from one mode to the other, the screen is completely erased. All information must be re-displayed by the program. The screen buffer is erased also.

B.11. IC_CREATE_DIR

The IC_CREATE_DIR builtin creates a directory.

The syntax is:

```
CALL "IC_CREATE_DIR" USING name
```

Where

name

is a PIC X(n) and holds the directory name to be created.

If the directory already exists, a File Exists (Exception Status 32) will be returned.

B.12. IC_CURRENT_DIR

The IC_CURRENT_DIR builtin allows the program to get the current working directory.

The syntax is:

```
CALL "IC_CURRENT_DIR" USING name
```

Where

name

is a PIC X(n) and holds the current working directory name.

Any error will result in a non-zero exception status and the ON EXCEPTION clause, if present, to be executed.

B.13. IC_DECODE_URL

The IC_DECODE_URL builtin decodes a URL-encoded string provided in one data item, placing the result in another data item. IC_DECODE_URL is the opposite of IC_ENCODE_URL.

The syntax is:

```
CALL "IC_DECODE_URL" USING source-string, destination-string
```

Where

source-string
: is a PIC X(n) data item that is to be decoded..

destination-string
: is a PIC X(n) data item into which the decoded characters are to be moved.

Any error will result in a non-zero exception status and the ON EXCEPTION clause, if present, to be executed.

Any %hex-format characters (unsafe or reserved characters) that are in the URL *source-string* are converted from their %hex-format and placed into the *destination-string* as standard ASCII. Both parameters should be PIC X(n). Processing of the *source-string* stops at the length of the string or on a LOW-VALUE. Generally the *destination-string* should be initialized to LOW-VALUES.

Reserved characters are:

Character name	URL code
;	%2B
/	%2F
?	%3F
:	%3A
@	%40
=	%3D
&	%26 (or &amp)

Unsafe characters:

Character name	URL code
(space)	%20
<	%3C
>	%3E
"	%22
#	%23
%	%25
[	%5B
]	%5D
{	%7B
}	%7D
|	%7C
\	%5C
^	%5E
~	%7E
`	%60

B.14. IC_DELAY

Suspends program execution for the given number of tenths of seconds. If no argument is specified, 30 (3 seconds) is used.

The syntax is:

```
CALL "IC_DELAY" [ USING delay-time ]
```

Where

delay-time

is a PIC 9(4) COMP between 0 and 65535. A 0 implies only that a resched be done.

B.15. IC_DETACH_PROGRAM

The IC_DETACH_PROGRAM builtin allows a COBOL program to be started on another logical console.

IC_DETACH_PROGRAM is enabled with the Detach/Host programs privilege in the Program Environment of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 221 "This operation is not permitted.".

The syntax is:

```
CALL "IC_DETACH_PROGRAM" USING program [, console [, file-name ]]
```

Where

program
: is a PIC X(n) string holding a valid COBOL program including switches to be executed.

console
: is a PIC 9(4) COMP and specifies the logical console on which to start the new program. 65535 says to use the next available console and return that value into console. Otherwise that logical console is used to start the program. If console is not given, then the next available console is used.

file-name
: is a PIC X(n) and specifies an output filename to which to send any output from this program. If not specified, all output is sent to the null device (discarded). File-name can use ICLINK.

An available detachable console is defined to be a logical console that is:

1) enabled,
2) whose device is set to NUL (on Windows) or null (on UNIX), and
3) is currently not running a detached program.

NOTES:
If a detached program is started with no optional output file, then all output from the program will go to the null device (discarded).

All detached programs will generate an end-of-file (EOF) error on any ACCEPT or READ from the console, as the input device will always be set to the null device.

A detached program can only execute non-screen DISPLAY statements. A screen DISPLAY will generate an error and the program will terminate.

Possible errors for IC_DETACH_PROGRAM include:

Error number	Meaning
1	Invalid operation
36	Filename is not valid (for an invalid program name)
209	Parameter mismatch (for no program name specified or if console is invalid, i.e., greater than 65535 or not a number)
212	No more programs are available (if no available consoles can be found to detach this program to)
219	Invalid task number (if the console specified by console is not avilable or is in use)
221	This operation is not permitted (if the calling program does not have the Detach/Host programs privilege).

The detached program will inherit the starting program's username. Its privileges are those specified for the console on which it is running. Detached programs cannot execute any builtins or system calls that perform screen I/O.

If a detached program terminates abnormally, any error will be written to the standard output file or to the starting program's standard error file on UNIX.

NOTE: A standard CALL PROGRAM error like Program Not Found, Program Too Big, etc. is not returned by an IC_DETACH_PROGRAM because it occurs after the "detached program" has been detached from the current program.

B.16. IC_DIR_LIST

The IC_DIR_LIST builtin allows directory information on a file or files to be retrieved.

The syntax is:

```
CALL "IC_DIR_LIST" USING lookup-file, entries [, output-file [,  rev ] ]
```

Where

lookup-file
: is a PIC X(n) and specifies the template (or filename) to look up.

entries
: is a PIC 9(4) COMP into which is returned the number of entries found.

output-file
: is a PIC X(n) and specifies the filename to which file-entry printer records are to be written. Output-file can use ICLINK.

rev
: is a PIC 9(2) COMP (one-byte binary), that specifies the revision of the file-entry for *output-file*.Valid values are 1 and 2 (default is 1). Rev 2 entry is 4-bytes larger (dates have 4-byte years).

Each file-entry record is like one of the following:

```
01 FILE-ENTRY-REV1.
   02 MODIFIED-INFO.
        03 DATE-MODIFIED       PIC 9(6).
        03 TIME-MODIFIED       PIC 9(8).
   02 ACCESSED-INFO.
        03 DATE-ACCESSED       PIC 9(6).
        03 TIME-ACCESSED       PIC 9(8).
   02 FILESIZE-BYTES           PIC 9(10).
   02 F-ATTRIBUTES             PIC X(8).
   02 F-ATTRIBUTE-RED  REDEFINES F-ATTRIBUTES.
        03 READABLE-ON         PIC X(1).
        03 WRITABLE-ON         PIC X(1).
        03 PROTECTABLE-ON      PIC X(1).
        03 ARCHIVE-IT          PIC X(1).
        03 DIRECTORY-TYPE      PIC X(1).
        03 SYSTEM-TYPE         PIC X(1).
        03 EXECUTABLE-TYPE     PIC X(1).
        03 FILLER              PIC X(1).
   02 FILENAME                 PIC X(64).

01 FILE-ENTRY-REV2.
   02 MODIFIED-INFO.
        03 DATE-MODIFIED       PIC 9(8).
        03 TIME-MODIFIED       PIC 9(8).
   02 ACCESSED-INFO.
        03 DATE-ACCESSED       PIC 9(8).
        03 TIME-ACCESSED       PIC 9(8).
   02 FILESIZE-BYTES           PIC 9(10).
   02 F-ATTRIBUTES             PIC X(8).
   02 F-ATTRIBUTE-RED  REDEFINES F-ATTRIBUTES.
        03 READABLE-ON         PIC X(1).
        03 WRITABLE-ON         PIC X(1).
        03 PROTECTABLE-ON      PIC X(1).
        03 ARCHIVE-IT          PIC X(1).
        03 DIRECTORY-TYPE      PIC X(1).
        03 SYSTEM-TYPE         PIC X(1).
        03 EXECUTABLE-TYPE     PIC X(1).
        03 FILLER              PIC X(1).
02 FILENAME                    PIC X(64).
```

Any error is stored into Exception Status and the ON EXCEPTION clause, if present, is executed. The EXCEPTION STATUS gives the error.

B.17. IC_DISABLE_HOTKEY

The IC_DISABLE_HOTKEY builtin completely or selectively disables hotkeys.

The syntax is:

```
CALL "IC_DISABLE_HOTKEY" [ USING argument-1 [, argument-2 ]...]
```

Where

argument-n

is a PIC 9(2) COMP item specifying a number from 0 to 99. This number identifies the hotkey that is to be disabled.

For example, if an argument contains 25, the program hotkey25.cx will not be executed as the result of pressing a key to which it has been assigned.

Multiple hotkeys may be disabled by specifying multiple arguments. ALL hotkeys may be disabled by calling IC_DISABLE_HOTKEY with no arguments. Hotkeys remain disabled until they are enabled by the IC_ENABLE_HOTKEY builtin or until the icrun process terminates.

B.18. IC_DISABLE_INTS

The IC_DISABLE_INTS builtin disables console interrupts for the current task. This can be done to protect critical sections of code from unexpected or premature exit.

The syntax is:

```
CALL "IC_DISABLE_INTS"
```

Console interrupts will be disabled for the console executing the call. Console interrupts will remain disabled until either a call is made to IC_ENABLE_INTS or the runtime system terminates.

B.19. IC_ENABLE_HOTKEY

The IC_ENABLE_HOTKEYS builtin allows hotkeys to be completely or selectively enabled.

The syntax is:

```
CALL "IC_ENABLE_HOTKEY" [ USING argument-1 [, argument-2 ]... ]
```

Where

argument-n

is a PIC 9(2) COMP item specifying a number from 0 to 99. This number identifies the hotkey that is to be enabled.

For example, if an argument contains 25, the program hotkey25.cx may be executed as the result of pressing a key to which it has been assigned.

Multiple hotkeys may be enabled by specifying multiple arguments. ALL hotkeys may be enabled by calling IC_ENABLE_HOTKEY with no arguments. Defined hotkeys are always enabled until they are disabled by the IC_DISABLE_HOTKEY builtin.

B.20. IC_ENABLE_INTS

The IC_ENABLE_INTS builtin allows console interrupts to be enabled under program control. This can be used along with IC_DISABLE_INTS to protect critical sections of code from unexpected or premature exit. Console interrupt privilege is required in order execute this builtin successfully.

The syntax is:

```
CALL "IC_ENABLE_INTS"
```

Console interrupts will be enabled for the console executing the call. Console interrupts will remain enabled until a call is made to IC_DISABLE_INTS. IC_ENABLE_INTS cannot be used to enable interrupts on consoles that do not initially support console interrupts as configured with the console interrupt privilege in the Program Environments section of the configuration file (.cfi).

Exception 221 "This operation is not permitted", is returned when the call is made but the current console does not have the console interrupt privilege.

B.21. IC_ENCODE_URL

The IC_ENCODE_URL builtin encodes an ASCII source string into a valid URL-encoded string converting reserved and unsafe characters into their %hex formats for the correct interpretation of the data by a browser. IC_ENCODE_URL acts the opposite of IC_DECODE_URL.

The syntax is:

```
CALL "IC_ENCODE_URL" USING source-string, destination-string
```

Where

source-string
is a PIC X(n) data item that is to be encoded..

destination-string
is a PIC X(n) data item into which the encoded characters are to be moved. In almost all cases this string will be larger than the source string.

Any error will result in a non-zero exception status and the ON EXCEPTION clause, if present, to be executed.

Any unsafe or reserved characters that are in the ASCII *source-string* are converted to their %hex-format and placed into the *destination-string*. Both parameters should be PIC X(n). Processing of the *source-string* stops at the length of the string or on a LOW-VALUE. Generally the *destination-string* should be initialized to LOW-VALUES.

Reserved characters are:

Character name	URL code
;	%2B
/	%2F
?	%3F
:	%3A
@	%40
=	%3D
&	%26 (or &amp)

Unsafe characters:

Character name	URL code
(space)	%20
<	%3C
>	%3E
"	%22
#	%23
%	%25
[	%5B
]	%5D
{	%7B
}	%7D
|	%7C
\	%5C
^	%5E
~	%7E
`	%60

B.22. IC_EXTRACT_STRING

The IC_EXTRACT_STRING builtin extracts a range of characters from a data-item and stores them into another one.

The syntax is:

```
CALL "IC_EXTRACT_STRING" USING source, src-pos, src-len, dest
```

Where

source
: is the data item from which the characters are to be extracted

src-pos
: is a two-byte PIC S9(4) COMP-5 item whose content identifies the leftmost character position of the string to be extracted. It must be in the range 1 to the length of source.

src-len
: is a two-byte PIC S9(4) COMP-5 item whose content identifies the number of characters to be extracted. It must be in the range 1 to length of source minus src-pos plus 1.

dest
: is the data-item into which the extracted characters are to be moved

The characters extracted from the source data item are treated as an alphanumeric item and moved to dest according to the rules for an alphanumeric to alphanumeric MOVE.

Exception Status 13 "Invalid Data" is returned if src-pos or src-len are not within the required ranges.

NOTE: For **ANSI 74 and ANSI 85**, use of this builtin is not recommended. A more efficient and standard manner to accomplish this task is to use reference modification:

```
MOVE source (src-pos:scr-len) TO dest.
```

B.23. IC_FULL_DATE

The IC_FULL_DATE builtin returns a full date and time string including century in a single call. It should be used to replace ACCEPT FROM DATE and ACCEPT FROM TIME statements, especially to support the year 2000.

The syntax is:

```
CALL "IC_FULL_DATE" USING date-string
```

Where

date-string

is a PIC X(20) into which the following data will be stored.

Data	Description
YYYY	year e.g., 1994
ddd	day of the year (1-366)
MM	month (1-12)
DD	day of the month (1-31)
HH	hour (0-23)
MM	minute (0-59)
SS	second (0-59)
hh	hundredths of seconds (0-99) or 0 if not supported
w	day of the week (1-7) where 1-Mon, ... 7-Sunday

The argument must be 20 characters or else an error is returned and no data is moved.

Using the IC_FULL_DATE builtin or the CURRENT_DATE intrinsic function are recommended methods to get date and time information as it returns a full, four-digit year, and, in addition, crossing midnight is not a problem as it is with two separate statements like ACCEPT FROM DATE and ACCEPT FROM TIME.

B.24. IC_GET_DISK_SPACE

The IC_GET_DISK_SPACE builtin allows total and free disk space to be determined.

The syntax is:

```
CALL "IC_GET_DISK_SPACE" USING location, free-space [, total-space
    [, disk-units ]]
```

Where

location
: is a PIC X(n) that holds the drive-name (Windows) or filesystem name (UNIX) upon which to get the needed data. Space implies the current drive or filesystem be used.

free-space
: is a PIC 9(9) COMP receiving the free space in bytes.

total-space
: is a PIC 9(9) COMP receiving the total space in bytes. This argument is optional.

disk-units
: is a PIC 9(9) COMP receiving the number of bytes per unit that the *free-space* and *total-space* were returned in. If not specified, 1 is used.

By using the disk-units parameter in combination with free-space and total-space, disks larger than 4GB can be described.

B.25. IC_GET_ENV

The IC_GET_ENV builtin allows an environment variable to be read.

The syntax is:

```
CALL "IC_GET_ENV" USING name-argument, return-argument
```

Where

name-argument
: is a PIC X(n) that holds the name of the environment variable to be read

return-argument
: is a PIC X(n) into which is returned the value of that argument according to the rules for MOVE.

If the environment variable cannot be found, an error is generated and the ON EXCEPTION clause, if present, is executed.

B.26. IC_GET_FILE_IND (Added in 3.23)

The IC_GET_FILE_IND builtin returns header information about a particular ICISAM indexed file. Thallows an environment variable to be read. This call uses ICDATAPATH, if needed, to find the file. This call only works for ICISAM indexed files.

The syntax is:

```
CALL "IC_GET_FILE_IND" USING file-name, file-info-struc
```

Where

File-name

is a PIC X(n) that holds the name of the file to lookup.

File-info-struct

is a structure with the following format:

```
 01 FILE-INFO-STRUCT.
    02 IND-STRUCT-REV        PIC 9(4) COMP.
    02 FILLER                PIC X(2).
 * 5, 6, or 7
    02 IND-VERSION           PIC 9(2) COMP.
 * 0x80-Purge attribute value is set
 * 0x40-Purge attribute value             (on/off)
 * 0x20-4GB big file attribute value is set
 * 0x10-4GB big file attribute value      (on/off)
    02 IND-ATTRIBUTES        PIC 9(2) COMP.
    02 IND-MIN-REC-SIZE      PIC 9(4) COMP.
    02 IND-MAX-REC-SIZE      PIC 9(4) COMP.
    02 IND-NUM-KEYS          PIC 9(4) COMP.
    02 IND-KEY-TABLE.
 * 32 bytes each
       10 IND-KEY-ENTRY        OCCURS 17 TIMES.
 * 0x80-duplicates (7.00),       0x40-case-insensitive (7.00)
 * 0x20-reverse ordered (7.10)   0x10-Do not invert null (7.10)
 * 0x08-Also keys
          15 IND-KEY-OPTS                          PIC 9(2) COMP.
          15 IND-KEY-BASE-SIZE                     PIC 9(2) COMP.
          15 IND-KEY-NULL-VALUE                    PIC 9(2) COMP.
          15 IND-KEY-COUNT                         PIC 9(2) COMP.
          15 IND-KEY-BASE-OFFSET                   PIC 9(4) COMP.
          15 FILLER                                PIC X(2).
          15 IND-ALSOS.
             17 IND-ALSO-OFFSET OCCURS 6 TIMES     PIC 9(4) COMP.
          15 IND-OCCURS REDEFINES IND-ALSOS.
             17 IND-OCCURS-SPAN                    PIC 9(4) COMP.
             17 IND-OCCURS-SUFFIX-COUNT            PIC 9(2) COMP.
             17 IND-OCCURS-SIZE   OCCURS 3 TIMES   PIC 9(2) COMP.
             17 IND-OCCURS-OFFSET OCCURS 3 TIMES   PIC 9(4) COMP.
    02 IND-RECORD-COUNT                            PIC 9(9) COMP.
  * Below is always 0 for rev 5
    02 IND-DELRECORD-COUNT                         PIC 9(9) COMP.
```

See the Indexed API for more info on this structure. INDEXED-STRUCT-REV returns 0x6931. (Decimal 26929).

The IND-DELRECORD-COUNT is set to 0xFFFFFFFF (4,294,967,295) when it cannot be determined. It is set this way also for files that use ICNETD.

B.27. IC_GET_KEY

The IC_GET_KEY builtin allows the program to get a keystroke from the keyboard in a terminal independent manner with a timeout. If no keystroke is received within the timeout period, an error is returned with the Exception Status 76 "Device Timeout". On an error, the KEY-TYPE and KEY-CODE are set to zero

The syntax is:

```
CALL "IC_GET_KEY" USING key-struc, time-out-value
```

Where

key-struc

specifies a structure defined like this:

```
01 KEY-STRUC.
    05 KEY-TYPE        PIC 99 COMP.
    05 KEY-CODE        PIC 99 COMP.
    05 KEY-CODE-X REDEFINES KEY-CODE PIC X.
```

KEY-TYPE is the type of character and will contain one of the values: 1, 2, 3, 4, 5, 7, or 8, as shown in the following table.

KEY-CODE returns the given code for the character, as specified in the current terminal definition. The table below shows KEY-CODE values that are possible for each KEY-TYPE value.

NOTES:
The key types and key codes correspond to those which are specified for the terminal description file (.tdi) which was configured using ICCONFIG (ICEDCFW).

Characters set to *Special Function (Illegal, Ignored, Refresh screen, Enter minus), Hot Keys* and Interrupt keys are not returned; instead, the defined action is carried out. For example, when an IC_GET_KEY call is executed if a hot key is pressed, the hot-key program is executed, and nothing is returned to the call.

No echoing of the keystroke is done to the screen except for those configured as *Special Function*.

KEY-TYPE value	KEY-TYPE description	Possible KEY-CODE values, for each KEY-TYPE value
1	normal character	The character's numeric value. NOTE: KEY-CODE-X can be displayed and is the character.
2	editing function	1=left a character 2= right a character 3=backspace 4=delete a character 5= insert mode on/off 6=clear field 7=clear to end of field 8= beginning of field 9=end of field 10=right a word 11= left a word 12=destructive TAB 13=left tab stop 14= right tab stop 15=sound bell 16=back delete
3	terminate field	Escape key code for terminate field key.
4	previous field	Escape key code for previous field key.
5	next field	Escape key code for next field key.
7	previous row	Escape key code for previous row key.
8	next row	Escape key code for next row key.

TABLE 35. IC_GET_KEY values returned

timeout-value

specifies a PIC 9(4) COMP containing the number of tenths of seconds to wait before terminating the READ. The values 0 through 63000 set a timeout in tenths of seconds, a 65535 is interpreted to wait forever, a 65534 says to default to the value specified as the global timeout (ICTIMEOUT), while a number between 63000 and 65534 will set the value to 63000. This value represents the time allowed between keystrokes before the system will timeout and terminate the operation. Setting a 0 essentially only reads the input buffer.

B.28. IC_HANGUP (Added in 3.00)

The IC_HANGUP terminates the runtime system. If the optional argument is specified the value is returned to the runtime system's parent process as an exit code.

The syntax is:

```
CALL "IC_HANGUP" [ USING exit-code ]
```

Where

exit-code

is a PIC 9(n). *Exit-code* may be zero, or any value between 10 and 255 inclusive. Values 1 thru 9 are reserved for runtime use.

Exception status 13 "Invalid data" is returned if the *exit-code* is out of range.

B.29. IC_HEX_TO_NUM (Added in 3.00)

The IC_HEX_TO_NUM builtin converts a hexadecimal string into a decimal number.

The syntax is:

```
CALL "IC_HEX_TO_NUM" USING hex-string, decimal-number
```

Where

hex-string
: specifies a PIC X(n) or A(n) where n is from 1 to 8 inclusive and contains valid hex digits.

decimal-number
: specifies a numeric type and the hex-string is converted to an integer and returned into *decimal-number*.

If the string cannot be converted or if the result will not fit, then an exception 13 (Invalid Data) is generated, and the ON EXCEPTION clause, if present, is executed.

B.30. IC_INFOS_STATUS_TEXT (*VXCOBOL*) (Added in 3.00)

The IC_INFOS_STATUS_TEXT builtin returns the text associated with an INFOS STATUS.

The syntax is:

```
CALL "IC_INFOS_STATUS_TEXT" USING infos-status, message-buffer
```

Where

infos-status
: specifies a PIC X(11) holding an INFOS STATUS as returned by ICRUN.

message-buffer
: specifies a PIC X(n) into which the message text is returned.

If the data in the INFOS STATUS item does not conform to the standards for INFOS status, an error is generated and the ON EXCEPTION clause, if present, is executed.

The message file infostat.ms holds the messages returned by IC_INFOS_STATUS_TEXT. The text file infostat.txt can be used to change the text of the messages as needed. See ICMAKEMS in the Utilities manual for more information on changing message text.

B.31. IC_INSERT_STRING

The IC_INSERT_STRING builtin allows a data item to be stored into a specified range of characters in another data item.

The syntax is:

```
CALL "IC_INSERT_STRING" USING source, dest, dest-pos, dest-len
```

Where

source
: is the data item to be stored

dest
: is the data-item into which all or part of the source is to be moved

dest-pos
: is a two-byte PIC S9(4) COMP-5 item whose content identifies the leftmost character position in the destination data item into which characters will be stored. It must be in the range 1 to the length of dest.

dest-len
: is a two-byte PIC S9(4) COMP-5 item whose content identifies the number of characters positions of dest into which characters will be stored. It must be in the range 1 to length of dest minus dest-pos plus 1.

The characters identified in the dest data item are treated as an alphanumeric item and source is moved to that item according to the rules for an alphanumeric to alphanumeric MOVE.

Exception Status 13 "Invalid Data" is returned if dest-pos or dest-len are not within the required ranges.

NOTE: For **ANSI 74 and ANSI 85**, use of this builtin is not recommended. A more efficient and standard manner to accomplish this task is to use reference modification:

```
MOVE source TO dest ( dest-pos:dest-len)
```

B.32. IC_KILL_TERM

The IC_KILL_TERM builtin allows **ICOBOL** runtime user tasks to be terminated. This builtin is similar to the IC_ABORT_TERM builtin except that it terminates the *process* rather than just aborting the executing program.

The IC_KILL_TERM builtin requires the Abort terminal privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 221 "This operation is not permitted.".

The syntax is:

```
CALL "IC_KILL_TERM" [ USING term-number ]
```

Where

term-number

is a PIC 9(4) COMP item that holds the terminal number of the process to be terminated.

If no terminal number is specified, a terminal status window of all logged-on terminals will be displayed. You are then prompted as to which terminal you want to terminate. Once terminated, the terminal will be removed from the status window.

On UNIX, IC_KILL_TERM requests ICEXEC to issue a UNIX signal of SIGTERM to the PID corresponding to the console number selected.

On Windows, the runtime passes the request to ICEXEC.

For more on IC_KILL_TERM with no terminal number, see the Kill Terminal utility in the Utilities Manual.

B.33. IC_LOGON (Added in 3.00)

The IC_LOGON builtin chains to logon. It runs the standard LOGON program and makes the terminal line Inactive in the terminal status window. IC_LOGON does not remove the terminal from the Terminal Status window. No ICISAM files should be open in the LOGON program since the system can and will abort users executing LOGON when entered via IC_LOGON or after the initial sign-on.

The syntax is:

```
CALL "IC_LOGON"
```

Any error will result in a non-zero exception status and the ON EXCEPTION clause, if present, to be executed.

> **NOTE**: A CALL PROGRAM "LOGON" is not the same as CALL “IC_LOGON”, since it will not mark the terminal as being Inactive.

B.34. IC_LOWER

The IC_LOWER builtin converts the specified string to all lower-case characters.

The syntax is:

```
CALL "IC_LOWER" USING string
```

Where

string

specifies a PIC X(n) that holds the data to be converted to lower-case.

NOTE: For **ANSI 74 and ANSI 85**, a more efficient way of accomplishing this task is by using the LOWER-CASE intrinsic function:

```
MOVE FUNCTION LOWER-CASE ( string ) TO string.
```

B.35. IC_MOVE_FILE_DATA

The IC_MOVE_FILE_DATA builtin provides the ability to quickly copy files from one place to another with various options.

The syntax is:

```
CALL "IC_MOVE_FILE_DATA" USING option, source, destination [, count
    [, start-src-pos [, start-dst-pos ]]]
```

Where

option

is a PIC 9(2) COMP and is composed of the following bit options:

Option-bit	Meaning
1	Don't erase destination if it exists
2	Write at eof (ignore start-dst-pos)
4	The destination file must exist
8	The destination file must NOT exist

Below are the useful combinations of the above option-bits.

Option	Destination file		Destination Position
	Does NOT exist	Exists	
0	create	erase	as specified
1	create	don't erase	as specified
3	create	don't erase	at eof
4	ERROR	erase	as specified
5	ERROR	don't erase	as specified
7	ERROR	don't erase	at eof
8	create	ERROR	as specified

source

is a PIC X(n) and holds the source filename to be copied ICLINK can be used.

destination

Is a PIC X(n) and holds the destination filename. It cannot be a directory. ICLINK can be used.

count

is a PIC 9(9) COMP and holds an optional count for how many bytes to copy from source or until EOF. If given, the number of bytes actually copied is returned.

start-src-pos

is a PIC 9(9) COMP and holds a optional byte offset in the source from which to start the copy. 0 is the beginning of file.

start-dst-pos

is a PIC 9(9) COMP and holds an optional byte offset in the destination to which copying should start.

The source file must exist and must be available to be opened for binary input.

This call allows a file to be copied upon itself with possible unintended results.

Source and destination are processed as an External Filename as described on page 751.

B.36. IC_MOVE_STRING

The IC_MOVE_STRING builtin allows a range of characters to be extracted from one data item and be stored into a specified range of characters in another data item.

The syntax is:

```
CALL "IC_MOVE_STRING" USING source, src-pos, src-len, dest, dest-pos,
    dest-len
```

Where

source
: is the data item from which the characters are to be extracted

src-pos
: is a two-byte PIC S9(4) COMP-5 item whose content identifies the leftmost character position of the string to be extracted. It must be in the range 1 to the length of source.

src-len
: is a two-byte PIC S9(4) COMP-5 item whose content identifies the number of characters to be extracted. It must be in the range 1 to length of source minus src-pos plus 1.

dest
: is the data-item into which all or part of the source is to be moved

dest-pos
: is a two-byte PIC S9(4) COMP-5 item whose content identifies the leftmost character position in the destination data item into which characters will be stored. It must be in the range 1 to the length of dest.

dest-len
: is a two-byte PIC S9(4) COMP-5 item whose content identifies the number of characters positions of dest into which characters will be stored. It must be in the range 1 to length of dest minus dest-pos plus 1.

The characters extracted from the source data item are treated as an alphanumeric item and moved to the characters identified in the dest data item according to the rules for an alphanumeric to alphanumeric MOVE.

Exception Status 13 "Invalid Data" is returned if src-pos, src-len, dest-pos, or dest-len are not within the required ranges.

NOTE: For **ANSI 74 and ANSI 85**, use of this builtin is not recommended. A more efficient and standard method to accomplish this task is to use reference modification:

```
MOVE source ( src-pos:src-len ) TO dest ( dest-pos:dest-len )
```

B.37. IC_MSG_TEXT

The IC_MSG_TEXT builtin allows system message text to be retrieved for a corresponding Exception Status.

The syntax is:

```
CALL "IC_MSG_TEXT" USING exc-code, return-argument
```

Where

exc-code
: is a PIC 9(5) that holds the numeric Exception Status for the message to be retrieved.

return-argument
: is a PIC X(n) (n should be at least 60) into which is returned the corresponding text.

B.38. IC_NUM_TO_HEX (Added in 3.00)

The IC_NUM_TO_HEX builtin converts a decimal number into a hexadecimal string.

The syntax is:

```
CALL "IC_NUM_TO_HEX" USING decimal-number, hex-string
```

Where

decimal-number

specifies a numeric type containing an integer whose value is greater than or equal to 0 and < (2**32 - 1)

hex-string

specifies a PIC X(n) or A(n). An n=8 will hold any possible numeric value. Decimal-number is converted into a hex string and placed in *hex-string*.

If the number cannot be converted (negative, fractional digits) or if the result will not fit, then an exception 13 (Invalid Data) is generated and the ON EXCEPTION clause, if present, is executed.

B.39. IC_PID_EXISTS (Added in 3.00)

IC_PID_EXISTS, checks for the existence of a specific pid.

The syntax is:

```
CALL "IC_PID_EXISTS" USING pid
```

Where

pid
is a numeric identifier containing the process identifier (PID) whose existence is to be verified

If the PID specified does not exist, exception status 219 (Invalid Task Number) is generated and the ON EXCEPTION clause, if present, is executed.

B.40. IC_PRINT_STAT

The IC_PRINT_STAT builtin calls the Printer Control Utility.

The IC_PRINT_STAT builtin requires the Printer Control privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call returns Exception Status 221 "This operation is not permitted".

The syntax is:

```
CALL "IC_PRINT_STAT" [ USING packet-1 [, packet-2 ]... ]
```

Where

packet-n

may be the name of any of the following packets.

No filtering:

```
01  PCQ-FILTER-NULL.
    03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE ZERO.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
```

Filter by range of PCQ numbers:

```
01  PCQ-FILTER-QUEUE.
    03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE 1.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
    03 QUEUE-PKT-MIN-PCQ      PIC 9(4) COMP.
    03 QUEUE-PKT-MAX-PCQ      PIC 9(4) COMP.
```

Filter by range file sizes:

```
01  PCQ-FILTER-SIZE.
    03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE 2.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
    03 SIZE-PKT-MIN-SIZE      PIC 9(8) COMP.
    03 SIZE-PKT-MAX-SIZE      PIC 9(8) COMP.
```

Filter by range of owners' console numbers (Windows) or user-ids (UNIX):

```
01  PCQ-FILTER-OWNER-ID.
    03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE 3.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
    03 OWNER-ID-PKT-MIN-ID    PIC 9(4) COMP.
    03 OWNER-ID-PKT-MAX-ID    PIC 9(4) COMP.
```

Filter by range of printed-by users' console number (Windows) or user-ids (UNIX):

```
01  PCQ-FILTER-PRINT-ID.
    03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE 4.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
    03 PRINT-ID-PKT-MIN-ID    PIC 9(4) COMP.
    03 PRINT-ID-PKT-MAX-ID    PIC 9(4) COMP.
```

Filter by owner's user name:

```
* Filter by owner's user name
01  PCQ-FILTER-OWNER-NAME.
    03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE 5.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
    03 OWNER-NAME-PKT-NAME    PIC X(16).
```

Filter by printed-by user name:

```
01 PCQ-FILTER-PRINT-NAME.
03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE 6.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
03 PRINT-NAME-PKT-NAME        PIC X(16).
```

Filter by simple filename:

```
01  PCQ-FILTER-FILE-NAME.
    03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE 7.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
    03 FILE-NAME-PKT-NAME     PIC X(64).

*** NOTE *** FILE-NAME-PKT-NAME can use ICLINK.
```

Filter by READ access to file:

```
01 PCQ-FILTER-READ-ACCESS.
03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE 8.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
```

Filter by status (mode): (values for status field are defined below)

```
* Filter by status (mode); values for status field are defined below
01  PCQ-FILTER-STATUS.
    03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE 9.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
    03 STATUS-PKT-STATUS      PIC 9(4) COMP.
    03 FILLER                 PIC 9(4) COMP VALUE ZERO.
```

Filter by directory name: (values for status field are defined below)

```
* Filter by directory name; values for status field are defined below
01  PCQ-FILTER-DIR-NAME.
    03 PCQ-FILTER-HEADER.
       05 PKT-ID              PIC 9(2) COMP VALUE 10.
       05 PKT-REV             PIC 9(2) COMP VALUE ZERO.
       05 FILLER              PIC 9(4) COMP VALUE ZERO.
    03 DIR-NAME-PKT-NAME      PIC X(64).
```

DIR-NAME-PKT-NAME can use ICLINK.

Values for STATUS-PKT-STATUS:

```
01  FILLER.
*  No status
    03 PCQ-STATUS-NULL               PIC 9(2) COMP VALUE ZERO.
*  Not yet printed
    03 PCQ-STATUS-NEW                PIC 9(2) COMP VALUE 1.
*  Already printed
    03 PCQ-STATUS-OLD                PIC 9(2) COMP VALUE 2.
*  Error has occurred
    03 PCQ-STATUS-ERROR              PIC 9(2) COMP VALUE 3.
*  Update in progress
    03 PCQ-STATUS-OPEN               PIC 9(2) COMP VALUE 4.
*  Queued to print
    03 PCQ-STATUS-QUEUE              PIC 9(2) COMP VALUE 5.
*  Holding in the print queue
    03 PCQ-STATUS-HOLD               PIC 9(2) COMP VALUE 6.
*  Printing
    03 PCQ-STATUS-PRINT              PIC 9(2) COMP VALUE 7.
*  Retrying
    03 PCQ-STATUS-RETRY              PIC 9(2) COMP VALUE 8.
*  Terminating
    03 PCQ-STATUS-TERM               PIC 9(2) COMP VALUE 9.
```

If no packets are specified, the Printer Control Utility is started and uses either no filtering or a default filter specified with the ICPCQFILTER environment variable.

If any packets are specified, then only those files which meet the specified criterion will be displayed. If more than one of these items is specified, then only files which meet ALL of the specified criteria will be displayed.

No more than 9 packets may be specified, and except for PCQ-FILTER-NULL, each packet may be specified at most one time.

Note that each packet has a common header format that varies only by the value of the packet identifier (and possibly rev at some later date).

NOTE: These packet definitions are shipped on the release media in the file pqfilter.ws found in the examples directory.

B.41. IC_QUEUE_LIST (Added in 3.01)

The IC_QUEUE_LIST builtin is used to retrieve the status of one of the printer control queues. The

The IC_QUEUE_LIST builtin allows a COBOL program to obtain a snapshot of the current contents of the printer control queue. The information that is provided is equivalent to that viewed with the interactive Printer Control Utility initiated by the IC_PRINT_STAT builtin. The information returned may optionally be filtered to reduce the number of entries returned to a particular set of interest.

The syntax is:

```
CALL "IC_QUEUE_LIST" USING packet [, filt-pkt-1 [, filt-pkt-2 ]... ]
```

Where

packet

is the following structure:

```
01 IC-QUEUE-LIST-PKT.
   05 IC-QUEUE-LIST-PKT-REV               PIC 9(2) COMP VALUE 1.
   05 IC-QUEUE-LIST-OS-TYPE               PIC 9(2) COMP.
      88 IC-QUEUE-LIST-OS-TYPE-WINDOWS VALUE 1.
      88 IC-QUEUE-LIST-OS-TYPE-UNIX    VALUE 2.
   05 IC-QUEUE-LIST-COUNT                 PIC 9(4) COMP.
   05 IC-QUEUE-LIST-ENTRY OCCURS 1 TO 1024 TIMES
                          DEPENDING ON IC-QUEUE-LIST-COUNT.
      10 IC-QUEUE-LIST-FILE-SIZE              PIC 9(9) COMP.
      10 IC-QUEUE-LIST-PRINT-ERR-CODE         PIC 9(9) COMP.
      10 IC-QUEUE-LIST-OWNER-USER-ID          PIC 9(9) COMP.
      10 IC-QUEUE-LIST-PRINT-USER-ID          PIC 9(9) COMP.
      10 IC-QUEUE-LIST-TIME-MODIFIED          PIC X(24).
      10 IC-QUEUE-LIST-TIME-PRINTED           PIC X(24).
      10 IC-QUEUE-LIST-OWNER-USER-NAME        PIC X(16).
      10 IC-QUEUE-LIST-PRINT-USER-NAME        PIC X(16).
      10 IC-QUEUE-LIST-FIRST-PAGE             PIC 9(9) COMP.
      10 IC-QUEUE-LIST-LAST-PAGE              PIC 9(9) COMP.
      10 IC-QUEUE-LIST-PCQ-UNIT               PIC 9(4) COMP.
      10 IC-QUEUE-LIST-PRIORITY               PIC 9(4) COMP.
      10 IC-QUEUE-LIST-COPIES                 PIC 9(4) COMP.
      10 IC-QUEUE-LIST-DISPOSITION            PIC 9(2) COMP.
      10 IC-QUEUE-LIST-OPTIONS.
         15 IC-QUEUE-LIST-OPTION-NO-BANNER        PIC X.
         15 IC-QUEUE-LIST-OPTION-NOTIFY           PIC X.
         15 IC-QUEUE-LIST-OPTION-START-FF         PIC X.
         15 IC-QUEUE-LIST-OPTION-COPIES-FF        PIC X.
         15 IC-QUEUE-LIST-OPTION-END-FF           PIC X.
         15 FILLER                                PIC X(11).
      10 IC-QUEUE-LIST-STATUS                 PIC 9(2) COMP.
      10 IC-QUEUE-LIST-OS-INFO                PIC X(24).
      10 IC-QUEUE-LIST-WINDOWS     REDEFINES IC-QUEUE-LIST-OS-INFO.
         15 IC-QUEUE-LIST-WIN-PRINT-COPY          PIC 9(4) COMP.
         15 IC-QUEUE-LIST-WIN-PRINT-PCNT          PIC 9(2) COMP.
         15 FILLER                                PIC X(21).
      10 IC-QUEUE-LIST-UNIX        REDEFINES IC-QUEUE-LIST-OS-INFO.
         15 IC-QUEUE-LIST-UNIX-REQ-ID-LEN         PIC 9(4) COMP.
         15 IC-QUEUE-LIST-UNIX-REQ-ID             PIC X(22).
      10 IC-QUEUE-LIST-QUEUE-SPOT                 PIC 9(4) COMP.
      10 IC-QUEUE-LIST-FILE-NAME-LEN          PIC 9(4) COMP.
      10 IC-QUEUE-LIST-SIMPLE-NAME-OFS        PIC 9(4) COMP.
      10 FILLER                               PIC X(2).
      10 IC-QUEUE-LIST-FILE-NAME              PIC X(256).
```

filt-pkt-n is a printer control filter packet.

(See the IC_PRINT_STAT builtin's documentation for a complete description.)

On a call to IC_QUEUE_LIST, the packet revision must be set to one. No other field needs to be set, and if set it will be overwritten. The fields are described below:

IC-QUEUE-LIST-PKT-REV

Revision of this packet structure -- it must always be set to 1.

IC-QUEUE-LIST-OS-TYPE

Operating system on which call is executed (returned by call). 1 is WINDOWS and 2 is UNIX.

IC-QUEUE-LIST-COUNT
Number of entries returned returned from call

IC-QUEUE-LIST-ENTRY
Array of queue entries. The maximum size may be lowered from 1024 to a smaller number, but MUST be a least as large as the configured number of printer control entries.

IC-QUEUE-LIST-FILE-SIZE
Size of the file to be printed.

IC-QUEUE-LIST-PRINT-ERR-CODE
Standard **ICOBOL** exception code. Corresponding text can be retrieved for it using the IC_MSG_TXT builtin function. This code is valid when IC-QUEUE-LIST-STATUS is IC-QUEUE-LIST-STATUS-ERROR or IC-QUEUE-LIST-STATUS-RETRY.

IC-QUEUE-LIST-OWNER-USER-ID
User id number (UNIX) or console number (WINDOWS) of the file's owner. If not yet known or assigned the field will contain NULL-USER-ID.

IC-QUEUE-LIST-PRINT-USER-ID
User id number (UNIX) or console number (WINDOWS) of the last user who printed the file. If not yet known or assigned the field will contain NULL-USER-ID.

IC-QUEUE-LIST-TIME-MODIFIED
The time the file was last modified returned as a string of the form mmm-dd-yyyy hh:mm:ss.hh

IC-QUEUE-LIST-TIME-PRINTED
The time the file was last printed returned as a string of the form mmm-dd-yyyy hh:mm:ss.hh

IC-QUEUE-LIST-OWNER-USER-NAME
User name of the file's owner

IC-QUEUE-LIST-PRINT-USER-NAME
User name of the last user who printed the file

IC-QUEUE-LIST-FIRST-PAGE
First page to print. (May be set for modify operation.) If this value is NULL-PAGE-NUMBER all pages will be printed.

IC-QUEUE-LIST-LAST-PAGE
Last page to print. (May be set for modify operation.) If this value is NULL-PAGE-NUMBER all from the specified first through the end of the file will be printed. If IC-QUEUE-LIST-FIRST-PAGE is NULL-PAGE-NUMBER, the IC-QUEUE-LIST-LAST-PAGE should be as well.

IC-QUEUE-LIST-PCQ-UNIT
PCQ unit on which the file is assigned to print. (May be set for modify operation.) It should always be less than the maximum configured PCQ.

IC-QUEUE-LIST-PRIORITY
Print job's priority. (May be set for modify operation.) Normally this is set to NULL-PRIORITY to allow the system to assign. Note that UNIX and Windows priorities differ (see below).

IC-QUEUE-LIST-COPIES
Number of copies to be printed. (May be set for modify operation)

IC-QUEUE-LIST-DISPOSITION
Disposition of the printer queue entry when the file has finished printing. (It may be set for modify operation.)

IC-QUEUE-LIST-OPTION-NO-BANNER
(UNIX only) Suppress banner page when printing? Field contains 'Y' or 'N' for yes or no respectively. (May be set for modify operation.)

IC-QUEUE-LIST-OPTION-NOTIFY
(UNIX only) Notify when printing is complete? Field contains 'Y' or 'N' for yes or no respectively. (May be set for modify operation.)

IC-QUEUE-LIST-OPTION-START-FF
(Windows only) Start printing with a form-feed? Field contains 'Y' or 'N' for yes or no respectively. (May be set for modify operation.)

IC-QUEUE-LIST-OPTION-COPIES-FF
(Windows only) End each copy with a form-feed? Field contains 'Y' or 'N' for yes or no respectively. (May be set for modify operation.)

IC-QUEUE-LIST-OPTION-END-FF
(Windows only) End printing with a form-feed? Field contains 'Y' or 'N' for yes or no respectively. (May be set for modify operation.)

IC-QUEUE-LIST-STATUS
: This is a code for the current status of the print job. (When any operation is performed, the status is cross-checked against the operation for validity.) Certain information in this packet is only valid when a print job has a particular status.

IC-QUEUE-LIST-WIN-PRINT-COPY
: (Windows only) Copy number currently being printed. This number is only valid when IC-QUEUE-LIST-STATUS is IC-QUEUE-LIST-STATUS-PRINT.

IC-QUEUE-LIST-WIN-PRINT-PCNT
: (Windows only) Percentage of job already printed. This number is only valid when IC-QUEUE-LIST-STATUS is IC-QUEUE-LIST-STATUS-PRINT.

IC-QUEUE-LIST-UNIX-REQ-ID-LEN
: (UNIX only) Length of the lp request id. This number is only valid when IC-QUEUE-LIST-STATUS is IC-QUEUE-LIST-STATUS-QUEUE, IC-QUEUE-LIST-STATUS-HOLD, IC-QUEUE-LIST-STATUS-PRINT or IC-QUEUE-LIST-STATUS-TERM.

IC-QUEUE-LIST-UNIX-REQ-ID
: (UNIX only) The lp request id. This field is only valid when IC-QUEUE-LIST-STATUS is IC-QUEUE-LIST-STATUS-QUEUE, IC-QUEUE-LIST-STATUS-HOLD, IC-QUEUE-LIST-STATUS-PRINT or IC-QUEUE-LIST-STATUS-TERM.

IC-QUEUE-LIST-QUEUE-SPOT
: Print job's current ordinal spot in the queue. This field is only valid when IC-QUEUE-LIST-STATUS is IC-QUEUE-LIST-STATUS-QUEUE or IC-QUEUE-LIST-STATUS-HOLD.

IC-QUEUE-LIST-FILE-NAME-LEN
: Length of the pathname found in IC-QUEUE-LIST-FILE-NAME.

IC-QUEUE-LIST-SIMPLE-NAME-OFs
: One-based offset to simple filename found in IC-QUEUE-LIST-FILE-NAME

IC-QUEUE-LIST-FILE-NAME
: Full pathname of the file to be printed

Other flag values used with the control are defined below:

* Flag values for unassigned entries

```
01 NULL-PAGE-NUMBER           PIC 9(9) COMP VALUE 4294967295.
01 NULL-USER-ID               PIC 9(9) COMP VALUE 4294967295.
01 NULL-PRIORITY              PIC 9(4) COMP VALUE 65535.
```

* Priority values

```
01 UNIX-HIGH-PRIORITY         PIC 9(4) COMP VALUE 0.
01 UNIX-LOW-PRIORITY          PIC 9(4) COMP VALUE 39.
01 UNIX-DEFAULT-PRIORITY      PIC 9(4) COMP VALUE 39.
01 WINDOWS-HIGH-PRIORITY      PIC 9(4) COMP VALUE 99.
01 WINDOWS-LOW-PRIORITY       PIC 9(4) COMP VALUE 1.
01 WINDOWS-DEFAULT-PRIORITY   PIC 9(4) COMP VALUE 1.
```

* Values for IC-QUEUE-LIST-STATUS

```
01 FILLER.
*  No status
   03 IC-QUEUE-LIST-STATUS-NULL     PIC 9(2) COMP VALUE ZERO.
*  Not yet printed
   03 IC-QUEUE-LIST-STATUS-NEW      PIC 9(2) COMP VALUE 1.
*  Already printed
   03 IC-QUEUE-LIST-STATUS-OLD      PIC 9(2) COMP VALUE 2.
*  Error has occurred
   03 IC-QUEUE-LIST-STATUS-ERROR    PIC 9(2) COMP VALUE 3.
*  Update in progress
   03 IC-QUEUE-LIST-STATUS-OPEN     PIC 9(2) COMP VALUE 4.
*  Queued to print
   03 IC-QUEUE-LIST-STATUS-QUEUE    PIC 9(2) COMP VALUE 5.
*  Holding in the print queue
   03 IC-QUEUE-LIST-STATUS-HOLD     PIC 9(2) COMP VALUE 6.
*  Printing
   03 IC-QUEUE-LIST-STATUS-PRINT    PIC 9(2) COMP VALUE 7.
*  Retrying due to error
   03 IC-QUEUE-LIST-STATUS-RETRY    PIC 9(2) COMP VALUE 8.
*  Terminating
   03 IC-QUEUE-LIST-STATUS-TERM     PIC 9(2) COMP VALUE 9.
```

* Values for IC-QUEUE-LIST-DISPOSITION

```
       01 FILLER.
       *  Keep file (don't delete or remove)
          03 IC-QUEUE-LIST-KEEP-DISPOSITION      PIC 9(2) COMP VALUE ZERO.
       *  Remove file from PASS after printing
          03 IC-QUEUE-LIST-REMOVE-DISPOSITION    PIC 9(2) COMP VALUE 1.
       *  Delete file (& remove) after printing
          03 IC-QUEUE-LIST-DELETE-DISPOSITION    PIC 9(2) COMP VALUE 2.
```

If no filter packets are specified, then the information returned without filtering or filtered by a default filter specified with the ICPQFILTER environment variable.

If any filter packets are specified, the only entries for those files which meet the specified criteria will be returned. If more than one filter packet is specified, then only entries for files which meet ALL of the specified criteria will be displayed. No more than 9 filter packets may be specified, and except for PCQ-FILTER-NULL, each packet may be specified at most one time.

The exception status codes which may be returned include:

13 Invalid data	One or more of the arguments contain invalid data
36 File name is not valid	Simple filename cannot be isolated
203 Program was not found	Printer control must be enabled
209 Parameter mismatch on call	Number, size or type of arguments is invalid
220 No more entries in the table	There are no applicable entries
221 Operation is not permitted	The call requires the Printer Control privilege in the program environments section of the configuration file (.cfi)
378 Data area passed to a system call is too small	The size of the entry array has call is too small been changed so that there is insufficient room to return all the entries in the .pq file
476 Filename too long	The length of the filename exceeds the size of the packet. (Can occur **on UNIX** with pathnames > 255 characters.)

NOTE: The filter packet definitions are shipped on the **ICOBOL** release media in the file pqfilter.ws found in the examples directory.

The main IC_QUEUE_LIST packet definitions are shipped in the **ICOBOL** release media in the file pq_list.ws found in the examples directory.

B.42. IC_QUEUE_OPERATION (Added in 3.01)

The IC_QUEUE_OPERATION builtin is used to retrieve the status of one of the printer control queues. The

The IC_QUEUE_OPERATION builtin allows a COBOL program to perform operations on items in the printer control queue. The operations that are available are equivalent to those available with the interactive Printer Control Utility initiated by the IC_PRINT_STAT builtin, with the addition of a PCQ info operation and get default filter operation.

The syntax is:

```
CALL "IC_QUEUE_OPERATION" USING operation, packet
```

Where

operation

is a numeric value which indicates what operation to perform

1 = Cancel a job	6 = Remove a job
2 = Delete a job	7 = Terminate a job
3 = Hold a job	8 = Unhold a job
4 = Modify job's options	9 = Get PCQ info
5 = Print a job	10 = Get default PCQ filter

packet

For operations 1-8, packet is an IC-QUEUE-LIST-ENTRY from the IC_QUEUE_LIST builtin's main packet. See the IC_QUEUE_LIST documentation for full details. It has the following structure:

```
01 IC-QUEUE-LIST-ENTRY.
    10 IC-QUEUE-LIST-FILE-SIZE               PIC 9(9) COMP.
    10 IC-QUEUE-LIST-PRINT-ERR-CODE          PIC 9(9) COMP.
    10 IC-QUEUE-LIST-OWNER-USER-ID           PIC 9(9) COMP.
    10 IC-QUEUE-LIST-PRINT-USER-ID           PIC 9(9) COMP.
    10 IC-QUEUE-LIST-TIME-MODIFIED           PIC X(24).
    10 IC-QUEUE-LIST-TIME-PRINTED            PIC X(24).
    10 IC-QUEUE-LIST-OWNER-USER-NAME         PIC X(16).
    10 IC-QUEUE-LIST-PRINT-USER-NAME         PIC X(16).
    10 IC-QUEUE-LIST-FIRST-PAGE              PIC 9(9) COMP.
    10 IC-QUEUE-LIST-LAST-PAGE               PIC 9(9) COMP.
    10 IC-QUEUE-LIST-PCQ-UNIT                PIC 9(4) COMP.
    10 IC-QUEUE-LIST-PRIORITY                PIC 9(4) COMP.
    10 IC-QUEUE-LIST-COPIES                  PIC 9(4) COMP.
    10 IC-QUEUE-LIST-DISPOSITION             PIC 9(2) COMP.
    10 IC-QUEUE-LIST-OPTIONS.
        15 IC-QUEUE-LIST-OPTION-NO-BANNER        PIC X.
        15 IC-QUEUE-LIST-OPTION-NOTIFY           PIC X.
        15 IC-QUEUE-LIST-OPTION-START-FF         PIC X.
        15 IC-QUEUE-LIST-OPTION-COPIES-FF        PIC X.
        15 IC-QUEUE-LIST-OPTION-END-FF           PIC X.
        15 FILLER                                PIC X(11).
    10 IC-QUEUE-LIST-STATUS                  PIC 9(2) COMP.
    10 IC-QUEUE-LIST-OS-INFO                 PIC X(24).
    10 IC-QUEUE-LIST-WINDOWS      REDEFINES IC-QUEUE-LIST-OS-INFO.
        15 IC-QUEUE-LIST-WIN-PRINT-COPY          PIC 9(4) COMP.
        15 IC-QUEUE-LIST-WIN-PRINT-PCNT          PIC 9(2) COMP.
        15 FILLER                                PIC X(21).
    10 IC-QUEUE-LIST-UNIX         REDEFINES IC-QUEUE-LIST-OS-INFO.
        15 IC-QUEUE-LIST-UNIX-REQ-ID-LEN         PIC 9(4) COMP.
        15 IC-QUEUE-LIST-UNIX-REQ-ID             PIC X(22).
    10 IC-QUEUE-LIST-QUEUE-SPOT              PIC 9(4) COMP.
    10 IC-QUEUE-LIST-FILE-NAME-LEN           PIC 9(4) COMP.
    10 IC-QUEUE-LIST-SIMPLE-NAME-OFS         PIC 9(4) COMP.
    10 FILLER                                PIC X(2).
    10 IC-QUEUE-LIST-FILE-NAME               PIC X(256).
```

For operation 9 (Get PCQ Info), packet is an IC-QUEUE-OP-INFO-PKT. It is defined as follows:

```
01 IC-QUEUE-OP-INFO-PKT.
    03 IC-QUEUE-OP-INFO-PKT-REV          PIC 9(2) COMP VALUE 1.
    03 FILLER                            PIC X.
    03 IC-QUEUE-OP-INFO-PCQ-COUNT        PIC 9(4) COMP.
    03 IC-QUEUE-OP-INFO-PCQ-MAX          PIC 9(4) COMP.
    03 IC-QUEUE-OP-INFO-JOB-COUNT        PIC 9(4) COMP.
```

For operation 10 (Get default PCQ filter), packet is an IC-QUEUE-OP-FILT-PKT. It is defined as follows:

```
01 IC-QUEUE-OP-FILT-PKT.
    05 IC-QUEUE-OP-FILT-PKT-REV          PIC 9(2) COMP VALUE 1.
    05 FILLER                            PIC X.
    05 IC-QUEUE-OP-FILT-OPTIONS.
        10 IC-QUEUE-OP-FILT-QUEUE            PIC X.
        10 IC-QUEUE-OP-FILT-SIZE             PIC X.
        10 IC-QUEUE-OP-FILT-OWNER-ID         PIC X.
        10 IC-QUEUE-OP-FILT-PRINT-ID         PIC X.
        10 IC-QUEUE-OP-FILT-OWNER            PIC X.
        10 IC-QUEUE-OP-FILT-PRINT            PIC X.
        10 IC-QUEUE-OP-FILT-NAME             PIC X.
        10 IC-QUEUE-OP-FILT-ACCESS           PIC X.
        10 IC-QUEUE-OP-FILT-STATUS           PIC X.
        10 IC-QUEUE-OP-FILT-DIR              PIC X.
        10 FILLER                            PIC X(6).
    05 IC-QUEUE-OP-FILT-STATUS-VAL       PIC 9(4) COMP.
    05 IC-QUEUE-OP-FILT-MIN-PCQ          PIC 9(4) COMP.
    05 IC-QUEUE-OP-FILT-MAX-PCQ          PIC 9(4) COMP.
    05 IC-QUEUE-OP-FILT-MIN-OWNER-ID     PIC 9(4) COMP.
    05 IC-QUEUE-OP-FILT-MAX-OWNER-ID     PIC 9(4) COMP.
    05 IC-QUEUE-OP-FILT-MIN-PRINT-ID     PIC 9(4) COMP.
    05 IC-QUEUE-OP-FILT-MAX-PRINT-ID     PIC 9(4) COMP.
    05 IC-QUEUE-OP-FILT-MIN-SIZE         PIC 9(8) COMP.
    05 IC-QUEUE-OP-FILT-MAX-SIZE         PIC 9(8) COMP.
    05 IC-QUEUE-OP-FILT-OWNER-NAME       PIC X(16).
    05 IC-QUEUE-OP-FILT-PRINT-NAME       PIC X(16).
    05 IC-QUEUE-OP-FILT-FILE-NAME        PIC X(64).
    05 IC-QUEUE-OP-FILT-DIR-NAME         PIC X(64).
```

Job operations

On a call to IC_QUEUE_OPERATION with any option except the PCQ info option (9), the list entry packet provided should be one that was filled in by a call to IC_QUEUE_LIST. All fields should be left exactly as returned from the call to IC_QUEUE_LIST. The only exception is that certain fields -- as noted below -- may be set with the new values requested on a modify operation.

The fields which set for a modify operation are IC-QUEUE-LIST-FIRST-PAGE, IC-QUEUE-LIST-LAST-PAGE, IC-QUEUE-LIST-PCQ-UNIT, IC-QUEUE-LIST-PRIORITY, IC-QUEUE-LIST-COPIES, IC-QUEUE-LIST-DISPOSITION.

On UNIX, IC-QUEUE-LIST-OPTION-NO-BANNER and IC-QUEUE-LIST-OPTION-NOTIFY may be set.

On Windows, IC-QUEUE-LIST-OPTION-START-FF, IC-QUEUE-LIST-OPTION-COPIES-FF, and IC-QUEUE-LIST-OPTION-END-FF may be set.

Once the operation is performed the provided entry is updated with its current values. The entry is set to LOW-VALUES after the REMOVE or DELETE operations are performed.

The incoming operation code is cross checked against IC-QUEUE-LIST-STATUS for validity and allowed or rejected according to the following table:

Status	Allowable operations
IC-QUEUE-LIST-STATUS-NEW	Delete, Modify, Print, Remove
IC-QUEUE-LIST-STATUS-OLD	Delete, Modify, Print, Remove
IC-QUEUE-LIST-STATUS-ERROR	Delete, Modify, Print, Remove
IC-QUEUE-LIST-STATUS-OPEN	-
IC-QUEUE-LIST-STATUS-QUEUE	Cancel, Hold
IC-QUEUE-LIST-STATUS-HOLD	Cancel, Unhold
IC-QUEUE-LIST-STATUS-PRINT	Terminate
IC-QUEUE-LIST-STATUS-RETRY	Terminate
IC-QUEUE-LIST-STATUS-TERM	-

PCQ Info Operation

On a call to IC_QUEUE_OPERATION with the PCQ info option (9), the info packet's revision must be set to one. No other field needs to be set, and if set it will be overwritten. The fields are described below:

IC-QUEUE-OP-INFO-PKT-REV
: Revision of this packet structure -- it must always be set to 1.

IC-QUEUE-OP-INFO-PCQ-COUNT
: This is the number of PCQ devices configured

IC-QUEUE-OP-INFO-PCQ-MAX-UNIT
: This is the number of the highest PCQ device (currently 127)

IC-QUEUE-OP-INFO-JOB-COUNT
: This is the configured maximum number of PCQ jobs

PCQ Default Info Operation

On a call to IC_QUEUE_OPERATION with the PCQ default filter option (10), the filter packet's revision must be set to one. No other field needs to be set, and if set it will be overwritten. The fields are described below:

IC-QUEUE-OP-FILT-PKT-REV
: Revision of this packet structure -- it must always be set to 1.

IC-QUEUE-OP-FILT-QUEUE
: If this field contains a 'Y', filtering by queue number is enabled. The two fields IC-QUEUE-OP-FILT-MIN-PCQ and IC-QUEUE-OP-FILT-MIN-PCQ are valid and contain the minimum and maximum queues being viewed.

IC-QUEUE-OP-FILT-SIZE
: If this field contains a 'Y', filtering by file size is enabled. The fields IC-QUEUE-OP-FILT-MIN-SIZE and IC-QUEUE-OP-FILT-MIN-SIZE are valid and contain the minimum and maximum file sizes being viewed.

IC-QUEUE-OP-FILT-OWNER-ID
: If this field contains a 'Y', filtering by owner id is enabled. (This is the console number **on Windows** and the user-id **on UNIX**.) The two fields IC-QUEUE-OP-FILT-MIN-OWNER-ID and IC-QUEUE-OP-FILT-MAX-OWNER-ID are valid and contain the minimum and maximum owner id value being viewed.

IC-QUEUE-OP-FILT-PRINT-ID
: If this field contains a 'Y', filtering by printed-by id is enabled. (This is the console number **on Windows** and the user-id **on UNIX**.) The two fields IC-QUEUE-OP-FILT-MIN-PRINT-ID and IC-QUEUE-OP-FILT-MAX-PRINT-ID are valid and contain the minimum and maximum printed-by id value being viewed.

IC-QUEUE-OP-FILT-OWNER
: If this field contains a 'Y', filtering by owner name is enabled. The field IC-QUEUE-OP-FILT-OWNER-NAME is valid and contains the owner name of the files being viewed.

IC-QUEUE-OP-FILT-PRINT
: If this field contains a 'Y', filtering by printed-by name is enabled. The field IC-QUEUE-OP-FILT-PRINT-NAME is valid and contains the printed-by name of the files being viewed.

IC-QUEUE-OP-FILT-NAME
: If this field contains a 'Y', filtering by file name is enabled. The field IC-QUEUE-OP-FILT-FILE-NAME is is valid and contains the simple filename of the files being viewed.

IC-QUEUE-OP-FILT-ACCESS
: If this field contains a 'Y', filtering by read access to the file is enabled.

IC-QUEUE-OP-FILT-STATUS
: If this field contains a 'Y', filtering by status is enabled. The field IC-QUEUE-OP-FILT-STATUS-VAL is valid and contains the status value of the files being viewed.

IC-QUEUE-OP-FILT-DIR
: If this field contains a 'Y', filtering by directory name is enabled. The field IC-QUEUE-OP-FILT-DIR-NAME is is valid and contains the directory name of the files being viewed.

IC-QUEUE-OP-FILT-MIN-PCQ

IC-QUEUE-OP-FILT-MAX-PCQ
: If IC-QUEUE-OP-FILT-QUEUE contains a 'Y', these two fields are valid and contain the minimum and maximum queues being viewed.

IC-QUEUE-OP-FILT-MIN-SIZE
IC-QUEUE-OP-FILT-MAX-SIZE
If IC-QUEUE-OP-FILT-QUEUE contains a 'Y', these fields are valid and contain the minimum and maximum file sizes being viewed.
IC-QUEUE-OP-FILT-MIN-OWNER-ID
IC-QUEUE-OP-FILT-MAX-OWNER-ID
If IC-QUEUE-OP-OWNER-ID contains a 'Y', these fields are valid and contain the minimum and maximum owner id value being viewed.
IC-QUEUE-OP-FILT-MIN-PRINT-ID
IC-QUEUE-OP-FILT-MAX-PRINT-ID
If IC-QUEUE-OP-PRINT-ID contains a 'Y', these fields are valid and contain the minimum and maximum printed-by id value being viewed.
IC-QUEUE-OP-FILT-STATUS
If this field contains a 'Y', filtering by status is enabled.
IC-QUEUE-OP-FILT-OWNER-NAME
If IC-QUEUE-OP-FILT-OWNER contains a 'Y', this field is valid and contains the owner name of the files being viewed.
IC-QUEUE-OP-FILT-PRINT-NAME
If IC-QUEUE-OP-FILT-PRINT contains a 'Y', this field is valid and contains the printed-by name of the files being viewed.
IC-QUEUE-OP-FILT-FILE-NAME
If IC-QUEUE-OP-FILT-NAME contains a 'Y', this field is valid and contains the simple filename of the files being viewed.
IC-QUEUE-OP-FILT-DIR-NAME
If IC-QUEUE-OP-FILT-DIR contains a 'Y', this field is valid and contains the directory name of the files being viewed.

Exception Status Codes

The exception status codes which may be returned include:

1 Invalid operation	An invalid operation code has been supplied or the operation is not valid with the status specified by the main packet.
2 File not found	A get default PCQ filter operation request was made when there is no default filter
13 Invalid data	One or more of the arguments contain invalid data
36 File name is not valid	Simple filename cannot be isolated
203 Program was not found	Printer control must be enabled
209 Parameter mismatch on call	Number, size or type of arguments is invalid
221 Operation is not permitted	The call requires the Printer Control privilege in the program environments section of the configuration file (.cfi)
476 Filename too long	The length of the filename exceed the size of the packet. (Can occur **on UNIX** with pathnames > 255 characters.)

NOTE: The main ic_queue_operation packet definitions are shipped in the **ICOBOL** release media in the file pq_list.ws found in the examples directory.

B.43. IC_QUEUE_STATUS (Added in 3.00)

The IC_QUEUE_STATUS builtin is used to retrieve the status of one of the printer control queues. The information returned is equivalent to that returned for the queue when TAB is pressed in the Printer Control Utility.

The syntax is:

```
CALL "ic_queue_status" USING queue-number, queue-status, queue-name
 [, queue-trans]
```

Where

queue-number
is an integer item containing the queue number for which the status is requested

queue-status
is a signed intger item into which a status code is returned, where the value is one of the following:

Status code	Meaning
1	Not available
2	Offline
3	Paused
4	Needs Attention
5	Retrying
6	Printing
7	Available

If the printer is the default printer, then the value returned will be negative. For example, if queue-status equals -7, then this is the default printer and it is available.

queue-name
is a PIC X(n) item into which the device name for the queue is returned

queue-trans
is an optional PIC X(n) item which, if present, receives the name of the printer translation

The following exception status codes may be returned:

Exception status code	Description
209	(Parameter mismatch on CALL) - Too many or few arguments, or incorrect type of argument
13	(Invalid data) - Queue number contains invalid data
2	(File not found) - Queue number specified not found

B.44. IC_REMOVE_DIR

The IC_REMOVE_DIR builtin allows a directory to be removed.

The syntax is:

```
CALL "IC_REMOVE_DIR" USING name
```

Where

name

is a PIC X(n) and holds the directory name to be removed.

Directory must be empty (except for . and .. files) to be removed. If the directory is not empty, a File Exists (Exception Status 32) will be returned. On UNIX, the current directory can be removed resulting in File Not Found errors when accessing any file based on the current directory. Generally, this should be avoided.

B.45. IC_RENAME

The IC_RENAME builtin allows a file to be renamed.

The syntax is:

```
CALL "IC_RENAME" USING old-filename, new-filename
```

Where

old-filename
: is a PIC X(n) that holds the old filename to be renamed.

new-filename
: is a PIC X(n) that holds the new filename.

Pathnames can be used. To rename an ICISAM file, you must rename each individual portion, explicitly supplying the .XD and .NX extensions with two builtin calls.

IC_RENAME does not go through the ICLINK link file facility.

Old-filename and new-filename are processed as an External Filename as described on page 751, except a full pathname is not made if only a simple name is given.

B.46. IC_RESOLVE_FILE

The IC_RESOLVE_FILE builtin resolves a filename to a full pathname using a search path. Templates are not allowed. When using ICDATAPATH this works just like an OPEN. When using ICCODEPATH this works just like a CALL or CALL PROGRAM.

The syntax is:

```
CALL "IC_RESOLVE_FILE" USING file-argument, lib-name [, search-path
   [, file-entry [, rev ]]]
```

Where

file-argument

is a PIC X(n) that holds the name of the file to be resolved. The fully resolved name is returned into this argument. If the file was found in a library only the simple name is returned in file-argument. If the file does not exist, the fully resolved name of where the file would be created is returned and the ON EXCEPTION clause is executed. ICLINK can be used.

lib-name

is a PIC X(n) that holds the fully resolved library name if the file-argument was found in a library. If not found in a library, this entry will be set to spaces.

search-path

is a PIC X(n) that holds the name of the **ICOBOL** search path to use. Valid search paths are ICDATAPATH, ICCODEPATH, and blank for no searching. If the argument is not specified, it defaults to ICDATAPATH. For an invalid argument, an error is returned and no processing is done.

file-entry

is a structure as defined below that provides status information about the file. If the file does not exist, no data is moved into this structure. The Filename piece of the structure can be any length but should be long enough to hold the longest simple name. Each entry can be defined as one of the following:

```
01  FILE-ENTRY-REV1.
    02 MODIFIED-INFO.
       03 DATE-MODIFIED              PIC 9(6).
       03 TIME-MODIFIED              PIC 9(8).
    02 ACCESSED-INFO.
       03 DATE-ACCESSED              PIC 9(6).
       03 TIME-ACCESSED              PIC 9(8).
    02 FILESIZE-BYTES                PIC 9(10).
    02 F-ATTRIBUTES                  PIC X(8).
    02 F-ATTRIBUTE-RED  REDEFINES F-ATTRIBUTES.
       03 READABLE-ON                PIC X(1).
       03 WRITABLE-ON                PIC X(1).
       03 PROTECTABLE-ON             PIC X(1).
       03 ARCHIVE-IT                 PIC X(1).
       03 DIRECTORY-TYPE             PIC X(1).
       03 SYSTEM-TYPE                PIC X(1).
       03 EXECUTABLE-TYPE            PIC X(1).
       03 FILLER                     PIC X(1).
    02 FILENAME                      PIC X(64).
```

or

```
01  FILE-ENTRY-REV2.
    02 MODIFIED-INFO.
       03 DATE-MODIFIED              PIC 9(8).
       03 TIME-MODIFIED              PIC 9(8).
    02 ACCESSED-INFO.
       03 DATE-ACCESSED              PIC 9(8).
       03 TIME-ACCESSED              PIC 9(8).
    02 FILESIZE-BYTES                PIC 9(10).
    02 F-ATTRIBUTES                  PIC X(8).
    02 F-ATTRIBUTE-RED  REDEFINES F-ATTRIBUTES.
       03 READABLE-ON                PIC X(1).
       03 WRITABLE-ON                PIC X(1).
       03 PROTECTABLE-ON             PIC X(1).
       03 ARCHIVE-IT                 PIC X(1).
       03 DIRECTORY-TYPE             PIC X(1).
       03 SYSTEM-TYPE                PIC X(1).
       03 EXECUTABLE-TYPE            PIC X(1).
       03 FILLER                     PIC X(1).
    02 FILENAME                      PIC X(64).
```

or

```
01  FILE-ENTRY-REV3.
    02 MODIFIED-INFO.
       03 DATE-MODIFIED              PIC 9(8).
       03 TIME-MODIFIED              PIC 9(8).
    02 ACCESSED-INFO.
       03 DATE-ACCESSED              PIC 9(8).
       03 TIME-ACCESSED              PIC 9(8).
    02 FILESIZE-BYTES                PIC 9(10).
    02 F-ATTRIBUTES                  PIC X(8).
    02 F-ATTRIBUTE-RED  REDEFINES F-ATTRIBUTES.
       03 READABLE-ON                PIC X(1).
       03 WRITABLE-ON                PIC X(1).
       03 PROTECTABLE-ON             PIC X(1).
       03 ARCHIVE-IT                 PIC X(1).
       03 DIRECTORY-TYPE             PIC X(1).
       03 SYSTEM-TYPE                PIC X(1).
       03 EXECUTABLE-TYPE            PIC X(1).
       03 LINK-TYPE                  PIC X(1).
    02 USER-COUNT                    PIC 9(5).
    02 FILENAME                      PIC X(64).
```

rev

is a PIC 9(2) COMP (one-byte binary), that specifies the revision of the file-entry lines provided in the output-file. If not specified, 1 is assumed. Valid entries are 1 ,2, and 3. If not specified, **ANSI 74 and ANSI 85** default to rev1, and ***VXCOBOL*** defaults to 3.

In the rev 1 structure each date is of the form YYMMDD. In the rev 2 and rev 3 structures each date is of the form YYYYMMDD and each time is of the form hhmmsshh. The USER-COUNT field returns the number of times the file is open to any **ICOBOL** runtime running on this machine. The attribute field is a space if the particular attribute is not set, and contains a single uppercase letter if it is set. (R-readable, W-writeable, P-protected, A-archive, D-directory, S-system, E-executable, L-linkfile).

If an error is generated, the ON EXCEPTION clause, if present, is executed. The EXCEPTION STATUS gives the error.

B.47. IC_SEND_MAIL (Added in 3.50)

The IC_SEND_MAIL builtin allows a COBOL program to send email directly from within COBOL using a standard SMTP server.

The syntax is:

```
CALL "IC_SEND_MAIL" USING to-list, from-addr, cc-list, bcc-list, subject,
   message [[, att-type, attachment] [, uname-pass]]
```

Where the parameters are strings that hold:

to-list	address [, address]...	(comma-separated list)
from-addr	address	
cc-list	[address [, address]...]	(comma-separated list)
bcc-list	[address [, address]...]	(comma-separated list)
subject	subject-line	
message	text-body-of-message	(use <cr><nl> to split lines)
att-type	mime type of file attached	(optional)
attachment	filename of file to attach	(req'd if att-type specified)
uname-pass	a username-password pair to pass as authorization to the SMTP server. password should be separated by a comma from username.	

All strings are trimmed of trailing blanks. A LOW_VALUE will terminate a string.

The number of parameters are checked along with at least one valid to-address is required, the from-address can not be blank and should be a valid address, cc-list, bcc-list, att-type, attachment, and uname-pass can be blank, subject cannot be blank, message can use <cr><nl> to split lines and can be blank.

If an att_type is specified, an attachment must be specified. If no att-type is specified, no attachment can be specified.

Some valid mime-types are text, video, ... Attachment must specify a valid file.

For Mime Types see:

```
http://www.iana.org/assignments/media%2Dtypes/index.htm
```

Environment variables:

ICSMTPSERVER	required to tell where the SMTP server is located. There is no default.
ICSMTPPORT	specifies the port for the SMTP server, 25 is the default

Error messages for this builtin include: :

1301	A mail recipient must be specified (To: field)
1302	A mail sender must be specified (From: field)
1303	A message subject must be specified (Subject: field)
1304	No mail server was specified (ICSMTPSERVER environment variable)
1305	The mail server port was not valid (ICSMTPPORT environment variable)
1306	SMTP System or Help message
1307	The SMTP service is ready
1308	The SMTP service is closing
1309	The SMTP action completed OK
1310	The recipient is nonlocal, message is being forwarded
1311	The recipient was not verified but message was accepted
1312	Start message input and end with <CRLF>.<CRLF>

1313 The SMTP service is not available - closing connection
1314 The command failed because the user's mailbox was unavailable
1315 The command failed because of a server error
1316 The command failed because of insufficient server storage

1318 The SMTP command failed with a 500 level error
1319 The SMTP command failed because mailbox was unavailable
2079 SMTP Authorization successful
2112 SMTP Authorization in progress
2113 SMTP Authorization required (A username/password is required)
2114 SMTP Authorization failed

A sample **ICOBOL** source that provides an interface to the new IC_SEND_MAIL builtin is available as sendmail.sr in the examples directory.

In the example program, sendmail.sr, the environment variable ICSMTPSERVER is read and can be set. A message can be composed and sent to people with both CC and BC addresses. An attachment can also be added.

B.48. IC_SEND_MSG

The IC_SEND_MSG builtin allows the user to send a message to one, several, or all logged-on **ICOBOL** users, either active or inactive on the same machine.

The IC_SEND_MSG builtin is enabled with the Message sending privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 221 "The operation is not permitted.".

Two modes are available.

Mode 1 (Interactive Mode)

For mode 1, the syntax is:

```
CALL "IC_SEND_MSG"
```

Upon invocation, a terminal status window of all logged on terminals is displayed. You are then prompted for the message that you wish to send. You are then prompted for the terminal number to send the message to. If none, the message is sent to all logged-on users.

For more on IC_SEND_MSG in mode 1 see the Message Broadcast utility in the Utilities Manual.

Mode 2 (Program Mode)

For mode 2, the syntax is:

```
CALL "IC_SEND_MSG" USING term-number, message [, line-no, col-no ]
```

Where

term-number
: is a PIC 9(4) COMP that specifies the terminal number to send the message to. A value of 65535 sends the message to all users.

message
: is a PIC X(n) string of the message to be sent. Trailing spaces and nulls are removed. If longer than 60 the string is truncated.

line-no
: is a numeric that specifies the row (0-255) of where to place the message on the terminal

col-no
: is a numeric that specifies the column (0-255) of where to place the message on the terminal

NOTE: For line and column 0,0 is the upper-left corner of the terminal.

If *n* is an invalid terminal number or is not currently active, an Exception Status 228 "The terminal is not logged on" is returned. If *n* is a terminal which is not enabled, Exception Status 229 "The terminal is not configured into the system" is returned.

When the line and column format is specified, the message can include DG attribute characters like reverse, blink, dim-on, dim-off, etc. On the specified terminal, all attributes will be reset at the end of the message and the cursor position will be returned to its starting position.

B.49. IC_SERIAL_NUMBER

The IC_SERIAL_NUMBER builtin returns the **ICOBOL** runtime license serial number, as determined by the license manager (ICPERMIT) from the current license activation key.

The syntax is:

```
CALL "IC_SERIAL_NUMBER" USING argument
```

Where

argument
: is a PIC 9(8) into which the license serial number will be stored.

B.50. IC_SET_ENV (Added in 3.13)

The IC_SET_ENV builtin allows an environment entry to be set.

The syntax is:

```
CALL "IC_SET_ENV" USING name, value
```

Where

name
: is a string that specifies the name of the environment variable to be set.

value
: is a string that specifies the data value for the environment entry. Trailing spaces are ignored.

Possible errors include:

Parameter mismatch
Invalid Data
No memory

B.51. IC_SET_TIMEOUT

The IC_SET_TIMEOUT builtin allows default timeouts to be enabled and disabled for ACCEPT and STOP literal statements.

The syntax is:

```
CALL "IC_SET_TIMEOUT" [ USING timeout ]
```

Where

timeout

is a PIC 9(4) COMP that specifies the default timeout in tenths of seconds. 65535 disables timeout, 65534 says to use that specified as the global timeout (ICTIMEOUT), while a number between 0 and 63000 will set the timeout to that value.

If no argument is specified, wait forever is set. The timeout value remains in effect whenever this program is active. I.E., if a CALL statement is made, while in the new program the timeout is reset to that specified by the global timeout (ICTIMEOUT) for the new program. Upon returning to the calling program, the timeout is restored to be the value that was set before the CALL.

When an ACCEPT statement times out, ESCAPE KEY is set to 99 and no data is moved to the particular item (just as when an ESC key is pressed).

B.52. IC_SET_USERNAME

The IC_SET_USERNAME builtin allows a program to set or change the name that is returned from the ACCEPT FROM USER NAME statement. On Windows, it also changes the owner and printed-by names used by the Printer Control Utility. This call does not change any identification of the user known to the operating system. In particular, the UNIX user-id for the process remains unchanged. (Hence, the permissions required on UNIX to perform certain Printer Control Utility functions remain unchanged after making this call.)

The syntax is:

```
CALL "IC_SET_USERNAME" USING username
```

Where

username

is a PIC X(n) string (1 <= n <= 15) containing the new username. The new username will consist of the characters from this string up through the first space or null.

On UNIX, Exception Status 13 "Invalid data", is returned if the name string does not represent a name of at least one character, i.e., you may not eliminate the username entirely.

Any username case conversion specified on the ICRUN command line will be applied to the username supplied to this builtin.

B.53. IC_SHUTDOWN (Added in 3.00)

The IC_SHUTDOWN builtin terminates the runtime system. The IC_SHUTDOWN call is enabled with the System Shutdown privilege in the Program Environment configuration of the configuration file (.cfi). If the optional argument is specified the value is returned to the runtime system's parent process as an exit code.

The syntax is:

```
CALL "IC_SHUTDOWN" [ USING exit-code ]
```

Where

exit-code

is a PIC 9(n). *Exit-code* may be zero, or any value between 10 and 255 inclusive. Values 1 thru 9 are reserved for runtime use.

Any error is stored into Exception Status and the ON EXCEPTION clause, if present, is executed. Exception status 13 "Invalid data" is returned if the *exit-code* is out of range.

The IC_SYS_INFO builtin is supported in **ICOBOL** to allow internal status information for the entire **ICOBOL** system to be viewed or read. The optional argument was added in 3.22.

The IC_SYS_INFO builtin is enabled with the System Information privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 203 "Program not found.".

The syntax is:

```
CALL "IC_SYS_INFO" [ USING sys-info-struc ].
```

Where

sys-info-struc

is a structure with the following format:

```
01 SYS-INFO-STRUCTURE.
  02 REV                    PIC 9(4) COMP.
  02 PROCESSES.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 TERMINALS.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 RPTERMINALS.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 DTTERMINALS.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 ANTERMINALS.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 SEQ-FILES.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 REL-FILES.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 IND-FILES.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 REC-LOCKS.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 OS-FILES.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 PRN-FILES.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
  02 PCQ-FILES.
         03 IN-USE          PIC 9(4) COMP.
         03 MAX-USED        PIC 9(4) COMP.
         03 TOTAL-COUNT     PIC 9(4) COMP.
```

```
        02 PCQ-JOBS.
              03 IN-USE         PIC 9(4) COMP.
              03 MAX-USED       PIC 9(4) COMP.
              03 TOTAL-COUNT    PIC 9(4) COMP.
        02 SER-FILES.
              03 IN-USE         PIC 9(4) COMP.
              03 MAX-USED       PIC 9(4) COMP.
              03 TOTAL-COUNT    PIC 9(4) COMP.
        02 CON-FILES.
              03 IN-USE         PIC 9(4) COMP.
              03 MAX-USED       PIC 9(4) COMP.
              03 TOTAL-COUNT    PIC 9(4) COMP.
        02 BUFFERS.
              03 IN-USE         PIC 9(4) COMP.
              03 MAX-USED       PIC 9(4) COMP.
              03 TOTAL-COUNT    PIC 9(4) COMP.
              03 IN-LRU         PIC 9(4) COMP.
              03 MIN-LRU        PIC 9(4) COMP.
              03 SIZE-BYTES     PIC 9(4) COMP.
        02 DEVICES.
              03 IN-USE         PIC 9(4) COMP.
              03 MAX-USED       PIC 9(4) COMP.
              03 TOTAL-COUNT    PIC 9(4) COMP.
        02 NEWBUFFERS.                          Added in 3.35 (rev 2)
              03 FILLER         PIC 9(4) COMP.
              03 IN-USE         PIC 9(9) COMP.
              03 MAX-USED       PIC 9(9) COMP.
              03 TOTAL-COUNT    PIC 9(9) COMP.
              03 IN-LRU         PIC 9(9) COMP.
              03 MIN-LRU        PIC 9(9) COMP.
              03 SIZE-BYTES     PIC 9(9) COMP.
```

In the optional argument is given then it is filled in with the appropriate information. If the argument is not given, then a screen of statistical information about various **ICOBOL** parameters is shown.

For the named resource, three numbers are provided. These are:

Value	Description
In Use	The number currently in use
MaxUsed	The most this has ever been, for this invocation
Max	The maximum number configured

The MaxUsed values can be used to either raise or lower individual System Parameters in the configuration file (.cfi), the CONFIG.SYS file (on Windows), or in the UNIX Kernel (on UNIX) to provide a better-tuned system.

In the *sys-info-structure*, REV will contain a 2 when used with ICRUN 3.35 and up, and a 1 with previous versions.

Note that the SYS-INFO-STRUCTURE has been upgraded in 3.35 to handle the larger buffer structure added in 3.30.

The rev returned will now be 2.

The previous BUFFER section of PIC 9(4) COMP values will be set to zero.

A new BUFFER section has been added at the end of the structure and all the sizes will be PIC 9(9) COMP.

The new structure is now 136 bytes long.

When passed an old strucuture, the new buffer information will NOT be stored.

B.55. IC_TERM_CTRL

The IC_TERM_CTRL builtin allows the user to view the status of all terminals and optionally perform some operation on one or more selected terminals.

- Users with the Message sending privilege can send messages to one or more terminals.
- Users with the Abort terminal privilege can abort or kill a terminal.
- Users with the Watch other terminals privilege can view the screen of a selected terminal and optionally control the selected terminal's keyboard.

The IC_TERM_CTRL builtin is enabled with the Terminal status privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled an Exception Status 221 "This operation is not permitted" will be given. The Abort terminal, Console Interrupt, Message sending, and Watch privileges enable the commands available to the user.

The syntax is:

```
CALL "IC_TERM_CTRL" [ USING term-num, watch-func-struc ].
```

Where

term-num

is a PIC 9(4) COMP that holds the terminal number of the terminal to be accessed.

watch-func-struc

is a structure holding the requested function and status-line setting for the session. It looks like:

```
01  WATCH-FUNC-STRUC.
    02 FUNCTION-CODE    PIC 99 COMP.
    02 STATUS-LINE-CODE PIC 99 COMP.
```

Valid settings for FUNCTION-CODE are:

0=Watch *term-num*
1=Control *term-num*

Valid settings for STATUS-LINE-CODE are:

0=no status line
1=status line at the top of the screen starting in col 1
2=status line at the top of the screen starting in col 41

When the optional parameters are given, the IC_TERM_CTRL screen is never seen, the appropriate function is executed and on exit control is returned to the calling program.

All terminals running ICOBOL on this machine are available.

For more on IC_TERM_CTRL see the Terminal Control Utility in the Utilities manual.

B.56. IC_TERM_STAT

The IC_TERM_STAT builtin is enabled with the Terminal status privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 221 "This operation is not permitted.".

The syntax is:

```
CALL "IC_TERM_STAT" [ USING term-num, stat-struc ]
```

Where

term-num

is a PIC 9(4) COMP that holds the terminal number of a particular console of which status information is required. 65535 means do a terminal status on the current console.

stat-struc

is a structure holding specific status information for the given terminal. The structure looks like:

```
01 STAT-STRUC.
    02 STRUC-REV                    PIC 9(4) COMP.
    02 TERM-NUM                     PIC 9(4) COMP.
    02 PROC-PID                     PIC 9(9) COMP.
    02 PROC-NUM                     PIC 9(4) COMP.
    02 PGM-STATE                    PIC 9(4) COMP.
    02 PGM-PC                       PIC 9(4) COMP.
    02 SP2-SRV-FLAG                 PIC X.
    02 CHAR-SRV-FLAG                PIC X.
    02 USERNAME                     PIC X(16).
    02 PGMNAME                      PIC X(30).
    02 FLAGS.
       03 MASTER-FLAG               PIC X.
       03 BATCH-FLAG                PIC X.
       03 ABORT-FLAG                PIC X.
       03 BREAK-FLAG                PIC X.
       03 DEBUG-FLAG                PIC X.
       03 SYS-INFO-FLAG             PIC X.
       03 MSG-FLAG                  PIC X.
       03 BACK-FLAG                 PIC X.
       03 PRINTER-FLAG              PIC X.
       03 PRT-MGR-FLAG              PIC X.
       03 SHUTDOWN-FLAG             PIC X.
       03 TERM-FLAG                 PIC X.
       03 EXEC-FLAG                 PIC X.
       03 WATCH-FLAG                PIC X.
       03 XWATCH-FLAG               PIC X.
       03 FILLER                    PIC X(3).
```

Terminal Status allows the user to view the status of all **ICOBOL** users on the machine as well as current system information.

For more on IC_TERM_STAT with no arguments see the Terminal Status utility in the Utilities Manual.

STRUCT-REV returns a 3 using the above description.

TERM-NUM is the terminal number of the console (it is the same as term-num or the current console number if term-num was set to 65535.

PROC-PID is the pid of the stated terminal.

PROC-NUM is the internal **ICOBOL** process number.

PGM-STATE is the current state of the process: 0=unused, 1=logging on/off, 2=inactive, 3=active, 4=stopped, 5=while debugging, 6=pushed to shell/executable, 7=watching, 8=defunct.

PGM-PC is always zero.

Previous to ICOBOL 3.30 the two bytes after the PGM-PC were filler bytes. With 3.30 and up they hold the following flags:

```
02 SP2-SRV-FLAG          PIC X.
02 CHAR-SRV-FLAG         PIC X.
```

If the process is using ICTHINS then the first flag (SP2-SRV_FLAG) will be set to "Y", if the process is using ICRUNRS then the second flag (CHAR-SRV-FLAG) will be set to "Y".

USERNAME is the name of the user of the terminal.

PGMNAME is the currently executing COBOL program.

Each of the flags is either Y or N indicating that the console is the master, is a batch job, or has a specific privilege. The XWATCH-FLAG was added in 3.30.

B.57. IC_TRIM (Added in 3.34)

The IC_TRIM builtin takes a string and returns the starting position in the string of the first non-blank character and the actual length of the string NOT including trailing spaces.

The syntax is:

```
CALL "IC_TRIM" USING string, num-start, num-len
```

Where

string
specifies a PIC X(n) that holds the string.

num-start
is a PIC 9(n) to which is returned the starting position in string for the first non-blank character.

num_len
is a PIC 9(n) to which is returned the actual length of the string NOT including trailing spaces.

NOTE: Num-len and num-start can return 0.

Examples:

For string " abcd   " num-start would return 3 and num-len would return 4.

For string "   " num-start and num-len would return 0.

For string "abc d   ", num-start would return 1 and num-len would return 5.

The ‘trimmed’ string can then be extracted with reference modification. (I.E., string (start:len))

B.58. IC_UPPER

The IC_UPPER builtin converts the specified string to all upper-case characters.

The syntax is:

```
CALL "IC_UPPER" USING string
```

Where

string

specifies a PIC X(n) that holds the data to be converted to upper-case.

NOTE: For **ANSI 74 and ANSI 85**, a more efficient way of accomplishing this task is by using the UPPER-CASE intrinsic function:

```
MOVE FUNCTION UPPER-CASE ( string ) TO string.
```

B.59. IC_VERSION

The IC_VERSION builtin allows a program to obtain the revision number of the runtime system which is executing the current program.

The syntax is:

```
CALL "IC_VERSION" USING version-string
```

Where

version-string

specifies a PIC X(n) item which will receive the revision number of the string. n >= 16 is suggested. The version string will be of the form x.xx[.xx][Beta x] where each x represents a digit and parts within brackets are present only if the runtime is an update and/or Beta test version.

The following exception status codes may be returned:

209 (Parameter mismatch on call) Too many or too few USING arguments

In the following example, a runtime system with revision 2.51 Beta 1 would display "2.51 Beta 1 ", and the 3.00 final release would display “3.00 “.

```
01 RUNTIME-VERSION      PIC X(16).
       ...
       CALL "IC_VERSION" USING RUNTIME-VERSION.
       DISPLAY QUOTE, RUNTIME-VERSION, QUOTE.
```

B.60. IC_WINDOW_TITLE

The IC_WINDOW_TITLE builtin allows the caller to set the title bar of the GUI icrunw or icrunrc screen. This title remains in effect until another call to IC_WINDOW_TITLE. This call is available under Windows and when running ThinClient Client (icrunrc) under Windows.

The syntax is:

```
CALL "IC_WINDOW_TITLE" USING fmt [, string-1 ]...
```

Where

fmt
: specifies a PIC X(n) that holds a format string on how to display the title.

String-1
: specifies a PIC X(n) that provides a series of character strings to be applied to the format.

The format string may contain any characters which will be literally used in the title. It may also contain any of the following:

Format string Characters	Description
%%	Insert a per cent sign at this location
%C	Insert the name of the COBOL program at this location (only allowed once)
%D	Insert today's date at this location
%P	Insert the process name at this location
%S	Insert the next string argument at this location
%T	Insert the current time at this location

Any other character following the % will be treated as invalid. Trailing spaces or low-values will be trimmed from the format string.

Without this call, the default is"%P - %C".

The string values are used to satisfy the %S format directive. They are inserted in their entirety, including trailing spaces, unless a LOW-VALUE is encountered in which case the string ends at the preceeding character.

The following exception status codes may be returned:

Exception status	Description
209 - Parameter mismatch on call	- Too few arguments - more string parameters than %S directives
13 - Invalid data	- bad format string
39 - Out of disk space	- constructed string exceeds buffer size

When this builtin is NOT available in the Thinclient case, an error 231 - "Unsupported feature for the current terminal type" will be given.

Example:

Suppose these lines appear in the program MYPROG.SR:

```
01 FORMAT      PIC X(50) VALUE "%S: My application (%C)".
01 STRING-1    PIC X(10) VALUE "My company".

CALL "ic_window_title" USING FORMAT, STRING-1 ON EXCEPTION
```

The window title would be set to:

```
"My Company: My Application (myprog)".
```

The program name (myprog) would change as your application calls different COBOL programs.

B.61. IC_WINDOWS_MSG_BOX

The IC_WINDOWS_MSG_BOX builtin allows a COBOL program to display a message box using the Windows MessageBox function when the console is on a Windows machine either with Gui runtime (icrunw) orThinCleint (icrunrc).

The syntax is:

```
CALL "IC_WINDOWS_MSG_BOX" USING msg-text, msg-title, msg-ctrl, msg-button
```

Where

msg-text

specifies a PIC X(n) item containing the message to be displayed. Trailing spaces are ignored.

msg-title

specifies a PIC X(n) item containing the title for the dialog box. Windows does not automatically break the lines to fit in the message box so the message string must contain new lines to break the lines at the appropriate places. Trailing spaces are ignored.

msg-ctrl

specifies a PIC X(3) item which controls the appearance and behavior of the message box

The first character controls the type of message box that will be displayed, i.e., the number of buttons and their legends. The valid character values and their meanings are:

1st character	Number of buttons to display	Legends for the buttons
SPACE	one push button	OK
"0"	one push button	OK
"1"	two push buttons	OK and Cancel
"2"	two push buttons	Retry and Cancel
"3"	two push buttons	Yes and No
"4"	three push buttons	Yes, No, and Cancel
"5"	three push buttons	Abort, Retry, and Ignore

The second character controls which of the buttons will be the default. The selected button must correspond to one of the buttons that is available as a result of the value entered in the first character position. If the Enter key is pressed this button is selected. The valid selections follow:

2nd character	Default button
SPACE	First button on the message box
"A"	Abort button
"C"	Cancel button
"I"	Ignore button
"N"	No button
"O"	OK button
"R"	Retry button
"Y"	Yes button

The third character controls the icon which is displayed in the message box. Valid selections are as follows:

3rd character	Icon to display in message box
SPACE	No icon
"!"	An exclamation point
"?"	A question mark
"I"	Icon consists of the lower case letter i
"X"	A stop sign icon

msg-button

specifies a PIC X(1) item which is returned to from the call. This value identifies which button was selected. If the message box has a Cancel button, pressing ESC is equivalent to selecting the Cancel button.

Value returned from the call	Meaning (which button was selected)
"A"	Abort button
"C"	Cancel button
"I"	Ignore button
"N"	No button
"O"	OK button
"R"	Retry button
"Y"	Yes button

The following exception status codes may be returned

Exception status	Description
209 (Parameter mismatch on call)	Too many or too few USING arguments, or invalid length for argument
13 (Invalid data)	One or more values in msg-ctrl is not valid
8 (Insufficient memory)	Not enough memory to execute call

When this builtin is NOT available in the Thinclient case, an error 231 - "Unsupported feature for the current terminal type" will be given.

Example:

Suppose the following code appears in a program running on Windows:

```
01 MSG-TEXT  PIC X(32) VALUE "The account number is not valid.".
01 MSG-TITLE PIC X(29) VALUE "Error: Invalid account number".
01 MSG-CTRL  PIC X(3)  VALUE "2RX".
...
CALL "IC_WINDOWS_MSG_BOX" USING MSG-TEXT, MSG-TITLE, MSG-CTRL,
     MSG-BUTTON.
```

A message box will appear containing 2 buttons -- retry and cancel. The box will have the title "Error: Invalid account number" and contain a stop sign icon and the text "The account number is not valid." as shown in the next image.

Error: Invalid account number.
The account number is not valid.
Retry
Cancel

B.62. IC_WINDOWS_SETFONT (Added in 3.13)

The IC_WINDOWS_SETFONT builtin allows the current font and font-size to be changed when running the GUI Windows runtime or ThinClient Client (icrunrc). This call will also run on UNIX platforms when using the ThinClient client (icrunrc) on Windows. (Added in 3.30)

The syntax is:

```
CALL "IC_WINDOWS_SETFONT" [ USING fontname, fontsize ]
```

Where

fontname
: is a string that specifies the name of a fixed font to which the font should be set. If not specified, the default usually "Courier New" will be used. The name of the selected font will be returned into this location if the call is successful.

fontsize
: is an integer that specifies the font point size from 2 to 99, or is an integer that specifies the font point size in deci-points from 20 to 999 as the number 1020 thru 1999, or a zero which will cause the Windows ChooseFont Dialog box to be shown to the user.

This call acts just like the ICFONT and ICFONTSIZE environment entries when ICRUNW is started.

This call is available only on Windows when running on a graphic desktop and using the GUI runtime (icrunw) or when running the ThinClient (icrunrc). (It does not matter if the ICRUNRC is connected to a UNIX server.)

If no parameters are specified, the default ChooseFont dialog will be displayed. (Added in 3.56)

Possible errors include:

Parameter mismatch	Invalid Data
Program not found	No memory
Path not found	Invalid Format
Invald operation	

When this builtin is NOT available in the Thinclient case, an error 231 - "Unsupported feature for the current terminal type" will be given.

B.63. IC_WINDOWS_SHELLEXECUTE (Windows only) (Added in 3.11)

The IC_WINDOWS_SHELLEXECUTE performs an operation on a specified file.

The syntax is:

```
CALL "IC_WINDOWS_SHELLEXECUTE" USING lpverb, lpFile, lpParameters,
   lpDirectory, nShowCmd
```

Where

lpverb

Is a string, referred to as a verb, that specifies the action to be performed. The set of available verbs depends on the particular file or folder. Generally the actions available from an object's context menu are available verbs. The following verbs are commonly used:

edit	Launches an editor and opens the document for editing.
explore	Explores the folder specified by *lpFile*.
find	Initiates a search starting from the specified directory.
open	Opens the file specified by *lpFile*.
print	Prints the document specified by *lpFile*.
properties	Displays the file or folder's properties.

If set to spaces, then NULL is passed to the Windows function which defaults to the "default" verb or an open.

lpFile

is a string that specifies the file or object on which to execute the specified verb.

lpParameters

is a string that is a string of parameters to be passed to the application specified by *lpFile* if *lpFile* is an executable. If *lpFile* is a document then *lpParameters* should be spaces.

lpDirectory

is a string that specifies the default directory. If set to spaces the current directory is used. (NULL is passed to the Windows call.)

nShowcmd

is a Numeric with a value as given under IC_WINDOWS_SHOW_CONSOLE as *cmd*.

If the Windows ShellExecute call returns with a value greater than 32 than IC_WINDOWS_SHELLEXECUTE returns a success. Otherwise, it is an error and an exception is generated and the ON EXCEPTION clause is executed, if provided.

This call is available only on Windows when running on a graphic desktop.

More on this can be seen by looking at the Microsoft call "ShellExecute".

Possible errors include:

Parameter mismatch	Invalid Data
Program not found	No memory
Path not found	Invalid Format
Access Denied	Sharing violation
Invalid operation	

This call can be used to:

A) start Internet Explorer by giving a valid URL address (www.egns.com)
B) start an e-mail by giving "mailto: <name>".
C) start a find file by giving the verb "find" with lpFile set to a directory specifier.

Basically you should be able to do all the actions associated with an object that can be seen by using Explorer to view the file and then right-clicking on the object. The top entry in the list is the default selection.

B.64. IC_WINDOWS_SHOW_CONSOLE

The IC_WINDOWS_SHOW_CONSOLE builtin specifies how the window is to be shown when running with the GUI runtime or the Windows ThinClient (icrunrc) screen..

The syntax is:

```
CALL "IC_WINDOWS_SHOW_CONSOLE" USING cmd
```

Where

cmd

specifies a PIC 9(2) COMP item with one of the following values:

Value	Command	Description
1	Hide	Hides the window.
2	Maximize	Maximizes the window.
3	Minimize	Minimizes the window and activates the next top-level window in the z-order.
4	Restore	Activates and displays the window. If the window is minimized or maximized, the system restores it to its original size and position. An application should specify this flag when restoring a minimized window.
5	Show	Activates the window and displays it in its current size and position.
6	ShowDefault	Sets the show state based on how the program was started.
7	ShowMaximized	Activates the window and displays it as a maximized window.
8	ShowMinimized	Activates the window and displays it as a minimized window.
9	ShowMinNoActive	Displays the window as a minimized window. The active window remains active.
10	ShowNA	Displays the window in its current state. The active window remains active.
11	ShowNoActivate	Displays a window in its most recent size and position. The active window remains active.
12	ShowNormal	Activates and displays a window. If the window is minimized or maximized, the system restores it to its original size and position.

NOTE: An Active window is where keyboard input is directed.

The following exception status codes may be returned

Exception status code	Description
209 (Parameter mismatch on call)	Too many or too few USING arguments, or invalid length for argument
13 (Invalid data)	*Cmd* is not valid

When this builtin is NOT available in the Thinclient case, an error 231 - "Unsupported feature for the current terminal type" will be given.

Example:

Suppose the following code appears in a program running on Windows:

```
MOVE 1 TO WINDOW-CMD.
CALL "IC_WINDOWS_SHOW_CONSOLE" USING WINDOW-CMD.
```

The current window will be hidden. If you had used a 3 (minimize), the current window would be minimized.

This builtin is useful with Sp2 programs since it can be used to hide the runtime system's main window if it is not needed and restored if it is needed.

VIII. INTRINSIC FUNCTIONS

(Added in 3.00)

A. General Description

Each intrinsic function definition specifies:

1) the name and description of the function

2) the type of the function

3) the general format of the function

4) the arguments, if any

5) the returned value.

See Page 129, Function-identifier, for rules and explanations on the referencing of functions.

A.1. Types of Functions

Types of intrinsic functions are:

1) Alphanumeric functions. These are of the class and category alphanumeric. The number of character positions in this data item is specified in the function definition. Alphanumeric functions have an implicit usage display. Unless stated otherwise in the definition of a function, the data item is represented in the alphanumeric coded character set in effect when the function is referenced at runtime.

2) Numeric functions. These are of the class and category numeric. A numeric function has an operational sign.

3) Integer functions. These are of the class and category numeric. An integer function has an operational sign and no digits to the right of the decimal point.

4) Index functions. These are of the class and category index.

A.2. Arguments

Arguments specify values used in the evaluation of a function. Arguments are specified in the function-identifier. The definition of a function specifies the number of arguments required, which may be zero, one, or more. For some functions, the number of arguments may be variable. The order in which arguments are specified in a function-identifier determines the interpretation given to each value in arriving at the function value.

Arguments may be required to have a certain class or a subset of a certain class, to be a keyword, a type declaration, or a mnemonic-name. The types of argument are:

1) Alphabetic. An elementary data item of the class alphabetic or an alphanumeric literal containing only alphabetic characters shall be specified. The size associated with the argument may be used in determining the value of the function.

2) Alphanumeric. A data item of the class alphabetic or alphanumeric or an alphanumeric literal shall be specified. The size associated with the argument may be used in determining the value of the function.

3) Index. An index data item shall be specified. The size associated with the argument may be used in determining the value of the function.

4) Integer. An arithmetic expression that will always result in an integer value or an integer data item shall be specified. The value of the arithmetic expression, including operational sign, is used in determining the value of the function.

5) Numeric. An arithmetic expression or a numeric data item shall be specified. The value of the arithmetic expression is used in determining the value of the function.

6) Pointer. A pointer identifier shall be specified. The size associated with the argument may be used in determining the value of the function.

NOTE: Permissible value errors return 0 for numeric and integer functions as well as raise the SIZE error condition. It is advisable to use arithmetic statements with SIZE ERROR checks when assigning function values if there is any possibility that improper arguments are supplied or if a size error could occur.

A.3. Returned values

The evaluation of a function produces a returned value in a temporary elementary data item. The type of a function identifies the type of the returned value as specified in the section, Types of functions, on Page 579.

The returned value rules for certain integer and numeric intrinsic functions contain one or more equivalent arithmetic expressions. An equivalent arithmetic expression is a formal definition that defines the relationship among a function, its arguments, and its returned value. In the presentation of the equivalent arithmetic expressions where there is a variable number of occurrences of an argument, the rules may contain an equivalent arithmetic expression for one, two, and *n* occurrences.

The returned value of numeric and integer functions depends on whether an equivalent arithmetic expression is specified for the function.

The returned value for numeric and integer functions is contained in a temporary standard intermediate data item. With the exception of the CURRENT-DATE function, DATE-TO-YYYYMMDD function, the DAY-TO-YYYYDDD function, the RANDOM function when no argument is specified, and the YEAR-TO-YYYY function, the returned value shall be the same for all instances of a given function within a single execution of the runtime element so long as the value and order of the arguments are the same.

When an equivalent arithmetic expression is specified:

1) the returned value shall equal the value of the equivalent arithmetic expression.

NOTE — As a result, the relation condition

function-identifier = equivalent-arithmetic-expression

will evaluate to true.

A.4. Date conversion functions

The Gregorian calendar is used in the date conversion functions. The starting date of Monday, January 1, 1601, was chosen to establish a simple relationship between the Standard Date and DAY-OF-WEEK: integer date 1 was a Monday, DAY-OF-WEEK 1.

A.5. Summary of functions

The following table summarizes the intrinsic functions that are available.

The "arguments" column defines argument type and the "type" column defines the type of the function, as follows:

```
Alph    means       alphabetic
Anum    means       alphanumeric
Ind     means       index
Int     means       integer
Num     means       numeric
Ptr     means       pointer
```

NOTE — Num in the arguments column includes Int. Both Int and Num are listed in the arguments column when the type of the argument determines the type of the function.

Intrinsic-function-name	Arguments	Type	Value returned
ABS	Int1 or Num1	Depends upon argument	The absolute value of argument
ACOS	Num1	Num	Arccosine of Num1
ANNUITY	Num1, Int2	Num	Ratio of annuity paid for Int2 periods at interest of Num1 to initial investment of one
ASIN	Num1	Num	Arcsine of Num1
ATAN	Num1	Num	Arctangent of Num1
BYTE-LENGTH	Alph1 or Anum1 or Ind1 or Num1 or Ptr1	Int	Length of argument in number of bytes
CHAR	Int1	Anum	Character in position Int1 of the alphanumeric program collating sequence
COS	Num1	Num	Cosine of Num1
CURRENT-DATE		Anum	Current date and time and difference from Coordinated Universal Time
DATE-OF-INTEGER	Int1	Int	Standard date equivalent (YYYYMMDD) of integer date
DATE-TO-YYYYMMDD	Int1, Int2	Int	Argument-1 converted from YYMMDD to YYYYMMDD based on the value of argument-2
DAY-OF-INTEGER	Int1	Int	Julian date equivalent (YYYYDDD) of integer date
DAY-TO-YYYYDDD	Int1, Int2	Int	Argument-1 converted from YYDDD to YYYYDDD based on the value of argument-2
E		Num	Value of *e*, the natural base
EXP	Num1	Num	*e* raised to the power Num1
EXP10	Num1	Num	10 raised to the power Num1
FACTORIAL	Int1	Int	Factorial of Int1
FRACTION-PART	Num1	Num	Fraction part of Num1
HIGHEST-ALGEBRAIC	Int1 or Num1 or Anum1	Int Num	Greatest algebraic value that may be represented in the argument
INTEGER	Num1	Int	The greatest integer not greater than Num1
INTEGER-OF-DATE	Int1	Int	Integer date equivalent of standard date (YYYYMMDD)
INTEGER-OF-DAY	Int1	Int	Integer date equivalent of Julian date (YYYYDDD)

Intrinsic-function-name	Arguments	Type	Value returned
INTEGER-PART	Num1	Int	Integer part of Num1
LENGTH	Alph1 or Anum1 or Ind1 or Num1 or Ptr1	Int	Length of argument in number of character positions
LOG	Num1	Num	Natural logarithm of Num1
LOG10	Num1	Num	Logarithm to base 10 of Num1
LOWER-CASE	Alph1 or Anum1	Depends upon argu-ment*	All letters in the argument are set to lower-case
LOWEST-ALGEBRAIC	Int1 or Num1 or Anum1	Int Num	Lowest algebraic value that may be represented in the argument.
MAX	Alph1 ... or Anum1 ... or Ind1 ... or Int1 ... or Num1 ...	Depends upon argu-ments*	Value of maximum argument
MEAN	Num1 ...	Num	Arithmetic mean of arguments
MEDIAN	Num1 ...	Num	Median of arguments
MIDRANGE	Num1 ...	Num	Mean of minimum and maximum arguments
MIN	Alph1 ... or Anum1 ... or Ind1... or Int1 ... or Num1 ...	Depends upon argu-ments*	Value of minimum argument
MOD	Int1, Int2	Int	Int1 modulo Int2
NUMVAL	Anum1	Num	Numeric value of simple numeric string
NUMVAL-C	Anum1 or Anum2	Num	Numeric value of numeric string with optional commas and currency sign
NUMVAL-F	Anum1	Num	Numeric value of numeric string representing a floating-point number
ORD	Alph1 or Anum1	Int	Ordinal position of the argument in collating sequence
ORD-MAX	Alph1 ... or Anum1 ... or Ind1 or Num1 ...	Int	Ordinal position of maximum argument
ORD-MIN	Alph1 ... or Anum1 ... or Ind1 or Num1 ...	Int	Ordinal position of minimum argument
PI		Num	The value of pi
PRESENT-VALUE	Num1, Num2 ...	Num	Present value of a series of future period-end amounts, Num2, at a discount rate of Num1
RANDOM	Int1	Num	Random number
RANGE	Int1 ... or Num1 ...	Depends upon argument	Value of maximum argument minus value of minimum argument
REM	Num1, Num2	Num	Remainder of Num1/Num2
REVERSE	Alph1 or Anum1	Depends upon argu-ment*	Reverse order of the characters of the argument

Intrinsic-function-name	Arguments	Type	Value returned
SIGN	Num1	Int	The sign of Num1
SIN	Num1	Num	Sine of Num1
SQRT	Num1	Num	Square root of Num1
STANDARD-DEVIATION	Num1 ...	Num	Standard deviation of arguments
SUM	Int1 ... or Num1 ...	Depends upon argu-ments	Sum of arguments
TAN	Num1	Num	Tangent of Num1
TEST-DATE-YYYYMMDD	Int1	Int	0 if Int1 is a valid standard date; otherwise identifies the sub-field in error
TEST-DAY-YYYYDDD	Int1	Int	0 if Int1 is a valid Julian date; otherwise identifies the sub-field in error
TEST-NUMVAL	Anum1	Int	0 if argument-1 conforms to the requirements of the NUMVAL function; otherwise identifies the character in error
TEST-NUMVAL-C	Anum1 or Anum2 or Key2 Key3	Int	0 if argument-1 conforms to the requirements of the NUMVAL-C function; otherwise identifies the character in error
TEST-NUMVAL-F	Anum1	Int	0 if argument-1 conforms to the requirements of the NUMVAL-F function; otherwise identifies the character in error
UPPER-CASE	Alph1 or Anum1	Depends upon argu-ment*	All letters in the argument are set to upper-case
VARIANCE	Num1 ...	Num	Variance of argument
WHEN-COMPILED		Anum	Date and time compilation unit was compiled
YEAR-TO-YYYY	Int1, Int2	Int	Argument-1 converted from YY to YYYY based on the value of argument-2
* A function that has only alphabetic arguments is type alphanumeric.			

TABLE 36. Summary of Intrinsic Functions

B. Intrinsic Functions

B.1. ABS

The ABS function returns the absolute value of the argument.

The type of this function depends on the argument type as follows:

Argument type	Function type
Integer	Integer
Numeric	Numeric

B.1.1 General format

FUNCTION ABS (*argument-1*)

B.1.2 Arguments

1) *Argument-1* shall be class numeric.

B.1.3 Returned values

1) The equivalent arithmetic expression shall be as follows:

a) When the value of *argument-1* is zero or positive,

(*argument-1*)

b) When the value of *argument-1* is negative,

(– *argument-1*)

B.1.4 Example

The following code fragments illustrate the use of this function.

```
01  A     PIC S999     VALUE -987.
01  B     PIC S999V99  VALUE -876.98.
01  C     PIC S999V99  VALUE 0.
01  D     PIC S999V99.

    COMPUTE D = FUNCTION ABS (A).
    IF D = 987
           PERFORM CORRECT-VALUE.

    COMPUTE D = FUNCTION ABS (B).
    IF D = 876.98
           PERFORM CORRECT-VALUE.

    COMPUTE D = FUNCTION ABS (C).
    IF D = 0
           PERFORM CORRECT-VALUE.
```

EXAMPLE 29. ABS function

B.2. ACOS

The ACOS function returns a numeric value in radians that approximates the arccosine of *argument-1*.

The type of this function is numeric.

B.2.1 General format

FUNCTION ACOS (*argument-1*)

B.2.2 Arguments

1) *Argument-1* shall be class numeric.

2) The value of *argument-1* shall be greater than or equal to –1 and less than or equal to +1.

B.2.3 Returned values

1) The returned value is the approximation of the arccosine of *argument-1* and is greater than or equal to zero and less than or equal to pi.

B.2.4 Example

The following code fragments illustrate the use of this function.

```
01  B                   PIC S9(10)          VALUE 4.
01  WS-NUM              PIC S9(5)V9(6).
01  MIN-RANGE           PIC S9(5)V9(7).
01  MAX-RANGE           PIC S9(5)V9(7).

    MOVE ZERO TO WS-NUM.
    MOVE  0.000000 TO MIN-RANGE.
    MOVE  0.000020 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION ACOS(1.0).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
        PERFORM CORRECT-VALUE.

    EVALUATE FUNCTION ACOS(0)
    WHEN 1.57076 THRU 1.57082
        PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    MOVE  -0.000040 TO MIN-RANGE.
    MOVE  0.00004 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION ACOS(IND (B) - 2).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
        PERFORM CORRECT-VALUE.
```

EXAMPLE 30. ACOS function

B.3. ANNUITY

The ANNUITY function (annuity immediate) returns a numeric value that approximates the ratio of an annuity paid at the end of each period for the number of periods specified by *argument-2* to an initial investment of one. Interest is earned at the rate specified by *argument-1* and is applied at the end of the period, before the payment.

The type of this function is numeric.

B.3.1 General format

FUNCTION ANNUITY (*argument-1*, *argument-2*)

B.3.2 Arguments

1) *Argument-1* shall be class numeric.

2) The value of *argument-1* shall be greater than or equal to zero.

3) *Argument-2* shall be a positive integer.

B.3.3 Returned values

1) The equivalent arithmetic expression shall be as follows:

   a) When the value of *argument-1* is zero,

      (1 / *argument-2*)

   b) When the value of argument-1 is not zero,

      (*argument-1* / (1 – (1 + *argument-1*)** (–*argument-2*)))

B.3.4 Example

The following code fragments illustrate the use of this function.

```
01  WS-NUM              PIC S9(5)V9(6).
01  MIN-RANGE           PIC S9(5)V9(7).
01  MAX-RANGE           PIC S9(5)V9(7).

    EVALUATE FUNCTION ANNUITY(2.9, 4)
    WHEN  2.91252 THRU  2.91264
        PERFORM CORRECT-VALUE
    WHEN OTHER
        PERFORM BAD-VAL.
    MOVE ZERO TO WS-NUM.
    MOVE 0.576553 TO MIN-RANGE.
    MOVE 0.576599 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION ANNUITY(
        FUNCTION ANNUITY(0, 3), 3).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
        PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    MOVE 4.49978 TO MIN-RANGE.
    MOVE 5.50022 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION ANNUITY(0, 2) + 5.
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
        PERFORM CORRECT-VALUE.
```

EXAMPLE 31. ANNUITY function

B.4. ASIN

The ASIN function returns a numeric value in radians that approximates the arcsine of *argument-1*.

The type of this function is numeric.

B.4.1 General format

FUNCTION ASIN (*argument-1*)

B.4.2 Arguments

1) *Argument-1* shall be class numeric.

2) The value of *argument-1* shall be greater than or equal to –1 and less than or equal to +1.

B.4.3 Returned values

1) The returned value is the approximation of the arcsine of *argument-1* and is greater than or equal to –pi/2 and less than or equal to +pi/2.

B.4.4 Example

The following code fragments illustrate the use of this function.

```
01  PI                  PIC S9V9(17) VALUE 3.141592654.
01  WS-NUM              PIC S9(5)V9(6).
01  MIN-RANGE           PIC S9(5)V9(7).
01  MAX-RANGE           PIC S9(5)V9(7).

    EVALUATE FUNCTION ASIN(0.5)
    WHEN  0.523588 THRU 0.523609
             PERFORM CORRECT-VALUE
    WHEN OTHER
             PERFORM BAD-VAL.

    MOVE ZERO TO WS-NUM.
    MOVE -1.52610 TO MIN-RANGE.
    MOVE -1.52604 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION ASIN(-.999).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                    PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    MOVE 0.142546 TO MIN-RANGE.
    MOVE 0.142558 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION ASIN(FUNCTION ASIN(PI - 3)).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                   PERFORM CORRECT-VALUE.
```

EXAMPLE 32. ASIN function

B.5. ATAN

The ATAN function returns a numeric value in radians that approximates the arctangent of *argument-1*.

The type of this function is numeric.

B.5.1 General format

FUNCTION ATAN (*argument-1*)

B.5.2 Arguments

1) *Argument-1* shall be class numeric.

B.5.3 Returned values

1) The returned value is the approximation of the arctangent of *argument-1* and is greater than –pi/2 and less than +pi/2.

B.5.4 Example

The following code fragments illustrate the use of this function.

```
01  B              PIC S9(10)      VALUE 2.
01  SQRT3          PIC S9V9(17)    VALUE 1.732050808.
01  ARR                            VALUE "40537".
    02  IND OCCURS 5 TIMES PIC 9.
01  WS-NUM         PIC S9(5)V9(6).
01  MIN-RANGE      PIC S9(5)V9(7).
01  MAX-RANGE      PIC S9(5)V9(7).

   MOVE ZERO TO WS-NUM.
   MOVE -0.785414 TO MIN-RANGE.
   MOVE -0.785382 TO MAX-RANGE.
   COMPUTE WS-NUM = FUNCTION ATAN(-1).
   IF (WS-NUM >= MIN-RANGE) AND (WS-NUM <= MAX-RANGE) THEN
           PERFORM CORRECT-VALUE.

   MOVE ZERO TO WS-NUM.
   MOVE -0.000020 TO MIN-RANGE.
   MOVE  0.000020 TO MAX-RANGE.
   COMPUTE WS-NUM = FUNCTION ATAN(IND(B)).
   IF (WS-NUM >= MIN-RANGE) AND (WS-NUM <= MAX-RANGE) THEN
           PERFORM CORRECT-VALUE.

   MOVE ZERO TO WS-NUM.
   MOVE 0.522827 TO MIN-RANGE.
   MOVE  0.522869 TO MAX-RANGE.
   COMPUTE WS-NUM = FUNCTION ATAN((1 / SQRT3) - .001).
   IF (WS-NUM >= MIN-RANGE) AND (WS-NUM <= MAX-RANGE) THEN
           PERFORM CORRECT-VALUE.
```

EXAMPLE 33. ATAN function

B.6. BYTE-LENGTH

The BYTE-LENGTH function returns an integer equal to the length of the argument in bytes.

The type of the function is integer.

B.6.1 General format

FUNCTION BYTE-LENGTH (*argument-1*)

B.6.2 Arguments

1) *Argument-1* shall be an alphanumeric literal or a data item of any class or category.

B.6.3 Returned values

1) If *argument-1* is an elementary data item or a literal, the returned value shall be an integer equal to the length of *argument-1* in bytes.

2) If *argument-1* is a group data item:

a) If *argument-1* or any data item subordinate to argument-1 is described with the DEPENDING phrase of the OCCURS clause, the returned value shall be an integer equal to the length of *argument-1* in bytes, as a sending operand, determined by evaluation of the data item specified in the DEPENDING phrase in accordance with the rules of the OCCURS clause. The contents of the data item specified in the DEPENDING phrase are used at the time the BYTE-LENGTH function is evaluated.

b) Otherwise, the value returned shall be an integer equal to the length of *argument-1* in bytes.

c) The returned length shall include the number of implicit FILLER positions, if any, in *argument-1*.

B.6.4 Example

The following code fragments illustrate the use of this function.

```
01  DATA-BLOCK.
    03  DATA-ARRAY OCCURS 1 TO 5 TIMES DEPENDING ON I.
        05  DATA-ELEMENTS OCCURS 65535 TIMES.
            07  DATA-COUNTER  PIC 9(18).
            07  ARRAY-VALUE   PIC 9(18).
            07  FILL-A-BYTE   PIC X(15).

01  A                         PIC S999       VALUE -999.
01  B                         PIC -999.99.
01  C                         PIC 9(9) COMP.
01  I                         PIC 99         VALUE 3.
01  NO-BYTES                  PIC 9(10).

    COMPUTE NO-BYTES = FUNCTION BYTE-LENGTH (A).
    IF NO-BYTES = 3 PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION BYTE-LENGTH (B).
    IF NO-BYTES = 7 PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION BYTE-LENGTH (C).
    IF NO-BYTES = 4 PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION BYTE-LENGTH (DATA-BLOCK).
    IF NO-BYTES = 10026855 PERFORM CORRECT-VALUE.

    MOVE 5 TO I.
    COMPUTE NO-BYTES = FUNCTION BYTE-LENGTH (DATA-BLOCK).
    IF NO-BYTES = 16711425 PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION BYTE-LENGTH (DATA-ARRAY (1)).
    IF NO-BYTES = 3342285 PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION BYTE-LENGTH (DATA-ELEMENTS (1, 1)).
    IF NO-BYTES = 51 PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION BYTE-LENGTH (FILL-A-BYTE (1, 1)).
    IF NO-BYTES = 15 PERFORM CORRECT-VALUE.
```

EXAMPLE 34. BYTE-LENGTH function

B.7. CHAR

The CHAR function returns a one-character alphanumeric value that is a character in the alphanumeric program collating sequence having the ordinal position equal to the value of *argument-1*. Since **ICOBOL** uses the ASCII collating sequence, these values are 1 through 256.

The type of this function is alphanumeric.

B.7.1 General format

FUNCTION CHAR (*argument-1*)

B.7.2 Arguments

1) *Argument-1* shall be an integer.

2) The value of *argument-1* shall be greater than zero and less than or equal to the number of positions in the alphanumeric program collating sequence.

B.7.3 Returned values

1) The returned value shall be the character in the alphanumeric program collating sequence having the ordinal position specified by *argument-1*.

B.7.4 Example

The following code fragments illustrate the use of this function.

```
01  D                     PIC S9(10) VALUE 100.
01  ARR                   VALUE "066037100070044".
    02 IND OCCURS 5 TIMES PIC 9(3).
01  WS-ANUM               PIC X.

    MOVE SPACE TO WS-ANUM.
    MOVE FUNCTION CHAR(37) TO WS-ANUM.
    IF WS-ANUM = "$" THEN
        PERFORM OK.

    MOVE SPACE TO WS-ANUM.
    MOVE FUNCTION CHAR(IND(5)) TO WS-ANUM.
    IF WS-ANUM = "+" THEN
        PERFORM OK.

    MOVE SPACE TO WS-ANUM.
    MOVE FUNCTION CHAR(D) TO WS-ANUM.
    IF WS-ANUM = "c" THEN
        PERFORM OK.
```

EXAMPLE 35. CHAR function

B.8. COS

The COS function returns a numeric value that approximates the cosine of an angle or arc, expressed in radians, that is specified by *argument-1*.

The type of this function is numeric.

B.8.1 General format

FUNCTION COS (*argument-1*)

B.8.2 Arguments

1) *Argument-1* shall be class numeric.

B.8.3 Returned values

1) The returned value is the approximation of the cosine of argument-1 and is greater than or equal to –1 and less than or equal to +1.

B.8.4 Example

The following code fragments illustrate the use of this function.

```
01  PI            PIC S9V9(17)    VALUE 3.141592654.
01  MINUSPI       PIC S9V9(17)    VALUE -3.141592654.
01  WS-NUM        PIC S9(5)V9(6).
01  MIN-RANGE     PIC S9(5)V9(7).
01  MAX-RANGE     PIC S9(5)V9(7).

    MOVE ZERO TO WS-NUM.
    MOVE -1.00000 TO MIN-RANGE.
    MOVE -0.999980 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION COS(MINUSPI).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                 PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    MOVE -0.000040 TO MIN-RANGE.
    MOVE  0.000040 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION COS(PI / 2).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                 PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    MOVE  0.999980 TO MIN-RANGE.
    MOVE  1.00000 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION COS(0).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                 PERFORM CORRECT-VALUE.
```

EXAMPLE 36. COS function

B.9. CURRENT-DATE

The CURRENT-DATE function returns a 21-character alphanumeric value that represents the calendar date, time of day, and local time differential factor provided by the system on which the function is evaluated.

The type of this function is alphanumeric.

B.9.1 General format

FUNCTION CURRENT-DATE

B.9.2 Returned values

1) The character positions returned, numbered from left to right, are:

Character Positions	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-16	Two numeric digits of the hundredths of a second past the second, in the range 00 through 99. The value 00 is returned if the system on which the function is evaluated does not have the facility to provide the fractional part of a second.
17	Either the character '–', the character '+', or the character '0'. The character '–' is returned if the local time indicated in the previous character positions is behind Coordinated Universal Time. The character '+' is returned if the local time indicated is the same as or ahead of Coordinated Universal Time. The character '0' is returned if the system on which this function is evaluated does not have the facility to provide the local time differential factor.
18-19	If character position 17 is '–', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the reported time is behind Coordinated Universal Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Coordinated Universal Time. If character position 17 is '0', the value 00 is returned.
20-21	Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the reported time is ahead of or behind Coordinated Universal Time, depending on whether character position 17 is '+' or '–', respectively. If character position 17 is '0', the value 00 is returned.

B.9.3 Example

The following code fragments illustrate the use of this function.

```
01  TODAYS-INFO.
    03  TODAYS-YEAR           PIC 9(4).
    03  TODAYS-MONTH          PIC 99.
    03  TODAYS-DAY            PIC 99.
    03  TODAYS-HOUR           PIC 99.
    03  TODAYS-MIN            PIC 99.
    03  TODAYS-SEC            PIC 99.
    03  TODAYS-HSEC           PIC 99.

    MOVE SPACES TO TODAYS-INFO.
    MOVE FUNCTION CURRENT-DATE TO TODAYS-INFO.
```

EXAMPLE 37. CURRENT-DATE function

B.10. DATE-OF-INTEGER

The DATE-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to standard date form (YYYYMMDD).

The type of this function is integer.

B.10.1 General format

FUNCTION DATE-OF-INTEGER (*argument-1*)

B.10.2 Arguments

1) *Argument-1* is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar. It shall not exceed the value of FUNCTION INTEGER-OF-DATE(99991231), which is 3,067,671.

B.10.3 Returned values

1) The returned value represents the ISO standard date equivalent of the integer specified in *argument-1*.

2) The returned value is in the form (YYYYMMDD) where YYYY represents a year in the Gregorian calendar; MM represents the month of that year; and DD represents the day of that month.

B.10.4 Example

The following code fragments illustrate the use of this function.

```
01  A                         PIC S9(10)     VALUE 400.
01  C                         PIC S9(10)     VALUE 300.
01  D                         PIC S9(10)     VALUE 1.
01  ARG1                      PIC S9(10)     VALUE 1.
01  ARR                                      VALUE "40537".
    02 IND OCCURS 5 TIMES     PIC 9.
01  TEMP                      PIC S9(5)V9(5).
01  WS-DATE                   PIC 9(8).

    MOVE ZERO TO WS-DATE.
    COMPUTE WS-DATE = FUNCTION DATE-OF-INTEGER(730).
    IF WS-DATE = 16021231 THEN
                 PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-DATE.
    COMPUTE WS-DATE = FUNCTION DATE-OF-INTEGER(1).
    IF WS-DATE = 16010101 THEN
                 PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-DATE.
    COMPUTE WS-DATE = FUNCTION DATE-OF-INTEGER(145655).
    IF WS-DATE = 19991016 THEN
                 PERFORM CORRECT-VALUE.
```

EXAMPLE 38. DATE-OF-INTEGER function

B.11. DATE-TO-YYYYMMDD

The DATE-TO-YYYYMMDD function converts *argument-1* from the form YYmmdd to the form YYYYmmdd. *Argument-2*, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding window, into which the year of *argument-1* falls.

The type of the function is integer.

B.11.1 General format

FUNCTION DATE-TO-YYYYMMDD (*argument-1* [, *argument-2*])

B.11.2 Arguments

1) *Argument-1* shall be zero or a positive integer less than 1000000.

NOTE — This function does not check *argument-1* to ensure that it is a valid date. The returned value can be an argument to the TEST-DATE-YYYYMMDD function to check its validity.

2) *Argument-2* shall be an integer.

3) If *argument-2* is omitted, the function shall be evaluated as though 50 were specified.

4) The sum of the year at the time of execution and the value of *argument-2* shall be less than 10000 and greater than 1699.

B.11.3 Returned values

1) The equivalent arithmetic expression shall be as follows:

(FUNCTION YEAR-TO-YYYY (YY, *argument-2*) * 10000 + mmdd)

where

YY = FUNCTION INTEGER (*argument-1*/10000)
mmdd = FUNCTION MOD (*argument-1*, 10000)

and where *argument-1* of the INTEGER and MOD functions and *argument-2* of the YEAR-TO-YYYY function are the same as *argument-1* and *argument-2* of the DATE-TO-YYYYMMDD function itself.

NOTES

1 — In the year 2002 the returned value for FUNCTION DATE-TO-YYYYMMDD (851003, 10) is 19851003. In the year 1994 the returned value for FUNCTION DATE-TO-YYYYMMDD (981002, (–10)) is 18981002.

2 — This function supports a sliding window algorithm. See the notes for the YEAR-TO-YYYY function for a discussion of how to specify a fixed window.

B.11.4 Example

The following code fragments illustrate the use of this function.

```
77  WS-DATE             PIC 9(8).

    COMPUTE WS-DATE = FUNCTION DATE-TO-YYYYMMDD (700615, 5).
    IF WS-DATE = 19700615 THEN
                 PERFORM CORRECT-VALUE.

    COMPUTE WS-DATE = FUNCTION DATE-TO-YYYYMMDD (490615).
    IF WS-DATE = 20490615 THEN
                 PERFORM CORRECT-VALUE.

    COMPUTE WS-DATE = FUNCTION DATE-TO-YYYYMMDD (040615, -10).
    IF WS-DATE = 19040615 THEN
                 PERFORM CORRECT-VALUE.
```

EXAMPLE 39. DATE-TO-YYYYMMDD function

B.12. DAY-OF-INTEGER

The DAY-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to Julian date form (YYYYDDD).

The type of this function is integer.

B.12.1 General format

FUNCTION DAY-OF-INTEGER (*argument-1*)

B.12.2 Arguments

1) *Argument-1* is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar. It shall not exceed the value of FUNCTION INTEGER-OF-DATE(99991231).

B.12.3 Returned values

1) The returned value represents the Julian equivalent of the integer specified in *argument-1*.

2) The returned value is an integer of the form (YYYYDDD) where YYYY represents a year in the Gregorian calendar and DDD represents the day of that year.

B.12.4 Example

The following code fragments illustrate the use of this function.

```
77  WS-DATE    PIC 9(7).
77  A          PIC S9(10)     VALUE 400.

    COMPUTE WS-DATE = FUNCTION DAY-OF-INTEGER (145732).
    IF WS-DATE = 2000001 THEN
                 PERFORM CORRECT-VALUE.

    EVALUATE FUNCTION DAY-OF-INTEGER(A)
    WHEN     1602035
                    PERFORM CORRECT-VALUE.

    COMPUTE WS-DATE = FUNCTION DAY-OF-INTEGER(1).
    IF WS-DATE = 1601001 THEN
                    PERFORM CORRECT-VALUE.
```

EXAMPLE 40. DAY-OF-INTEGER function

B.13. DAY-TO-YYYYDDD

The DAY-TO-YYYYDDD function converts *argument-1* from the form YYnnn to the form YYYYnnn. *Argument-2*, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding window, into which the year of *argument-1* falls.

The type of the function is integer.

B.13.1 General format

FUNCTION DAY-TO-YYYYDDD (*argument-1* [, *argument-2*])

B.13.2 Arguments

1) *Argument-1* shall be zero or a positive integer less than 100000.

   Note — This function does not check *argument-1* to ensure that it is a valid date. The returned value can be an argument to the TEST-DAY-YYYYDDD function to check its validity.

2) *Argument-2* shall be an integer.

3) If *argument-2* is omitted, the function shall be evaluated as though 50 were specified.

4) The sum of the year at the time of execution and the value of *argument-2* shall be less than 10000 and greater than 1699.

B.13.3 Returned values

1) The equivalent arithmetic expression shall be as follows:

   (FUNCTION YEAR-TO-YYYY (YY, *argument-2*) * 1000 + nnn)

   where

   YY = FUNCTION INTEGER (*argument-1*/1000)
   nnn = FUNCTION MOD (*argument-1*, 1000)

   and where *argument-1* of the INTEGER and MOD functions and *argument-2* of the YEAR-TO-YYYY function are the same as *argument-1* and *argument-2* of the DAY-TO-YYYYDDD function itself.

   NOTES

   1) In the year 2002 the returned value for FUNCTION DAY-TO-YYYYDDD (10004, 20) is 2010004. In the year 2013 the returned value for FUNCTION DAY-TO-YYYYDDD (95005, (–10)) is 1995005.

   2) This function supports a sliding window algorithm. See the notes for the YEAR-TO-YYYY function for a discussion of how to specify a fixed window.

B.13.4 Example

The following code fragments illustrate the use of this function.

```
01 YEARVAL     PIC 9(7).

DISPLAY "FUNCTION DAY-TO-YYYYDDD " NO ADVANCING.
COMPUTE YEARVAL = FUNCTION DAY-TO-YYYYDDD (10004, 20)
IF YEARVAL = 2010004
             PERFORM CORRECT-VALUE.

COMPUTE YEARVAL = FUNCTION DAY-TO-YYYY ( 85005, (-10))
IF YEARVAL = 1985005
             PERFORM CORRECT-VALUE.
```

EXAMPLE 41. DAY-TO-YYYYDDD function

B.14. E

The E function returns an approximation of e, the base of natural logarithms.

The type of the function is numeric.

B.14.1 General format

FUNCTION E

B.14.2 Returned values

1) The equivalent arithmetic expression shall be

(2 + .718281828459045235)

B.14.3 Example

The following code fragments illustrate the use of this function.

```
77  A           PIC S9V9(17).

    COMPUTE A = FUNCTION E.
    IF A = 2.71828182845904523
           PERFORM CORRECT-VALUE.
```

EXAMPLE 42. E function

B.15. EXP

The EXP function returns an approximation of the value of *e* raised to the power of the argument.

The type of the function is numeric.

B.15.1 General format

FUNCTION EXP (*argument-1*)

B.15.2 Arguments

1) *Argument-1* shall be class numeric.

B.15.3 Returned values

1) The equivalent arithmetic expression shall be:

(FUNCTION E ** *argument-1*)

B.15.4 Example

The following code fragments illustrate the use of this function.

```
77  A           PIC S999      VALUE -5.
77  RESULT      PIC S9(9)V9(9).

    COMPUTE RESULT = FUNCTION EXP (3).
    IF RESULT = 20.085536923
                PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION EXP (A).
    IF RESULT = 0.006737946
                PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION EXP (14.5)
    IF RESULT = 1982759.263537568
                PERFORM CORRECT-VALUE.
```

EXAMPLE 43. EXP function

B.16. EXP10

The EXP10 function returns an approximation of the value of 10 raised to the power of the argument.

The type of the function is numeric.

B.16.1 General format

FUNCTION EXP10 (*argument-1*)

B.16.2 Arguments

1) *Argument-1* shall be class numeric.

B.16.3 Returned values

1) The equivalent arithmetic expression shall be:

(10 ** *argument-1*)

B.16.4 Example

The following code fragments illustrate the use of this function.

```
77  A            PIC S999     VALUE -5.
77  RESULT       PIC S9(9)V9(9).
77  RESULTA      PIC S9(18).

    COMPUTE RESULT = FUNCTION EXP10 (3).
    IF RESULT = 1000.00000
                PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION EXP10 (A).
    IF RESULT = 0.000010000
                PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION EXP10 (14.5).
    IF RESULTA = 316227766016837
                 PERFORM CORRECT-VALUE.
```

EXAMPLE 44. EXP10 function

B.17. FACTORIAL

The FACTORIAL function returns an integer that is the factorial of *argument-1*.

The type of this function is integer.

B.17.1 General format

FUNCTION FACTORIAL (*argument-1*)

B.17.2 Arguments

1) *Argument-1* shall be an integer greater than or equal to zero.

B.17.3 Returned values

1) The equivalent arithmetic expression shall be as follows:

   a) When the value of *argument-1* is 0 or 1,

      (1)

   b) When the value of *argument-1* is 2,

      (2)

   c) When the value of *argument-1* is n,

      (n * (n – 1) * (n – 2) * ... * 1)

B.17.4 Example

The following code fragments illustrate the use of this function.

```
01  B                  PIC 9(10) VALUE 2.
01  ARR                 VALUE "40537".
    02  IND OCCURS 5 TIMES PIC 9.
01  RESULT                 PIC 9(18).

    COMPUTE RESULT = FUNCTION FACTORIAL (1).
    IF RESULT = 1 PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION FACTORIAL (IND (B)).
    IF RESULT = 1 PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION FACTORIAL (B + 7).
    IF RESULT = 362880 PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION FACTORIAL (B).
    IF RESULT = 2 PERFORM CORRECT-VALUE.
```

EXAMPLE 45. FACTORIAL function

B.18. FRACTION-PART

The FRACTION-PART function returns a numeric value that is the fraction portion of the argument.

The type of the function is numeric.

B.18.1 General format

FUNCTION FRACTION-PART (*argument-1*)

B.18.2 Arguments

1) *Argument-1* shall be of the class numeric.

B.18.3 Returned values

1) The equivalent arithmetic expression shall be:

   (*argument-1* – FUNCTION INTEGER-PART (*argument-1*))

   where the argument for the INTEGER-PART function is the same as for the FRACTION-PART function itself.

   NOTE — If the value of *argument-1* is +1.5, +0.5 is returned. If the value of *argument-1* is –1.5, –0.5 is returned.

B.18.4 Example

The following code fragments illustrate the use of this function.

```
77  WS-FRACTION         PIC -99.999

  COMPUTE WS-FRACTION = FUNCTION FRACTION-PART(6.3 - (4.2 / 2)).
  IF WS-FRACTION = .2
         PERFORM CORRECT-VALUE.

   COMPUTE WS-FRACTION =
     FUNCTION FRACTION-PART(1.35) - FUNCTION FRACTION-PART(2.85).
   IF WS-FRACTION = -.5
     PERFORM CORRECT-VALUE.

   EVALUATE FUNCTION FRACTION-PART (123.7890675)
   WHEN    .7890675
     PERFORM CORRECT-VALUE.
```

EXAMPLE 46. FRACTION-PART function

B.19. HIGHEST-ALGEBRAIC

The HIGHEST-ALGEBRAIC function returns a value that is equal to the greatest algebraic value that may be represented in *argument-1*.

The type of this function depends upon the argument types as follows:

Argument type	Function type
Integer	Integer
Numeric	Numeric
Numeric-edited	Numeric

B.19.1 General format

FUNCTION HIGHEST-ALGEBRAIC (*argument-1*)

B.19.2 Arguments

1) *Arguments-1* shall be an elementary data item of category numeric or numeric-edited.

B.19.3 Returned values

1) The value returned is equal to the positive algebraic value of greatest magnitude that may be represented in *argument-1*.

NOTE — The following illustrates the expected results for some values of *argument-1*.

Argument-1 characteristics	**Value returned**
S999	+999
S9(4) BINARY	+9999
99V9(3)	+99.999
$**,**9.99BCR	+99999.99
$**,**9.99	+99999.99

B.19.4 Example

The following code fragments illustrate the use of this function.

```
77  A                   PIC S9(5)V9(3).
77  B                   PIC S9(16)  COMP.
77  C                   PIC $$$$$$$9.99.
77  RESULT              PIC 9(18).

    COMPUTE RESULT = FUNCTION HIGHEST-ALGEBRAIC (A).
    IF RESULT = 99999
                PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION HIGHEST-ALGEBRAIC (B).
    IF RESULT = 36028797018963967
                PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION HIGHEST-ALGEBRAIC (C).
    IF RESULT = 9999999
                PERFORM CORRECT-VALUE.
```

EXAMPLE 47. HIGHEST-ALGEBRAIC function

B.20. INTEGER

The INTEGER function returns the greatest integer value that is less than or equal to the argument.

The type of this function is integer.

B.20.1 General format

FUNCTION INTEGER (*argument-1*)

B.20.2 Arguments

1) *Argument-1* shall be class numeric.

B.20.3 Returned values

1) *Argument-1* is not rounded.

2) The returned value is the greatest integer less than or equal to the value of *argument-1*

> **NOTE** — For example, if the value of *argument-1* is –1.5, –2 is returned. If the value of *argument-1* is +1.5, +1 is returned.

B.20.4 Example

The following code fragments illustrate the use of this function.

```
77  WS-INT              PIC S9(10).

    COMPUTE WS-INT = FUNCTION INTEGER (-9.763).
    IF WS-INT = -10
                PERFORM CORRECT-VALUE.

    COMPUTE WS-INT = FUNCTION INTEGER (230492.4828).
    IF WS-INT = 230492
                PERFORM CORRECT-VALUE.

    COMPUTE WS-INT = FUNCTION INTEGER (0.00032).
    IF WS-INT = 0
                PERFORM CORRECT-VALUE.
```

EXAMPLE 48. INTEGER function

B.21. INTEGER-OF-DATE

The INTEGER-OF-DATE function converts a date in the Gregorian calendar from standard date form (YYYYMMDD) to integer date form.

The type of this function is integer.

B.21.1 General format

FUNCTION INTEGER-OF-DATE (*argument-1*)

B.21.2 Arguments

1) *Argument-1* shall be an integer of the form YYYYMMDD, whose value is obtained from the calculation (YYYY * 10,000) + (MM * 100) + DD.

a) YYYY represents the year in the Gregorian calendar. It shall be an integer greater than 1600 and less than 10000.

b) MM represents a month and shall be a positive integer less than 13.

c) DD represents a day and shall be a positive integer less than 32 provided that it is valid for the specified month and year combination.

B.21.3 Returned values

1) The returned value is an integer that is the number of days the date represented by *argument-1* succeeds December 31, 1600, in the Gregorian calendar.

B.21.4 Example

The following code fragments illustrate the use of this function.

```
77  WS-INT              PIC 9(10).

    COMPUTE WS-INT = FUNCTION INTEGER-OF-DATE (20000101).
    IF WS-INT = 145732   PERFORM CORRECT-VALUE.

    COMPUTE WS-INT = FUNCTION INTEGER-OF-DATE (16010101).
    IF WS-INT = 1        PERFORM CORRECT-VALUE.

    MOVE 16010101 TO WS-INT.
    PERFORM DATEOFINT-TEST
            UNTIL FUNCTION INTEGER-OF-DATE(WS-INT) > 10.
    IF WS-INT = 16010111 PERFORM CORRECT-VALUE.

DATEOFINT-TEST.
    COMPUTE WS-INT = WS-INT + 1.
```

EXAMPLE 49. INTEGER-OF-DATE function

B.22. INTEGER-OF-DAY

The INTEGER-OF-DAY function converts a date in the Gregorian calendar from Julian date form (YYYYDDD) to integer date form.

The type of this function is integer.

B.22.1 General format

FUNCTION INTEGER-OF-DAY (*argument-1*)

B.22.2 Arguments

1) *Argument-1* shall be an integer of the form YYYYDDD, whose value is obtained from the calculation (YYYY * 1000) + DDD.

a) YYYY represents the year in the Gregorian calendar. It shall be an integer greater than 1600 and less than 10000.

b) DDD represents the day of the year. It shall be a positive integer less than 367 provided that it is valid for the year specified.

B.22.3 Returned values

1) The returned value is an integer that is the number of days the date represented by *argument-1* succeeds December 31, 1600, in the Gregorian calendar.

B.22.4 Example

The following code fragments illustrate the use of this function.

```
01  A                       PIC S9(10)  VALUE 1602035.
01  ARR                                 VALUE "16010011602035".
    02 IND OCCURS 2 TIMES   PIC 9(7).
01  WS-INT                 PIC 9(10).
    EVALUATE FUNCTION INTEGER-OF-DAY(A)
    WHEN     400
             PERFORM CORRECT-VALUE.

    IF FUNCTION INTEGER-OF-DAY(IND(1)) = 1 THEN
                      PERFORM CORRECT-VALUE.

    COMPUTE WS-INT = FUNCTION INTEGER-OF-DAY(A) + 10.
    IF WS-INT = 410 THEN
                PERFORM CORRECT-VALUE.
```

EXAMPLE 50. INTEGER-OF-DAY function

B.23. INTEGER-PART

The INTEGER-PART function returns an integer that is the integer portion of *argument-1*.

The type of this function is integer.

B.23.1 General format

FUNCTION INTEGER-PART (*argument-1*)

B.23.2 Arguments

1) *Argument-1* shall be class numeric.

B.23.3 Returned values

1) The equivalent arithmetic expression shall be:

(FUNCTION SIGN (*argument-1*) * FUNCTION INTEGER (FUNCTION ABS (*argument-1)*))

where the arguments for the SIGN and ABS functions are the same as for the INTEGER-PART function itself.

NOTE — If the value of *argument-1* is +1.5, +1 is returned; if the value of *argument-1* is –1.5, –1 is returned; and if the value of *argument-1* is zero, zero is returned.

B.23.4 Example

The following code fragments illustrate the use of this function.

```
77  WS-INTEGER      PIC -99.999

     COMPUTE WS-INTEGER =
        FUNCTION INTEGER-PART(6.3 - (4.2 / 2)).

     IF WS-INTEGER = 4 PERFORM CORRECT-VALUE.

     COMPUTE WS-INTEGER =
        FUNCTION INTEGER-PART(1.35) -
           FUNCTION INTEGER-PART(2.85).

     IF WS-INTEGER = 0 PERFORM CORRECT-VALUE.

     EVALUATE FUNCTION INTEGER-PART (123.7890675)
     WHEN    123
             PERFORM CORRECT-VALUE.
```

EXAMPLE 51. INTEGER-PART function

B.24. LENGTH

The LENGTH function returns an integer equal to the length of the argument in alphanumeric character positions.

The type of this function is integer.

B.24.1 General format

FUNCTION LENGTH (*argument-1*)

B.24.2 Arguments

1) *Argument-1* shall be alphanumeric literal or a data item of any class or category.

B.24.3 Returned values

1) If *argument-1* is an elementary data item or an alphanumeric literal, the returned value shall be an integer equal to the length of *argument-1* in alphanumeric character positions.

2) If *argument-1* is a group data item:

a) If *argument-1* or any data item subordinate to *argument-1* is described with the DEPENDING phrase of the OCCURS clause, the returned value shall be an integer equal to the length of *argument-1* in alphanumeric character positions, as a sending operand, determined by evaluation of the data item specified in the DEPENDING phrase in accordance with the rules of the OCCURS clause. The contents of the data item specified in the DEPENDING phrase are used at the time the LENGTH function is evaluated.

b) Otherwise, the returned value shall be an integer equal to the length of *argument-1* in alphanumeric character positions.

c) The returned length shall include the number of implicit FILLER positions, if any, in *argument-1*.

B.24.4 Example

The following code fragments illustrate the use of this function.

```
01  DATA-BLOCK.
    03  DATA-ARRAY                         OCCURS 1 TO 5 TIMES
                                           DEPENDING ON I.
        05  DATA-ELEMENTS                  OCCURS 65535 TIMES.
            07  DATA-COUNTER  PIC 9(18).
            07  ARRAY-VALUE   PIC 9(18).
            07  FILL-A-BYTE   PIC X(15).
01  A                          PIC S999              VALUE -999.
01  B                          PIC -999.99.
01  C                          PIC 9(9) COMP.
01  I                          PIC 99                VALUE 3.
01  NO-BYTES                   PIC 9(10).




    COMPUTE NO-BYTES = FUNCTION LENGTH (A).
    IF NO-BYTES = 3
          PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION LENGTH (B).
    IF NO-BYTES = 7
          PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION LENGTH (C).
    IF NO-BYTES = 4
          PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION LENGTH (DATA-BLOCK).
    IF NO-BYTES = 10026855
          PERFORM CORRECT-VALUE.

    MOVE 5 TO I.
    COMPUTE NO-BYTES = FUNCTION LENGTH (DATA-BLOCK).
    IF NO-BYTES = 16711425
          PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION LENGTH (DATA-ARRAY (1)).
    IF NO-BYTES = 3342285
          PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION LENGTH (DATA-ELEMENTS (1, 1)).
    IF NO-BYTES = 51
          PERFORM CORRECT-VALUE.

    COMPUTE NO-BYTES = FUNCTION LENGTH (FILL-A-BYTE (1, 1)).
    IF NO-BYTES = 15
          PERFORM CORRECT-VALUE.
```

EXAMPLE 52. LENGTH function

B.25. LOG

The LOG function returns a numeric value that approximates the logarithm to the base *e* (natural log) of *argument-1*.

The type of this function is numeric.

B.25.1 General format

FUNCTION LOG (*argument-1*)

B.25.2 Arguments

1) *Argument-1* shall be class numeric.

2) The value of *argument-1* shall be greater than zero.

B.25.3 Returned values

1) The returned value is the approximation of the logarithm to the base *e* of *argument-1*.

B.25.4 Example

The following code fragments illustrate the use of this function.

```
01  E                   PIC S9(1)V9(9) VALUE 2.718281828.
01  WS-NUM             PIC S9(5)V9(6).
01  MIN-RANGE          PIC S9(5)V9(7).
01  MAX-RANGE          PIC S9(5)V9(7).



    MOVE ZERO TO WS-NUM.
    MOVE  0.999980 TO MIN-RANGE.
    MOVE  1.00002 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION LOG(E).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                  PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    MOVE  0.632497 TO MIN-RANGE.
    MOVE  0.632547 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION LOG(3.2 / 1.7).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                  PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    MOVE  1.48569 TO MIN-RANGE.
    MOVE  1.48581 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION LOG(E + 1.7).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                  PERFORM CORRECT-VALUE.
```

EXAMPLE 53. LOG function

B.26. LOG10

The LOG10 function returns a numeric value that approximates the logarithm to the base 10 of *argument-1*.

The type of this function is numeric.

B.26.1 General format

FUNCTION LOG10 (*argument-1*)

B.26.2 Arguments

1) *Argument-1* shall be class numeric.

2) The value of *argument-1* shall be greater than zero.

B.26.3 Returned values

1) The returned value is the approximation of the logarithm to the base 10 of *argument-1*.

B.26.4 Example

The following code fragments illustrate the use of this function.

```
01  ARG1                    PIC S9(5)V9(5)      VALUE 10.00.
01  WS-NUM                  PIC S9(5)V9(6).
01  MIN-RANGE               PIC S9(5)V9(7).
01  MAX-RANGE               PIC S9(5)V9(7).



    MOVE ZERO TO WS-NUM.
    MOVE -0.000020 TO MIN-RANGE.
    MOVE  0.000020 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION LOG10(1).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
           PERFORM CORRECT-VALUE.

    MOVE -2.00004 TO MIN-RANGE.
    MOVE -1.99996 TO MAX-RANGE.
    IF (FUNCTION LOG10(.01) >= MIN-RANGE) AND
       (FUNCTION LOG10(.01) <= MAX-RANGE) THEN
           PERFORM CORRECT-VALUE.

    PERFORM LOG10-TEST
      UNTIL FUNCTION LOG10(ARG1) < 0.30.
    PERFORM CORRECT-VALUE.

LOG10-TEST.
    COMPUTE ARG1 = ARG1 - 1.00.
```

EXAMPLE 54. LOG10 function

B.27. LOWER-CASE

The LOWER-CASE function returns a character string that is the same length as *argument-1* with each uppercase letter replaced by the corresponding lowercase letter.

The type of the function depends on the argument type as follows:

Argument Type	Function Type
Alphabetic	Alphabetic
Alphanumeric	Alphanumeric

B.27.1 General format

FUNCTION LOWER-CASE (*argument-1*)

B.27.2 Arguments

1) *Argument-1* shall be class alphabetic or alphanumeric and shall be at least one character position in length.

B.27.3 Returned values

1) The same character string as *argument-1* is returned, except that each uppercase letter shall be replaced by the corresponding lowercase letter.

2) The character string returned has the same length as *argument-1*.

B.27.4 Example

The following code fragments illustrate the use of this function.

```
01  WS-ANUM             PIC X(10).



    MOVE SPACES TO WS-ANUM.
    IF FUNCTION LOWER-CASE("highnLOW") = "highnlow" THEN
           PERFORM CORRECT-VALUE.

    MOVE FUNCTION LOWER-CASE("figure") TO WS-ANUM.
    IF WS-ANUM = "figure" THEN
           PERFORM CORRECT-VALUE.

    MOVE SPACES TO WS-ANUM.
    MOVE FUNCTION LOWER-CASE("95") TO WS-ANUM.
    IF WS-ANUM = "95" THEN
           PERFORM CORRECT-VALUE.
```

EXAMPLE 55. LOWER-CASE function

B.28. LOWEST-ALGEBRAIC

The LOWEST-ALGEBRAIC function returns a value that is equal to the lowest algebraic value that may be represented in *argument-1*.

The type of this function depends upon the argument types as follows:

Argument type	Function type
Integer	Integer
Numeric	Numeric
Numeric-edited	Numeric

B.28.1 General format

FUNCTION LOWEST-ALGEBRAIC (*argument-1*)

B.28.2 Arguments

1) *Argument-1* shall be an elementary data item of category numeric or numeric-edited.

B.28.3 Returned values

1) The value returned is equal to the lowest algebraic value that may be represented in *argument-1*.

NOTE — The following illustrates the expected results for some values of *argument-1*.

Argument-1 characteristics	Value returned
S999	–999
S9(4) BINARY	–9999
99V9(3)	0
$**,**9.99BCR	–99999.99
$**,**9.99	0

B.28.4 Example

The following code fragments illustrate the use of this function.

```
77  A             PIC S9(5)V9(3).
77  B             PIC S9(7)  COMP.
77  C             PIC $$$$$$$9.99.
77  RESULT        PIC S9(18).



    COMPUTE RESULT = FUNCTION LOWEST-ALGEBRAIC (A).
    IF RESULT = -99999      PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION LOWEST-ALGEBRAIC (B).
    IF RESULT = -2147483648 PERFORM CORRECT-VALUE.

    COMPUTE RESULT = FUNCTION LOWEST-ALGEBRAIC (C).
    IF RESULT = 0           PERFORM CORRECT-VALUE.
```

EXAMPLE 56. LOWEST-ALGEBRAIC function

B.29. MAX

The MAX function returns the content of the *argument-1* that contains the maximum value.

The type of this function depends upon the argument types as follows:

Argument type	Function type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
Index	Index
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

B.29.1 General format

FUNCTION MAX ({ *argument-1* }...)

B.29.2 Arguments

1) *Argument-1* shall not be of class pointer.

2) All arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

B.29.3 Returned values

1) The returned value is the content of the *argument-1* having the greatest value. The comparisons used to determine the greatest value are made according to the rules for simple conditions. (See Page 233, Simple conditions.)

2) If the value of more than one *argument-1* is equal to the greatest value, the content of the *argument-1* returned is the leftmost *argument-1* having that value.

3) If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected *argument-1*.

B.29.4 Example

The following code fragments illustrate the use of this function.

```
01  WS-NUM                     PIC S9(6)V9(6).
01  WS-ANUM                    PIC X.



    MOVE ZERO TO WS-NUM.
    COMPUTE WS-NUM = FUNCTION MAX(-4.3, 10.2, -0.7, 3.9).
    IF (WS-NUM >= 10.1998) AND (WS-NUM <= 10.2002)
             PERFORM CORRECT-VALUE.

    MOVE SPACES TO WS-ANUM.
    MOVE FUNCTION MAX("R", "I", "I", "a") TO WS-ANUM.
    IF WS-ANUM = "a" THEN
             PERFORM CORRECT-VALUE.

    COMPUTE WS-NUM = FUNCTION MAX(A * B, (C + 1) / 2, 3 + 4).
    IF (WS-NUM >= MIN-RANGE) AND (WS-NUM <= MAX-RANGE) THEN
             PERFORM CORRECT-VALUE.
```

EXAMPLE 57. MAX function

B.30. MEAN

The MEAN function returns a numeric value that is the arithmetic mean (average) of its arguments.

The type of this function is numeric.

B.30.1 General format

FUNCTION MEAN ({ *argument-1* }...)

B.30.2 Arguments

1) *Argument-1* shall be class numeric.

B.30.3 Returned values

1) The equivalent arithmetic expression shall be as follows:

a) For one occurrence of *argument-1*,

(*argument-1*)

b) For two occurrences of argument-1,

((*argument-1* $_1$ + *argument-1* $_2$) / 2)

c) For n occurrences of *argument-1*,

((*argument-1* $_1$ + *argument-1* $_2$ +... + *argument-1* $_n$) / n)

B.30.4 Example

The following code fragments illustrate the use of this function.

```
01  A                       PIC S9(10)            VALUE 5.
01  B                       PIC S9(10)            VALUE 7.
01  C                       PIC S9(5)V9(5)        VALUE 34.26.
01  WS-NUM                 PIC S9(6)V9(6).
01  MIN-RANGE              PIC S9(5)V9(7).
01  MAX-RANGE              PIC S9(5)V9(7).




    EVALUATE FUNCTION MEAN(3.9, -0.3, 8.7, 100.2)
    WHEN 28.1244 THRU 28.1256
                  PERFORM CORRECT-VALUE.
    MOVE ZERO TO WS-NUM.
    MOVE  20.6896 TO MIN-RANGE.
    MOVE  20.6904 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION MEAN(C, 9 * A, 0, B / 2).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                  PERFORM CORRECT-VALUE.
    MOVE ZERO TO WS-NUM.
    MOVE 4.49991 TO MIN-RANGE.
    MOVE 4.50009 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION MEAN(FUNCTION MEAN(4, 2), 6).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                  PERFORM CORRECT-VALUE.
```

EXAMPLE 58. MEAN function

B.31. MEDIAN

The MEDIAN function returns the content of the argument whose value is the middle value in the list formed by arranging the arguments in sorted order.

The type of this function is numeric.

B.31.1 General format

FUNCTION MEDIAN ({ *argument-1* }...)

B.31.2 Arguments

1) *Argument-1* shall be class numeric.

B.31.3 Returned values

1) When the number of occurrences of *argument-1* is odd, the returned value shall be such that at least half of the occurrences referenced by *argument-1* are greater than or equal to the returned value and at least half are less than or equal. For the purposes of the equivalent arithmetic expression, the middle value is referred to as argument-a.

The equivalent arithmetic expression shall be

(argument-a)

2) When the number of occurrences of *argument-1* is even, the returned value is the arithmetic mean of the two middle values. For the purposes of the equivalent arithmetic expression, the two middle values are referred to as argument-b and argument-c.

The equivalent arithmetic expression shall be

((argument-b + argument-c) / 2)

3) The comparisons used to arrange the *argument-1* values in sorted order are made according to the rules for simple conditions. (See Page 233, Simple conditions.)

B.31.4 Example

The following code fragments illustrate the use of this function.

```
01  A           PIC S9(10)            VALUE 5.
01  B           PIC S9(10)            VALUE 7.
01  C           PIC S9(5)V9(5)        VALUE 34.26.
01  WS-NUM      PIC S9(6)V9(7).
01  MIN-RANGE   PIC S9(5)V9(7).
01  MAX-RANGE   PIC S9(5)V9(7).



    EVALUATE FUNCTION MEDIAN(3.9, -0.3, 8.7, 100.2)
    WHEN 6.29987 THRU 6.30013
                  PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    MOVE  34.2593 TO MIN-RANGE.
    MOVE  34.2607 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION MEDIAN(C, 9 * A, B / 2).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                  PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    COMPUTE WS-NUM = FUNCTION MEDIAN(10.2, -0.2, 5.6, -15.).
    IF (WS-NUM >= 2.69995) AND
       (WS-NUM <= 2.70005)
                  PERFORM CORRECT-VALUE.
```

EXAMPLE 59. MEDIAN function

B.32. MIDRANGE

The MIDRANGE (middle range) function returns a numeric value that is the arithmetic mean (average) of the values of the minimum argument and the maximum argument.

The type of this function is numeric.

B.32.1 General format

FUNCTION MIDRANGE ({ *argument-1* }...)

B.32.2 Arguments

1) *Argument-1* shall be class numeric.

B.32.3 Returned values

1) The equivalent arithmetic expression shall be

   ((FUNCTION MAX (*argument-1*) + FUNCTION MIN (*argument-1)*) / 2)

   where the arguments for the MAX and MIN functions are the same as the arguments for the MIDRANGE function itself.

B.32.4 Example

The following code fragments illustrate the use of this function.

```
01  A           PIC S9(10)             VALUE 5.
01  B           PIC S9(10)             VALUE 7.
01  C           PIC S9(5)V9(5)         VALUE 34.26.
01  WS-NUM     PIC S9(6)V9(7).
01  MIN-RANGE  PIC S9(5)V9(7).
01  MAX-RANGE  PIC S9(5)V9(7).



    EVALUATE FUNCTION MIDRANGE(3.9, -0.3, 8.7, 100.2)
    WHEN 49.9490 THRU 49.9510
                  PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    MOVE  22.4995   TO MIN-RANGE.
    MOVE  22.5004 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION MIDRANGE(C, 9 * A, 0, B / 2).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                  PERFORM CORRECT-VALUE.
    MOVE ZERO TO WS-NUM.
    MOVE 3.49993 TO MIN-RANGE.
    MOVE 3.50007 TO MAX-RANGE.
    COMPUTE WS-NUM =
            FUNCTION MIDRANGE(FUNCTION MIDRANGE(1, 3), 5).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                  PERFORM CORRECT-VALUE.
```

EXAMPLE 60. MIDRANGE function

B.33. MIN

The MIN function returns the content of the *argument-1* that contains the minimum value.

The type of this function depends upon the argument types as follows:

Argument Type	Function type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric
Index	Index
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

B.33.1 General format

FUNCTION MIN ({ *argument-1* }...)

B.33.2 Arguments

1) *Argument-1* shall not be of class pointer.

2) All arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

B.33.3 Returned values

1) The returned value is the content of the *argument-1* having the least value. The comparisons used to determine the least value are made according to the rules for simple conditions. (See Page 233, Simple conditions.)

2) If the value of more than one *argument-1* is equal to the least value, the content of the *argument-1* returned is the leftmost *argument-1* having that value.

3) If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected *argument-1*.

B.33.4 Example

The following code fragments illustrate the use of this function.

```
01  A          PIC S9(10)     VALUE 5.
01  B          PIC S9(10)     VALUE 7.
01  C          PIC S9(5)V9(5) VALUE 34.26.
01  WS-NUM     PIC S9(5)V9(6).
01  MIN-RANGE  PIC S9(5)V9(7).
01  MAX-RANGE  PIC S9(5)V9(7).



    IF (FUNCTION MIN(4.3, 2.6, 7.3, 9.1) >= 2.59995) AND
       (FUNCTION MIN(4.3, 2.6, 7.3, 9.1) <= 2.60005) THEN
                    PERFORM CORRECT-VALUE.
    MOVE ZERO TO WS-NUM.
    MOVE  1.99996 TO MIN-RANGE.
    MOVE  2.00004 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION MIN(A * B, (3 + 1) / 2, 3 + 4).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                    PERFORM CORRECT-VALUE.
    MOVE ZERO TO WS-NUM.
    MOVE 4.99990 TO MIN-RANGE.
    MOVE 5.00010 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION MIN(FUNCTION MIN(14, A), E, 50).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                    PERFORM CORRECT-VALUE.
```

EXAMPLE 61. MIN function

B.34. MOD

The MOD function returns an integer value that is *argument-1* modulo *argument-2*.

The type of this function is integer.

B.34.1 General format

FUNCTION MOD (*argument-1*, *argument-2*)

B.34.2 Arguments

1) *Argument-1* and *argument-2* shall be integers.

2) The value of *argument-2* shall not be zero.

B.34.3 Returned values

1) The equivalent arithmetic expression shall be

(*argument-1* – (*argument-2* * FUNCTION INTEGER (*argument-1* / *argument-2*)))

where *argument-1* and *argument-2* for the INTEGER function are the same as the arguments for the MOD function itself.

NOTE — The following illustrates the expected results for some values of *argument-1* and *argument-2*.

Argument-1	Argument-2	Return
11	5	1
–11	5	4
11	–5	–4
–11	–5	–1

B.34.4 Example

The following code fragments illustrate the use of this function.

```
01  A           PIC S9(10)     VALUE 5.
01  B           PIC S9(10)     VALUE 7.
01  WS-NUM      PIC S9(5)V9(6).
01  MIN-RANGE   PIC S9(5)V9(7).
01  MAX-RANGE   PIC S9(5)V9(7).



    EVALUATE FUNCTION MOD(11, 5)
    WHEN 1          PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    COMPUTE WS-NUM = FUNCTION MOD(-11, 5).
    IF WS-NUM = 4 THEN
                    PERFORM CORRECT-VALUE.

    MOVE ZERO TO WS-NUM.
    MOVE  6.99986 TO MIN-RANGE.
    MOVE  7.00014 TO MAX-RANGE.
    COMPUTE WS-NUM = FUNCTION MOD(FUNCTION INTEGER(A - B), 9).
    IF (WS-NUM >= MIN-RANGE) AND
       (WS-NUM <= MAX-RANGE) THEN
                    PERFORM CORRECT-VALUE.
```

EXAMPLE 62. MOD function

B.35. NUMVAL

The NUMVAL function returns the numeric value represented by the character string specified by *argument-1*. Leading and trailing spaces are ignored.

The type of this function is numeric.

B.35.1 General format

FUNCTION NUMVAL (*argument-1*)

B.35.2 Arguments

1) *Argument-1* shall be an alphanumeric literal or an alphanumeric data item whose content has one of the following two formats:

$$[\ space\text{-}string\ ]\left[\begin{matrix}+\\-\end{matrix}\right][\ space\text{-}string\ ]\left\{\begin{matrix}digit\ [\ .\ [\ digit\ ]\ ]\\ .\ digit\end{matrix}\right\}[\ space\text{-}string\ ]$$

or

$$[\ space\text{-}string\ ]\left\{\begin{matrix}digit\ [\ .\ [\ digit\ ]\ ]\\ .\ digit\end{matrix}\right\}[\ space\text{-}string\ ]\left[\begin{matrix}+\\-\\ \underline{CR}\\ \underline{DB}\end{matrix}\right][\ space\text{-}string\ ]$$

where

space-string is a string of one or more space characters and *digit* is a string of one to 18 digits. If *argument-1* is alphanumeric, CR or DB, if specified, shall be uppercase, lowercase or a combination thereof from the computer's alphanumeric character set.

2) The total number of digits in *argument-1* shall not exceed 18.

3) If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma shall be used in *argument-1* rather than a decimal point.

B.35.3 Returned values

1) The returned value is the numeric value represented by *argument-1*.

2) The number of digits returned is 18.

3) If *argument-1* contains CR, DB, or the minus sign, the returned value is negative.

B.35.4 Example

The following code fragments illustrate the use of this function.

```
01  TEMP     PIC S9(5)V9(5).



    IF (FUNCTION NUMVAL (".935") >= 0.934981) AND
        (FUNCTION NUMVAL (".935") <= 0.935019)
                        PERFORM CORRECT-VALUE.

    MOVE ZERO TO TEMP.
    COMPUTE TEMP = FUNCTION NUMVAL ("+394.2").
    IF (TEMP >= 394.192) AND
       (TEMP <= 394.208)
                        PERFORM CORRECT-VALUE.

    COMPUTE TEMP = FUNCTION NUMVAL ("   200.0002   - ").
    IF (TEMP >= -200.0042) AND
       (TEMP <= -199.9962)
                        PERFORM CORRECT-VALUE.
```

EXAMPLE 63. NUMVAL function

B.36. NUMVAL-C

The NUMVAL-C function returns the numeric value represented by the character string specified by argument-1. The currency sign, if any, and any grouping separators preceding the decimal separator are ignored. Optionally, the currency sign may be specified by *argument-2*.

The type of this function is numeric.

B.36.1 General format

FUNCTION NUMVAL-C (*argument-1* [, *argument-2*])

B.36.2 Arguments

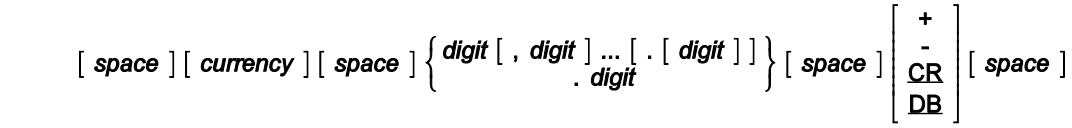
1) *Argument-1* shall be of class alphanumeric.

2) *Argument-2*, if specified, shall be of the same class as *argument-1*. *Argument-2* shall contain exactly one non-space character. *Argument-2* shall not contain any of the digits 0 through 9; characters '*', '+', '–', ',' ,'.' or space. *Argument-2* specifies a currency sign that may appear in *argument-1*.

3) If *argument-2* is not specified, there shall be only one currency sign for the compilation unit, either the default currency sign or one specified in the SPECIAL-NAMES paragraph.

4) *Argument-1* shall have one of the following two formats:

or



where

— *digits* is a string of one or more of the digits 0 through 9;
— except for currency, uppercase letters and the corresponding lowercase letters are equivalent;
— space is a string of zero or more spaces;
— currency is a string of one or more characters matching the currency sign in *argument-2*, if specified, or matching the default currency sign if *argument-2* is not specified;

5) If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the decimal separator is a comma and the grouping separator is the decimal point.

6) The total number of digits in *argument-1* shall not exceed 18.

B.36.3 Returned values

1) The returned value is the numeric value represented by *argument-1*.

2) The number of digits returned is 18.

3) The returned value is negative if *argument-1* contains CR, DB, or a minus sign.

B.36.4 Example

The following code fragments illustrate the use of this function.

```
01 NUMVALC                    PIC S9(7)V9(5).


COMPUTE NUMVALC = FUNCTION NUMVAL-C ("90") + 10.
IF NUMVALC = 100 THEN PERFORM CORRECT-VALUE.

COMPUTE NUMVALC = FUNCTION NUMVAL-C ("$924.93", "$").
IF NUMVALC = 924.912 PERFORM CORRECT-VALUE.

COMPUTE NUMVALC = FUNCTION NUMVAL-C ("$93,021", "$").
IF NUMVALC = 93021 PERFORM CORRECT-VALUE.
```

EXAMPLE 64. NUMVAL-C function

B.37. NUMVAL-F

The NUMVAL-F function returns the value or an approximation of the value represented by the character string specified by *argument-1*. Leading, trailing, and embedded spaces are ignored.

The type of this function is numeric.

B.37.1 General format

FUNCTION NUMVAL-F (*argument-1*)

B.37.2 Arguments

1) *Argument-1* shall be an alphanumeric literal or an alphanumeric data item whose content has the following format:

$$[\ space\ ]\left[\begin{matrix}+\\-\end{matrix}\right][\ space\ ]\left\{\begin{matrix}digit\ [\ .\ [\ digit\ ]\ ]\\ .\ digit\end{matrix}\right\}[\ space\ ]\left[\ E\ [\ space\ ]\left\{\begin{matrix}+\\-\end{matrix}\right\}[\ space\ ]\ n\ [\ space\ ]\ \right]$$

where *space* is a string of zero or more spaces; *n* is one, two, or three digits representing the exponent; and *digit* is a string of one to 18 digits. If *argument-1* is alphanumeric, *E* shall be either an uppercase or lowercase E in the computer's alphanumeric character set.

2) The total number of digits in the significand shall not exceed 18.

3) If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma shall be used in *argument-1* rather than a decimal point.

B.37.3 Returned values

1) Leading, trailing, and embedded spaces are ignored.

2) The returned value is the numeric value represented by argument-1, assuming that it can be expressed within 18 decimal digits.

B.37.4 Example

The following code fragments illustrate the use of this function.

```
01 NUMVALF             PIC S9(7)V9(5).

COMPUTE NUMVALF = FUNCTION NUMVAL-F ( "35").
IF NUMVALF = 35  PERFORM CORRECT-VALUE.

COMPUTE NUMVALF = FUNCTION NUMVAL-F ( "3E2" ).
IF NUMVALF = 300 PERFORM CORRECT-VALUE.

COMPUTE NUMVALF = FUNCTION NUMVAL-F ( "3E-2" ).
IF NUMVALF = .03 PERFORM CORRECT-VALUE.
```

EXAMPLE 65. NUMVAL-F function

B.38. ORD

The ORD function returns an integer value that is the ordinal position of *argument-1* in the program collating sequence. The lowest ordinal position is 1.

The type of this function is integer.

B.38.1 General format

FUNCTION ORD (*argument-1*)

B.38.2 Arguments

1) *Argument-1* shall be of one character position in length and shall be of class alphabetic or alphanumeric.

B.38.3 Returned values

1) The returned value shall be the ordinal position of *argument-1* in the current program collating sequence.

B.38.4 Example

The following code fragments illustrate the use of this function.

```
01 ORDINT              PIC S9(10).
01 A                   PIC X VALUE "F".



IF FUNCTION ORD("5") = 54 THEN
       PERFORM CORRECT-VALUE.

COMPUTE ORDINT = FUNCTION ORD(A).
IF ORDINT = 71 THEN
        PERFORM CORRECT-VALUE.
```

EXAMPLE 66. ORD function

B.39. ORD-MAX

The ORD-MAX function returns a value that is the ordinal number of the *argument-1* that contains the maximum value.

The type of this function is integer.

B.39.1 General format

FUNCTION ORD-MAX ({ *argument-1* }...)

B.39.2 Arguments

1) *Argument-1* shall not be of class pointer.

2) All arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

B.39.3 Returned values

1) The returned value is the ordinal number that corresponds to the position of the *argument-1* having the greatest value in the *argument-1* series.

2) The comparisons used to determine the greatest valued argument are made according to the rules for simple conditions. (See Page 233, Simple conditions.)

3) If the value of more than one *argument-1* is equal to the greatest value, the number returned corresponds to the position of the leftmost *argument-1* having that value.

B.39.4 Example

The following code fragments illustrate the use of this function.

```
01 ORDMAX PIC S9(10).
01  A     PIC S9(10)     VALUE 5.
01  B     PIC S9(10)     VALUE 7.
01  C     PIC S9(10)     VALUE 4.



COMPUTE ORDMAX = FUNCTION ORD-MAX(5, 3, 2, 8, 3, 1).
IF ORDMAX = 4 THEN PERFORM CORRECT-VALUE.
COMPUTE ORDMAX = FUNCTION ORD-MAX (A, B, C).
IF ORDMAX = 4 THEN PERFORM CORRECT-VALUE.
```

EXAMPLE 67. ORD-MAX function

B.40. ORD-MIN

The ORD-MIN function returns a value that is the ordinal number of the argument that contains the minimum value.

The type of this function is integer.

B.40.1 General format

FUNCTION ORD-MIN ({ *argument-1* }...)

B.40.2 Arguments

1) *Argument-1* shall not be of pointer.

2) If more than one *argument-1* is specified, all arguments shall be of the same class with the exception that mixing of arguments of alphabetic and alphanumeric classes is allowed.

B.40.3 Returned values

1) The returned value is the ordinal number that corresponds to the position of the *argument-1* having the least value in the *argument-1* series.

2) The comparisons used to determine the least valued *argument-1* are made according to the rules for simple conditions. (See Page 233, Simple conditions.)

3) If the value of more than one *argument-1* is equal to the least value, the number returned corresponds to the position of the leftmost *argument-1* having that value.

B.40.4 Example

The following code fragments illustrate the use of this function.

```
01 ORDMIN                PIC S9(10).

01  A                    PIC S9(10)     VALUE 5.
01  B                    PIC S9(10)     VALUE 7.
01  C                    PIC S9(10)     VALUE 4.
01  D                    PIC S9(10)     VALUE 10.



COMPUTE ORDMIN = FUNCTION ORD-MIN(5, 3, 2, 8, 3, 1).
IF ORDMIN = 6 THEN
    PERFORM CORRECT-VALUE.

COMPUTE ORDMIN = FUNCTION ORD-MIN(A, B, D).
IF ORDMIN = 1 THEN
    PERFORM CORRECT-VALUE.
```

EXAMPLE 68. ORD-MIN function

B.41. PI

The PI function returns a value that is an approximation of pi, the ratio of the circumference of a circle to its diameter.

The type of this function is numeric.

B.41.1 General format

FUNCTION PI

B.41.2 Returned values

1) The equivalent arithmetic expression shall be

(3 + .141592653589793238)

B.41.3 Example

The following code fragments illustrate the use of this function.

```
01 PI-NUM              PIC 9.9(5).
01 EXP-PI              PIC 9V9(5) VALUE 3.14159.
01 EXP-DISP-PI         PIC 9.9(5).



MOVE FUNCTION PI TO PI-NUM.
MOVE EXP-PI TO EXP-DISP-PI.
IF PI-NUM = EXP-DISP-PI
    PERFORM CORRECT-VALUE.
```

EXAMPLE 69. PI function

B.42. PRESENT-VALUE

The PRESENT-VALUE function returns a value that approximates the present value of a series of future period-end mounts specified by *argument-2* at a discount rate specified by *argument-1*.

The type of this function is numeric.

B.42.1 General format

FUNCTION PRESENT-VALUE (*argument-1*, { *argument-2* }...)

B.42.2 Arguments

1) *Argument-1* and *argument-2* shall be of the class numeric.

2) The value of *argument-1* shall be greater than –1.

B.42.3 Returned values

1) The equivalent arithmetic expression shall be as follows:

a) For one occurrence of *argument-2*,

(*argument-2* / (1 + *argument-1*))

b) For two occurrences of *argument-2*,

(*argument-2* $_1$ / (1 + *argument-1*) + *argument-2* $_2$ / (1 + *argument-1*) ** 2)

c) For *n* occurrences of *argument-2*, the equivalent arithmetic expression shall be

(FUNCTION SUM (

(*argument-2* $_1$ / (1 + *argument-1*) ** 1)

...

(*argument-2* $_n$ / (1 + *argument-1*) ** n)))

where *argument-1* and *argument-2* $_i$ in the terms of the SUM function are the same as the arguments for the PRESENT-VALUE function itself.

B.42.4 Example

The following code fragments illustrate the use of this function.

```
01 PV-NUM      PIC S9(5)V9(6).



MOVE 43.9991 TO MINVAL.
MOVE 44.0009 TO MAXVAL.
COMPUTE PV-NUM = FUNCTION PRESENT-VALUE(0, 23, 12, 9).
IF (PV-NUM >= MINVAL) AND
   (PV-NUM <= MAXVAL) THEN
               PERFORM CORRECT-VALUE.

MOVE  65.9974 TO MINVAL.
MOVE  66.0026 TO MAXVAL.
COMPUTE PV-NUM = FUNCTION PRESENT-VALUE
                          (-.5, (2 + 3), (6 / 3), (9 - 3)).
IF (PV-NUM >= MINVAL) AND
   (PV-NUM <= MAXVAL) THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 70. PRESENT-VALUE function

B.43. RANDOM

The RANDOM function returns a numeric value that is a pseudo-random number from a rectangular distribution.

The type of this function is numeric.

B.43.1 General format

FUNCTION RANDOM [(*argument-1*)]

B.43.2 Arguments

1) If *argument-1* is specified, it shall be zero or a positive integer. It is used as the seed value to generate a sequence of pseudo-random numbers.

2) If a subsequent reference specifies *argument-1*, a new sequence of pseudo-random numbers is started.

3) If the first reference to this function in the run unit does not specify *argument-1*, the seed value is zero.

4) In each case, subsequent references without specifying *argument-1* return the next number in the current sequence.

B.43.3 Returned values

1) The returned value is greater than or equal to zero and less than one.

2) For a given seed value on a given implementation, the sequence of pseudo-random numbers will always be the same.

3) The subset of the domain of *argument-1* values that will yield distinct sequences of pseudo-random numbers is 0 through 2^{31}-2.

B.43.4 Example

The following code fragments illustrate the use of this function.

```
01 RANDNUM     PIC S9(5)V9(6).



COMPUTE RANDNUM = FUNCTION RANDOM.
IF (RANDNUM >= 0) AND
   (RANDNUM < 1) THEN
               PERFORM CORRECT-VALUE.

COMPUTE RANDNUM = FUNCTION RANDOM(2) + 1.
IF (RANDNUM >= 1) AND
   (RANDNUM < 2) THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 71. RANDOM function

B.44. RANGE

The RANGE function returns a value that is equal to the value of the maximum argument minus the value of the minimum argument.

The type of this function depends upon the argument types as follows:

Argument type	Function type
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

B.44.1 General format

FUNCTION RANGE ({ *argument-1* }...)

B.44.2 Arguments

1) *Argument-1* shall be class numeric.

B.44.3 Returned values

1) The equivalent arithmetic expression shall be

(FUNCTION MAX (*argument-list*) – FUNCTION MIN (*argument-list*))

where *argument-list* is the *argument-1* list for the RANGE function itself.

B.44.4 Example

The following code fragments illustrate the use of this function.

EXAMPLE 72. RANGE function

```
01  RANGENUM   PIC S9(7)V9(7).
01  A          PIC S9(10)     VALUE 6.
01  B          PIC S9(10)     VALUE 8.
01  C          PIC S9(10)     VALUE -5.
01  D          PIC S9(10)     VALUE 12.


COMPUTE RANGENUM = FUNCTION RANGE(5, -2, -14, 0).
IF RANGENUM = 19 THEN
               PERFORM CORRECT-VALUE.

IF FUNCTION RANGE(A, B, C, D) = 17 THEN
               PERFORM CORRECT-VALUE.
```

B.45. REM

The REM function returns a numeric value that is the remainder of *argument-1* divided by *argument-2*.

The type of this function is numeric.

B.45.1 General format

FUNCTION REM (*argument-1*, *argument-2*)

B.45.2 Arguments

1) *Argument-1* and *argument-2* shall be class numeric.

2) The value of *argument-2* shall not be zero.

B.45.3 Returned values

1) The equivalent arithmetic expression shall be

(*argument-1* – (*argument-2* * FUNCTION INTEGER-PART (*argument-1* / *argument-2*)))

where *argument-1* and *argument-2* of the INTEGER-PART function are the same as the arguments for the REM function itself.

B.45.4 Example

The following code fragments illustrate the use of this function.

```
01 REMNUM      PIC S9(5)V9(6).
01  A          PIC S9(10)      VALUE 5

.

COMPUTE REMNUM = FUNCTION REM(-11, -5).
IF REMNUM = -1 THEN
               PERFORM CORRECT-VALUE.

COMPUTE REMNUM = FUNCTION REM(A, 2).
IF REMNUM = 1 THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 73. REM function

B.46. REVERSE

The REVERSE function returns a character string of exactly the same length as *argument-1* and whose characters are exactly the same as those of *argument-1*, except that they are in reverse order.

The type of the function depends on the argument type as follows:

Argument type	Function type
Alphabetic	Alphanumeric
Alphanumeric	Alphanumeric

B.46.1 General format

FUNCTION REVERSE (*argument-1*)

B.46.2 Arguments

1) *Argument-1* shall be of class alphabetic or alphanumeric and shall be at least one character position in length.

B.46.3 Returned values

1) If *argument-1* is a character string of length n, the returned value is a character string of length n such that for $1 \le j \le n$, the character in position j of the returned value is the character from position $n - j + 1$ of *argument-1*.

B.46.4 Example

The following code fragments illustrate the use of this function.

```
01 REVSNUM              PIC X(10).


MOVE FUNCTION REVERSE("figure") TO REVSNUM.
IF REVSNUM = "erugif" THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 74. REVERSE function

B.47. SIGN

The SIGN function returns +1, 0, or –1 depending on the sign of the argument.

The type of the function is integer.

B.47.1 General Format

FUNCTION SIGN (*argument-1*)

B.47.2 Arguments

1) *Argument-1* shall be class numeric.

B.47.3 Returned Values

1) The equivalent arithmetic expression shall be as follows:

a) When the value of *argument-1* is positive,

(1)

b) When the value of *argument-1* is zero,

(0)

c) When the value of *argument-1* is negative,

(–1)

B.47.4 Example

The following code fragments illustrate the use of this function.

```
01 VAL                PIC S9(1).
01 EXP-VAL1           PIC S9(1) VALUE -1.
01 EXP-VAL2           PIC S9(1) VALUE 0.
01 EXP-VAL3           PIC S9(1) VALUE 1.


MOVE -34431 TO NUM1.
COMPUTE VAL = FUNCTION SIGN (NUM1).
IF VAL = EXP-VAL1 THEN PERFORM CORRECT-VALUE.

MOVE 0 TO NUM1.
COMPUTE VAL = FUNCTION SIGN (NUM1).
IF VAL = EXP-VAL2 THEN PERFORM CORRECT-VALUE.

MOVE 34431 TO NUM1.
COMPUTE VAL = FUNCTION SIGN (NUM1).
IF VAL = EXP-VAL3 THEN PERFORM CORRECT-VALUE.
```

EXAMPLE 75. SIGN function

B.48. SIN

The SIN function returns a numeric value that approximates the sine of an angle or arc, expressed in radians, that is specified by *argument-1*.

The type of this function is numeric.

B.48.1 General format

FUNCTION SIN (*argument-1*)

B.48.2 Arguments

1) *Argument-1* shall be class numeric.

B.48.3 Returned values

1) The returned value is the approximation of the sine of *argument-1* and is greater than or equal to –1 and less than or equal to +1.

B.48.4 Example

The following code fragments illustrate the use of this function.

```
01 SINNUM      PIC S9(5)V9(6).



MOVE ZERO TO SINNUM.
MOVE -0.000020 TO MINVAL.
MOVE  0.000020 TO MAXVAL.

COMPUTE SINNUM = FUNCTION SIN(0).
IF (SINNUM >= MINVAL) AND
   (SINNUM <= MAXVAL) THEN
               PERFORM CORRECT-VALUE.
MOVE  0.865990 TO MINVAL.
MOVE  0.866060 TO MAXVAL.

COMPUTE SINNUM = FUNCTION SIN(PI / 3).

IF (SINNUM >= MINVAL) AND
   (SINNUM <= MAXVAL) THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 76. SIN function

B.49. SQRT

The SQRT function returns a numeric value that approximates the square root of *argument-1*.

The type of this function is numeric.

B.49.1 General format

FUNCTION SQRT (*argument-1*)

B.49.2 Arguments

1) *Argument-1* shall be class numeric.

2) The value of *argument-1* shall be zero or positive.

B.49.3 Returned values

1) *Argument-1* is not rounded.

2) The returned value shall be the absolute value of the exact square root of *argument-1* truncated to 19 digits.

B.49.4 Example

The following code fragments illustrate the use of this function.

```
01 SQRTNUM     PIC S9(5)V9(7).



MOVE  0.000000 TO MINVAL.
MOVE  0.000020 TO MAXVAL.
COMPUTE SQRTNUM = FUNCTION SQRT(0).
IF (SQRTNUM >= MINVAL) AND
   (SQRTNUM <= MAXVAL)
               PERFORM CORRECT-VALUE.

MOVE  0.316214  TO MINVAL.
MOVE  0.316240 TO MAXVAL.
COMPUTE SQRTNUM = FUNCTION SQRT(9 - 8.9).
IF (SQRTNUM >= MINVAL) AND
   (SQRTNUM <= MAXVAL) THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 77. SQRT function

B.50. STANDARD-DEVIATION

The STANDARD-DEVIATION function returns a numeric value that approximates the standard deviation of its arguments.

The type of this function is numeric.

B.50.1 General format

FUNCTION STANDARD-DEVIATION ({ *argument-1* }...)

B.50.2 Arguments

1) *Argument-1* shall be class numeric.

B.50.3 Returned values

1) The equivalent arithmetic expression shall be as follows:

(FUNCTION SQRT (FUNCTION VARIANCE (*argument-list*)))

where *argument-list* is the *argument-1* list for the STANDARD-DEVIATION function itself.

B.50.4 Example

The following code fragments illustrate the use of this function.

```
01 STDNUM      PIC S9(5)V9(6).


MOVE 6.92 TO MINVAL.
MOVE 7.02 TO MAXVAL.
COMPUTE STDNUM =
       FUNCTION STANDARD-DEVIATION(5, -2, -14, 0).
IF (STDNUM >= MINVAL) AND
   (STDNUM <= MAXVAL) THEN
               PERFORM CORRECT-VALUE.

MOVE  11.7995 TO MINVAL.
MOVE  11.8005 TO MAXVAL.
COMPUTE STDNUM =
   FUNCTION STANDARD-DEVIATION(2.6 + 30, 4.5 * 2).

IF (STDNUM >= MINVAL) AND
   (STDNUM <= MAXVAL) THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 78. STANDARD-DEVIATION function

B.51. SUM

The SUM function returns a value that is the sum of the arguments.

The type of this function depends upon the argument types as follows:

Argument type	Function type
All arguments integer	Integer
Numeric (some arguments may be integer)	Numeric

B.51.1 General format

FUNCTION SUM ({ *argument-1* }...)

B.51.2 Arguments

1) *Argument-1* shall be class numeric.

B.51.3 Returned values

1) The equivalent arithmetic expression shall be as follows:

a) For one occurrence of *argument-1*,

(*argument-1*)

b) For two occurrences of *argument-1*,

($argument\text{-}1_1 + argument\text{-}1_2$)

c) For *n* occurrences of *argument-1*,

($argument\text{-}1_1 + argument\text{-}1_2 + ... + argument\text{-}1_n$)

B.51.4 Example

The following code fragments illustrate the use of this function.

```
01 SUMNUM      PIC S9(6)V9(7).


COMPUTE SUMNUM = FUNCTION SUM(5, -2, -14, 0).
IF SUMNUM = -11 THEN
               PERFORM CORRECT-VALUE.
MOVE  41.5992 TO MINVAL.
MOVE  41.6008 TO MAXVAL.
COMPUTE SUMNUM = FUNCTION SUM(2.6 + 30, 4.5 * 2).
IF (SUMNUM >= MINVAL) AND
   (SUMNUM <= MAXVAL) THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 79. SUM function

B.52. TAN

The TAN function returns a numeric value that approximates the tangent of an angle or arc, expressed in radians, that is specified by *argument-1*.

The type of this function is numeric.

B.52.1 General format

FUNCTION TAN (*argument-1*)

B.52.2 Arguments

1) *Argument-1* shall be class numeric.

B.52.3 Returned values

1) The returned value is the approximation of the tangent of *argument-1*.

B.52.4 Example

The following code fragments illustrate the use of this function.

```
01 TAN-NUM     PIC S9(5)V9(7).


MOVE -0.000020 TO MINVAL.
MOVE  0.000020 TO MAXVAL.
COMPUTE TAN-NUM = FUNCTION TAN(0).
IF (TAN-NUM >= MINVAL) AND
   (TAN-NUM <= MAXVAL) THEN
          PERFORM CORRECT-VALUE.

MOVE  0.999960 TO MINVAL.
MOVE  1.00004 TO MAXVAL.
COMPUTE TAN-NUM = FUNCTION TAN(PI / 4).
IF (TAN-NUM >= MINVAL) AND
   (TAN-NUM <= MAXVAL) THEN
          PERFORM CORRECT-VALUE.
```

EXAMPLE 80. TAN function

B.53. TEST-DATE-YYYYMMDD

The TEST-DATE-YYYYMMDD function tests whether a date in standard date form (YYYYMMDD) is a valid date in the Gregorian calendar. *Argument-1* of the INTEGER-OF-DATE function is required to be in standard date form.

The type of this function is integer.

B.53.1 General Format

FUNCTION TEST-DATE-YYYYMMDD (*argument-1*)

B.53.2 Arguments

1) *Argument-1* shall be an integer.

B.53.3 Returned values

1) The returned value is:

a) If the value of *argument-1* is less than 16010000 or greater than 99999999,

(1)

Note 1 — The year is not within the range 1601 to 9999.

a) Otherwise, if the value of FUNCTION MOD (*argument-1* 10000) is less than 100 or greater than 1299,

(2)

Note 2 — The month is not within the range 1 through 12.

c) Otherwise, if the value of FUNCTION MOD (*argument-1* 100) is less than 1 or greater than the number of days in the month determined by FUNCTION INTEGER (FUNCTION MOD (*argument-1* 10000) / 100) of the year determined by FUNCTION INTEGER (*argument-1* / 10000),

(3)

Note 3 — The day is not valid for the given year and month.

d) Otherwise,

(0)

Note 4 — The date is valid.

B.53.4 Example

The following code fragments illustrate the use of this function.

```
01 DATE-VAL  PIC 9.


ACCEPT SYSTEM-DATE FROM DATE YYYYMMDD.
COMPUTE DATE-VAL =  FUNCTION TEST-DATE-YYYYMMDD ( SYSTEM-DATE ).
IF DATE-VAL = 0 THEN
             PERFORM CORRECT-VALUE.

*** year out of range.
COMPUTE DATE-VAL =  FUNCTION TEST-DATE-YYYYMMDD ( 14000000 ).
IF DATE-VAL = 1 THEN
             PERFORM CORRECT-VALUE.

*** month out of range.
COMPUTE DATE-VAL =  FUNCTION TEST-DATE-YYYYMMDD ( 20000000 ).
IF DATE-VAL = 2 THEN
             PERFORM CORRECT-VALUE.
```

EXAMPLE 81. TEST-DATE-YYYYMMDD function

B.54. TEST-DAY-YYYYDDD

The TEST-DAY-YYYYDDD function tests whether a date in Julian date form (YYYYDDD) is a valid date in the Gregorian calendar. *Argument-1* of the INTEGER-OF-DAY function is required to be in Julian date form.

The type of this function is integer.

B.54.1 General Format

FUNCTION TEST-DAY-YYYYDDD (*argument-1*)

B.54.2 Arguments

1) *Argument-1* shall be an integer.

B.54.3 Returned values

1) The returned value is:

a) If the value of *argument-1* is less than 1601000 or greater than 9999999,

(1)

Note 1 — The year is not within the range 1601 to 9999.

b) Otherwise, if the value of FUNCTION MOD (*argument-1* 1000) is less than 1 or greater than the number of days in the year determined by FUNCTION INTEGER (*argument-1* / 1000),

(2)

Note 2 — The day is not valid in the given year.

c) Otherwise,

(0)

Note 3 — The date is valid.

B.54.4 Example

The following code fragments illustrate the use of this function.

```
01 DAY-VAL     PIC 9.


ACCEPT SYSTEM-DAY FROM DAY YYYYDDD.
COMPUTE DAY-VAL =  FUNCTION TEST-DAY-YYYYDDD ( SYSTEM-DAY )
IF DAY-VAL = 0 THEN
               PERFORM CORRECT-VALUE.

**** year out of range
COMPUTE DAY-VAL =  FUNCTION TEST-DAY-YYYYDDD ( 1400000 ).
IF DAY-VAL = 1 THEN
               PERFORM CORRECT-VALUE.

**** day out of range
COMPUTE DAY-VAL =  FUNCTION TEST-DAY-YYYYDDD ( 1700462 ).
IF DAY-VAL = 2 THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 82. TEST-DAY-YYYYDDD function

B.55. TEST-NUMVAL

The TEST-NUMVAL function verifies that the contents of *argument-1* conform to the specification for argument-1 of the NUMVAL function.

The type of this function is integer.

B.55.1 General Format

FUNCTION TEST-NUMVAL (*argument-1*)

B.55.2 Arguments

1) *Argument-1* shall be an alphanumeric literal or an alphanumeric data item.

B.55.3 Returned values

1) The returned value is:

   a) If the content of *argument-1* does not conform to the argument rules for the NUMVAL function, the returned value shall be the position of the first character in error, from (1) to (FUNCTION LENGTH (*argument-1*) + 1)

   b) Otherwise:

      (0)

   NOTES

   1 — The returned value is (FUNCTION LENGTH (*argument-1*) + 1) if *argument-1* is zero-length or contains only spaces or a string such as " +.".

   2 — The returned value is (3) if a three-character argument contains a string such as "0 1".

   3 — The returned value identifies the position of the 19th digit if the total number of digits exceeds 18.

B.55.4 Example

The following code fragments illustrate the use of this function.

```
01 TST-NUMV    PIC 9.



COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL ( "35").
IF TST-NUMV = 0 THEN
               PERFORM CORRECT-VALUE.
COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL ( "$35").
IF TST-NUMV = 1
               PERFORM CORRECT-VALUE.
COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL ( "35+" ).
IF TST-NUMV  = 0
               PERFORM CORRECT-VALUE.
COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL ( "35$").
IF TST-NUMV  = 3
               PERFORM CORRECT-VALUE.
```

EXAMPLE 83. TEST-NUMVAL function

B.56. TEST-NUMVAL-C

The TEST-NUMVAL-C function verifies that the contents of *argument-1* conform to the specification for *argument-1* of the NUMVAL-C function.

The type of this function is integer.

B.56.1 General Format

FUNCTION TEST-NUMVAL-C (*argument-1 [,* argument-2])

B.56.2 Arguments

1) *Argument-1* shall be of class alphanumeric.

2) *Argument-2*, if specified, shall be of the same class as *argument-1*. *Argument-2* shall contain exactly one non-space character. Any leading or trailing spaces in *argument-2* are ignored. *Argument-2* shall not contain any of the digits 0 through 9; the characters '*', '+', '–', ',' , and '.' or space. *Argument-2* specifies a curency sign that may appear in *argument-1*.

B.56.3 Returned values

1) The returned value is:

a) If the content of *argument-1* does not conform to the argument rules for *argument-1* of the NUMVAL-C function, the returned value shall be the position of the first character in error, from (1) to (FUNCTION LENGTH (*argument-1*) + 1)

b) Otherwise:

(0)

NOTES

1 — The returned value is (FUNCTION LENGTH (*argument-1*) + 1) if *argument-1* is zero-length or contains only spaces or a string such as " +.".

2 — The returned value is (3) if a three-character argument contains a string such as "0 1".

3 — The returned value identifies the position of the 19th digit if the total number of digits exceeds 18.

B.56.4 Example

The following code fragments illustrate the use of this function.

```
01 TST-NUMV                   PIC 9.


COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-C ( "35").
IF TST-NUMV = 0
               PERFORM CORRECT-VALUE.

COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-C ( "$35", "$").
IF TST-NUMV = 0
               PERFORM CORRECT-VALUE.

COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-C ( "35E" ).
IF TST-NUMV = 3
               PERFORM CORRECT-VALUE.

COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-C ( "3 E 3" ).
IF TST-NUMV = 3
               PERFORM CORRECT-VALUE.

COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-C ( "35,433$", "$").
IF TST-NUMV = 7
               PERFORM CORRECT-VALUE.

COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-C ( A ).
IF TST-NUMV = 0
               PERFORM CORRECT-VALUE.
```

EXAMPLE 84. TEST-NUMVAL-C function

B.57. TEST-NUMVAL-F

The TEST-NUMVAL-F function verifies that the contents of *argument-1* conform to the specification for *argument-1* of the NUMVAL-F function.

The type of this function is integer.

B.57.1 General Format

FUNCTION TEST-NUMVAL-F (*argument-1*)

B.57.2 Arguments

1) *Argument-1* shall be an alphanumeric literal or an alphanumeric data item.

B.57.3 Returned values

1) The returned value is:

a) If the content of *argument-1* does not conform to the argument rules for the NUMVAL-F function, the returned value shall be the position of the first character in error, from (1) to (FUNCTION LENGTH (*argument-1*) + 1)

b) Otherwise, if the numeric value represented by *argument-1* cannot be represented in a floating-point data item because of exponent overflow:

(–1)

c) Otherwise, if the numeric value represented by *argument-1*, cannot be represented in a floating-point data item because of exponent underflow:

(–2)

d) Otherwise:

(0)

NOTES

1 — The returned value is (FUNCTION LENGTH (*argument-1*) + 1) if *argument-1* is zero-length or contains only spaces or a string such as “1.5E”.

2 — The returned value is (3) if a three-character argument contains a string such as “0 1”.

3 — If the total number of digits in the significand exceeds 18, the returned value identifies the position of the 19th digit.

4 — If the total number of digits in the exponent exceeds 3, the returned value identifies the position of the 4th digit.

5 — If the exponent has no sign, the returned value identifies the position of the first exponent digit.

B.57.4 Example

The following code fragments illustrate the use of this function.

EXAMPLE 85. TEST-NUMVAL-F function

```
01 TST-NUMV                    PIC 9.


COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-F ( "35").
IF TST-NUMV = 0 THEN
               PERFORM CORRECT-VALUE.

COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-F ( "3E2" ).
IF TST-NUMV = 0 THEN
               PERFORM CORRECT-VALUE.

COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-F ( "3E-2" ).
IF TST-NUMV = 0 THEN
               PERFORM CORRECT-VALUE.

COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-F ( "35$" ).
IF TST-NUMV = 3 THEN
               PERFORM CORRECT-VALUE.

COMPUTE TST-NUMV = FUNCTION TEST-NUMVAL-F ( "3,433").
IF TST-NUMV = 2 THEN
               PERFORM CORRECT-VALUE.
```

B.58. UPPER-CASE

The UPPER-CASE function returns a character string that is the same length as *argument-1* with each lowercase letter replaced by the corresponding uppercase letter.

The type of the function depends on the argument type as follows:

Argument type	Function type
Alphabetic	Alphabetic
Alphanumeric	Alphanumeric

B.58.1 General format

FUNCTION UPPER-CASE (*argument-1*)

B.58.2 Arguments

1) *Argument-1* shall be of class alphabetic or alphanumeric and shall be at least one character position in length.

B.58.3 Returned values

1) The same character string as *argument-1* is returned, except that each lowercase letter is replaced by the corresponding uppercase letter.

2) The character string returned has the same length as *argument-1*.

3) If the computer's character set does not include uppercase letters, no changes take place in the character string.

B.58.4 Example

The following code fragments illustrate the use of this function.

```
01 ANY-CHANGE-ANSWER      PIC X(7).


MOVE "abcdefg" TO ANY-CHANGE-ANSWER.
MOVE FUNCTION UPPER-CASE (ANY-CHANGE-ANSWER)
       TO ANY-CHANGE-ANSWER.
IF ANY-CHANGE-ANSWER = "ABCDEFG" THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 86. UPPER-CASE function

B.59. VARIANCE

The VARIANCE function returns a numeric value that approximates the variance of its arguments.

The type of this function is numeric.

B.59.1 General format

FUNCTION VARIANCE ({ *argument-1* }...)

B.59.2 Arguments

1) *Argument-1* shall be class numeric.

B.59.3 Returned values

1) The equivalent arithmetic expression shall be as follows:

a) For one occurrence of *argument-1*,

(0)

b) For two occurrences of *argument-1*,

(((*argument-1* $_1$ – FUNCTION MEAN (*argument-list*)) ** 2 +
(*argument-1* $_2$ – FUNCTION MEAN (*argument-list*)) ** 2) / 2)

c) For *n* occurrences of *argument-1*,

(FUNCTION SUM (

((*argument-1* $_1$ – FUNCTION MEAN (*argument-list*)) ** 2)

...

((*argument-1* $_n$ – FUNCTION MEAN (*argument-list*)) ** 2)) / n)

where *argument-list* is the *argument-1* list for the VARIANCE function itself and argument-1 $_i$ is the ith argument of the *argument-1* list for the VARIANCE function itself.

B.59.4 Example

The following code fragments illustrate the use of this function.

```
01 VARNUM             PIC S9(5)V9(6).



MOVE  48.6865 TO MINVAL.
MOVE  48.6885 TO MAXVAL.
COMPUTE VARNUM = FUNCTION VARIANCE(5, -2, -14, 0).
IF (VARNUM >= MINVAL) AND
   (VARNUM <= MAXVAL) THEN
               PERFORM CORRECT-VALUE.

MOVE  139.234 TO MINVAL.
MOVE  139.245 TO MAXVAL.
COMPUTE VARNUM = FUNCTION VARIANCE(2.6 + 30, 4.5 * 2).
IF (VARNUM >= MINVAL) AND
   (VARNUM <= MAXVAL) THEN
               PERFORM CORRECT-VALUE.
```

EXAMPLE 87. VARIANCE function

B.60. WHEN-COMPILED

The WHEN-COMPILED function returns the date and time the compilation unit was compiled as provided by the system on which the compilation unit was compiled.

The type of this function is alphanumeric.

B.60.1 General format

FUNCTION WHEN-COMPILED

B.60.2 Returned values

1) The character positions returned, numbered from left to right, are:

Character Positions	Contents
1-4	Four numeric digits of the year in the Gregorian calendar.
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-16	Two numeric digits of the hundredths of a second past the second, in the range 00 through 99. The value 00 is returned if the system on which the compilation was done does not have the facility to provide the fractional part of a second.
17	Either the character '–', the character '+', or the character '0'. The character '–' is returned if the local time indicated in the previous character positions is behind Coordinated Universal Time. The character '+' is returned if the local time indicated is the same as or ahead of Coordinated Universal Time. The character '0' is returned if the system on which the compilation was done does not have the facility to provide the local time differential factor.
18-19	If character position 17 is '–', two numeric digits are returned in the range 00 through 12 indicating the number of hours that the reported time is behind Coordinated Universal Time. If character position 17 is '+', two numeric digits are returned in the range 00 through 13 indicating the number of hours that the reported time is ahead of Coordinated Universal Time. If character position 17 is '0', the value 00 is returned.
20-21	Two numeric digits are returned in the range 00 through 59 indicating the number of additional minutes that the reported time is ahead of or behind Coordinated Universal Time, depending on whether character position 17 is '+' or '–', respectively. If character position 17 is '0', the value 00 is returned.

2) The returned value is the date and time of compilation of the compilation unit that contains this function. The returned value in a contained source unit is the compilation date and time associated with the compilation unit in which it is contained.

3) The returned value shall denote the same time as the compilation date and time if provided in the listing and in the generated object code, although their representations and precision may differ.

B.60.3 Example

The following code fragments illustrate the use of this function.

EXAMPLE 88. WHEN-COMPILED function

```
01 TEMP1                              PIC X(21).
01 WS-DATE.
      02  WS-YEAR                     PIC 9999.
               88 COM-YEAR                 VALUE 1990 THRU 9999.
      02  WS-MONTH                    PIC 99.
               88 COM-MONTH                VALUE 01 THRU 12.
      02  WS-DAY                      PIC 99.
               88 COM-DAY                  VALUE 01 THRU 31.
      02  WS-HOUR                     PIC 99.
               88 COM-HOUR                 VALUE 00 THRU 23.
      02  WS-MIN                      PIC 99.
               88 COM-MIN                  VALUE 00 THRU 59.
      02  WS-SECOND                   PIC 99.
               88 COM-SEC                  VALUE 00 THRU 59.
      02  WS-HUNDSEC                  PIC 99.
               88 COM-HUNDSEC              VALUE 00 THRU 99.
      02  WS-GREENW                   PIC X.
               88 COM-GREENW               VALUE "-", "+", "0".
      02 WS-OFFSET                    PIC 99.
               88 COM-OFFSET               VALUE 00 THRU 13.


MOVE FUNCTION WHEN-COMPILED TO TEMP1.
MOVE TEMP1 TO WS-DATE.
IF COM-YEAR            AND
   COM-MONTH           AND
   COM-DAY             AND
   COM-HOUR            AND
   COM-MIN             AND
   COM-SEC             AND
   COM-HUNDSEC         AND
   COM-GREENW          AND
   COM-OFFSET          THEN
       PERFORM CORRECT-VALUE.
```

B.61. YEAR-TO-YYYY

The YEAR-TO-YYYY function converts *argument-1*, the two low-order digits of a year, to a four-digit year. *Argument-2*, when added to the year at the time of execution, defines the ending year of a 100-year interval, or sliding window, into which the year of *argument-1* falls.

The type of the function is integer.

B.61.1 General format

FUNCTION YEAR-TO-YYYY (*argument-1* [, *argument-2*])

B.61.2 Arguments

1) *Argument-1* shall be a nonnegative integer that is less than 100.

2) *Argument-2* shall be an integer.

3) If *argument-2* is omitted, the function shall be evaluated as though 50 were specified.

4) The sum of the year at the time of execution and the value of *argument-2* shall be less than 10000 and greater than 1699.

B.61.3 Returned values

1) *Maximum-year* shall be calculated as follows:

(FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4)) + *argument-2*)

where *argument-2* of the NUMVAL function is the same as *argument-2* of the YEAR-TO-YYYY function itself.

2) The equivalent arithmetic expression shall be as follows:

a) When the following condition is true

FUNCTION MOD (*maximum-year*, 100) >= *argument-1*

The equivalent arithmetic expression shall be

(*argument-1* + 100 * (FUNCTION INTEGER (*maximum-year*/100)))

b) Otherwise, the equivalent arithmetic expression shall be

(*argument-1* + 100 * (FUNCTION INTEGER (*maximum-year*/100) – 1))

NOTES

1 — In the year 1995, the returned value for FUNCTION YEAR-TO-YYYY (4, 23) is 2004. In the year 2008 the returned value for FUNCTION YEAR-TO-YYYY (98, (–15)) is 1898.

2 — The YEAR-TO-YYYY function implements a sliding window algorithm. To use it for a fixed window, *argument-2* can be specified as follows, where *fixed-maximum-year* is the maximum year in the fixed 100-year interval:

(*fixed-maximum-year* – FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4)))

If the fixed window is 1973 through 2072, then in 2009 *argument-2* shall have the value of 63 and in 2019, the value of 53.

B.61.4 Example

The following code fragments illustrate the use of this function.

```
01 YEARVAL                    PIC 9(4).



DISPLAY "FUNCTION YEAR-TO-YYYY " NO ADVANCING.
COMPUTE YEARVAL = FUNCTION YEAR-TO-YYYY ( 4, 23 ).
IF YEARVAL = 2004
             PERFORM CORRECT-VALUE.

COMPUTE YEARVAL = FUNCTION YEAR-TO-YYYY ( 98, (-15))
IF YEARVAL = 1898
             PERFORM CORRECT-VALUE.
```

EXAMPLE 89. YEAR-TO-YYYY function

IX. SCREEN HANDLER

A. General Description

The **ICOBOL** SCREEN HANDLER implements a subset of Threshold, Inc.'s SCREEN DEMON calls, which use the CALL mechanism. ICSDMODE instructs **ICOBOL** how to enable the SCREEN HANDLER

A. 1. Enabling the SCREEN HANDLER

By default, the SCREEN HANDLER is disabled in the **ICOBOL** configuration file (.cfi). ICSDMODE is set by using the configuration file (.cfi) or with an environment setting. Any ICSCMODE environment setting overrides the setting in the Program Environment section of the configuration file (.cfi).

The syntax is:

ICSDMODE=disabled | underline | 0 | reverse | 1 | drawlines | 2

Where

disabled
: Disables the SCREEN HANDLER

0 or *underline*
: Run in standard SCREEN DEMON format, which is to underline the row above the box and underline the last row in the box for the top and bottom lines, and use reverse video for the sides.

1 or *reverse*
: Use reverse video for the entire box. This means that two (2) more lines than in standard mode are hidden under the box.

2 or *drawlines*
: Use the line-drawing character set of a terminal for the entire box. As with the previous setting, two (2) more lines than in standard mode are hidden under the box. If a particular terminal does not have a line-drawing character set, then "+", "-", and "|" are used for the corners, horizontal, and vertical portions of the box, respectively. Currently, only the terminal types ibm, xenix, 386ix, pcbios, and pcwindow support the line-drawing characters by default.

The ICSDMODE selection does not affect SD_DRAW_HLINE or SD_DRAW_VLINE or the value of the *height* of a box or the value of the *top-left-line* entry.

SCREEN DEMON is an enhancement product for AOS/VS available from its developer, Threshold, Inc., Auburn, AL. Phone (334) 821-0075.

The table below summarizes the **ICOBOL** SCREEN HANDLER calls that are available when the SCREEN HANDLER feature is enabled. A description of each argument, as well as details for the calls, follow the table.

FUNCTION	ARGUMENTS
SD_DRAW_BOX	USING *top-left-line*, *top-left-column*, *height*, *width* [, *label*]
SD_DRAW_HLINE	USING *top-left-line*, *top-left-column*, *width*
SD_DRAW_VLINE	USING *top-left-line*, *top-left-column*, *height*
SD_ERROR_MESSAGE	USING *msg-string* [, *top-left-line*, *top-left-column*]
SD_GET_IMAGE	USING *image-buffer*
SD_GET_POS	USING *position*
SD_MESSAGE	USING *msg-string* [, *top-left-line*, *top-left-column* [, *label*]]
SD_MESSAGE_ONLY	USING *msg-string* [, *top-left-line*, *top-left-column* [, *label*]]
SD_NEW_WINDOW	[USING *top-left-line*, *top-left-column*, *height*, *width* [, *label*]]
SD_POP_UP_MENU	USING *menu-packet* [, *label*]
SD_POP_UP_MENU2	USING *menu-packet* [, *label*]
SD_READ_CHAR	USING *char-field* [, *time-out-value*]
SD_REDRAW	(none)
SD_REMOVE_WINDOW	(none)
SD_RETURN_INPUT	USING *data-string*, *string-size*
SD_SET_ACCEPT_TIMEOUT	USING *time-out-value*
SD_SYS_ERROR_MESSAGE	USING *error-code* [, *msg-string* [, *top-left-line*, *top-left-column*]]

TABLE 37. Summary of Screen Handler Calls

NOTE: The following are included only for compatibility purposes, and executing them has no impact on how the screen handler operates even though they may set or clear some flags which are otherwise not used. (If these calls were implemented, as they are in the real Screen Demon, they would alter the performance characteristics of your program.) These calls will fail if the ICSDMODE environment variable has not been set. They will also fail if the call interface (i.e., number of arguments or argument size) is invalid. Otherwise, the calls will always succeed (and do nothing).

```
CALL "SD_CONTROL" USING control
CALL "SD_DISABLE"
CALL "SD_ENABLE"
CALL "SD_FLUSH"
CALL "SD_GET_CONTROL USING control
CALL "SD_TURBO_FULL"
CALL "SD_TURBO_OFF"
CALL "SD_TURBO_PARTIAL"
```

NOTE: In SD_CONTROL and SD_GET_CONTROL, the data item *control* is defined as PIC 9(4) COMP.

A.3. Error Handling

The following Exception Status codes may be returned.

Exception Code	Description
241	"The argument is too long to process"
203	"Program not found" if ICSDMODE is not set
076	"Device timeout" when SD_POP_UP_MENU exits due to a timeout set by SD_SET_ACCEPT_TIMEOUT
013	"Invalid data" when a parameter is invalid; i.e., line or column is out of range, string too long for box, etc.
008	"Insufficient memory" when there is no more memory available for screen images.
001	"Invalid operation" when trying to perform an option that is not currently valid; i.e., trying to do a SD_REMOVE_WINDOW when nothing is pushed.

The **ICOBOL** SCREEN HANDLER runs in a mode similar to the SCREEN DEMON partial turbo, i.e., the user's screen is updated at the end of every operation.

The **ICOBOL** SCREEN HANDLER cannot be disabled while under **ICOBOL**.

The calls SD_GET_IMAGE and SD_GET_POS are only defined for screens with 24 lines and 80 columns. If either is called on a larger size screen, SD_GET_IMAGE will return the upper left 24 by 80 quadrant and SD_GET_POS will generate an Exception Status 13.

For menu items there is a limit of 21 items otherwise an Invalid Data is given.

B. Calls

B.1. SD_DRAW_BOX

This call provides the ability to draw a box on the terminal. The area inside the box is cleared to spaces.

The syntax is:

```
CALL "SD_DRAW_BOX" USING top-left-line, top-left-column, height, width
    [, label ]
```

Where

top-left-line
: Specifies a PIC 9(4) COMP and defines the top left line position of a box or line. If <= 2, it will be centered.

top-left-column
: Specifies a PIC 9(4) COMP and defines the top left column position of a box or line. If <= 0, it will be centered.

height
: Specifies a PIC 9(4) COMP and defines how high (in lines) a box or line should extend from the top left position. The height does not include the top line.

width
: Specifies a PIC 9(4) COMP and defines how wide (in columns) a box or line should extend from the top left position. The width includes the edges of the box which are one space wide.

label
: Specifies a PIC X(n) and defines a label to be placed on the top line of a box underlined and bright. The label must be terminated with a null (LOW-VALUE).

B.2. SD_DRAW_HLINE and SD_DRAW_VLINE

These calls allow for either a horizontal line (SD_DRAW_HLINE) or a vertical line (SD_DRAW_VLINE) to be drawn on the terminal.

The syntax is:

```
CALL "SD_DRAW_VLINE" USING top-left-line, top-left-column, height

CALL "SD_DRAW_HLINE" USING top-left-line, top-left-column, width
```

Where

top-left-line
: Specifies a PIC 9(4) COMP and defines the top left line position of a box or line. If <= 2, it will be centered.

top-left-column
: Specifies a PIC 9(4) COMP and defines the top left column position of a box or line. If <= 0, it will be centered.

height
: Specifies a PIC 9(4) COMP and defines how high (in lines) a box or line should extend from the top left position. The height does not include the top line.

width
: Specifies a PIC 9(4) COMP and defines how wide (in columns) a box or line should extend from the top left position. The width includes the edges of the box which are one space wide.

NOTE: SD_DRAW_HLINE is only useful to set the underline attribute.

B.3. SD_GET_IMAGE

This call transfers a copy of the current image buffer to a buffer defined in the program's WORKING STORAGE. This call is only defined for a 24 by 80 screen. If called on a larger screen only the first 24 rows by 80 columns will be returned and the cursor position will be reported as (line-1)*80 + (column-1).

The syntax is:

```
CALL "SD_GET_IMAGE" USING image-buffer
```

Where

image-buffer

Specifies a structure of the following format:

```
01  IMAGE-BUFFER.
    05  SCREEN-LINE         OCCURS 24 TIMES PIC X(80).
    05  FILLER              OCCURS 24 TIMES.
        07 CHAR-ATTRIBUTE   OCCURS 80 TIMES PIC 9(2) COMP.
    05  CURSOR-POSITION                    PIC 9(4) COMP.
    05 CURRENT-ATTRIBUTES                  PIC 9(4) COMP.
    05 SCREEN-BUFFER-RESERVED              PIC X(252).
```

char-attribute

Is defined as:

bit	attribute
1	DIM
2	BLINK
4	UNDERSCORE
8	REVERSED
16	Alternate character set

B.4. SD_GET_POS

This call provides the program the ability to determine the cursor's current position on the screen. This call is only defined for a 24 by 80 screen. If called on a larger screen when the cursor is beyond the 24 by 80 area the call will fail with an Exception Status 13.

The syntax is:

```
CALL "SD_GET_POS" USING cursor-position
```

Where

cursor-position

Specifies a PIC 9(4) COMP in which the cursor position is stored as (linenumber-1)*80 + (columnnumber-1). Thus when positioned to the home position (line 1, col 1), the cursor-position would be zero(0).

B.5. SD_MESSAGE, SD_ERROR_MESSAGE, SD_MESSAGE_ONLY

These calls provide the ability to display a message. SD_MESSAGE and SD_ERROR_MESSAGE wait for an operator to acknowledge the message, but SD_MESSAGE_ONLY does not.

The syntax is:

```
CALL "SD_MESSAGE" USING msg-string [, top-left-line, top-left-column
   [, label ]]

CALL "SD_ERROR_MESSAGE" USING msg-string [, top-left-line,
   top-left-column ]

CALL "SD_MESSAGE_ONLY" USING msg-string [, top-left-line, top-left-column
   [, label ]]
```

Where

msg-string
: Specifies a PIC X(n) and contains a message to be displayed within a box. The msg-string must be terminated with a null (LOW-VALUE). A bar | symbol will cause the message to wrap to a new line.

top-left-line
: Specifies a PIC 9(4) COMP and defines the top left line position of a box or line. If <= 0, it will be centered vertically.

top-left-column
: Specifies a PIC 9(4) COMP and defines the top left column position of a box or line. If <= 0, it will be centered horizontally.

label
: Specifies a PIC X(n) and defines a label to be placed on the top line of a box underlined and bright. The label must be terminated with a null (LOW-VALUE).

If neither line nor column is included or both are set to <= 0, the message box will be centered on the screen.

If label is not included, no label will be provided.

SD_ERROR_MESSAGE is equivalent to calling SD_MESSAGE with the label set to "Error!".

B.6. SD_NEW_WINDOW

This call saves the current image in a push-down image stack that can later be restored with the SD_REMOVE_WINDOW call. Each call to SD_NEW_WINDOW will cause the current image to be pushed onto the image stack and a new image buffer will start to receive all subsequent output to the screen.

Calling SD_NEW_WINDOW with the optional parameters is short-hand for an SD_NEW_WINDOW followed by an SD_DRAW_BOX.

The syntax is:

```
CALL "SD_NEW_WINDOW" [ USING top-left-line, top-left-column, height, width
    [, label ]]
```

Where

top-left-line
: Specifies a PIC 9(4) COMP and defines the top left line position of a box or line. If <= 2, it will be centered.

top-left-column
: Specifies a PIC 9(4) COMP and defines the top left column position of a box or line. If <= 0, it will be centered.

height
: Specifies a PIC 9(4) COMP and defines how high (in lines) a box or line should extend from the top left position. The height does not include the top line.

width
: Specifies a PIC 9(4) COMP and defines how wide (in columns) a box or line should extend from the top left position. The width includes the edges of the box which are one space wide.

label
: Specifies a PIC X(n) and defines a label to be placed on the top line of a box underlined and bright. The label must be terminated with a null (LOW-VALUE).

B.7. SD_POP_UP_MENU

This call provides the ability to display a simple pop-up menu. The user can use the up-arrow and down-arrow keys or the first letter of a selection to position to an item. A newline selects that item and exits the menu while an ESC exits the menu with no selection. The screen area under the pop-up menu is restored upon exiting from the menu.

The syntax is:

```
CALL "SD_POP_UP_MENU" USING menu-packet [, label ]
```

Where

menu-packet

Specifies a structure of the following format:

```
01 MENU-PACKET.
   05 MENU-LINE               PIC 9(4) COMP.
   05 MENU-COLUMN             PIC 9(4) COMP.
   05 DEFAULT-ITEM            PIC 9(4) COMP.
   05 SELECTED-ITEM           PIC 9(4) COMP.
   05 SELECTED-STRING         PIC X(30).
   05 MENU-ITEMS-STRING       PIC X(n).
```

label

Specifies a PIC X(n) and defines a label to be placed on the top line of a box underlined and bright. The label must be terminated with a null (LOW-VALUE).

The line and column arguments specify the top-left-line and top-left-column for the box containing the pop-up menu (just as documented under SD_DRAW_BOX). If these are set to zero, the menu is centered. The menu-items-string is the list of options to be displayed in the pop-up menu. This string should contain the string for each selection-item separated by a vertical bar (|) and ending with two vertical bars (||). For example:

```
"First selection|Second selection|Third selection||".
```

The default-item specifies the default item, i.e., the item to which the cursor will be positioned initially. The selected-item and selected-string are the returned item number and string when exiting the pop-up menu. If an ESC was hit, a zero(0) and spaces will be returned.

Within the pop-up menu, the user can use the up-arrow and down-arrow keys to position to the previous or next selection or the first letter of a selection to move to the next selection starting with that letter.

If a default timeout has been specified by SD_SET_ACCEPT_TIMEOUT and no input is entered within that time, the CALL returns with an Exception Status 76 "Device timeout", and the SELECTED-ITEM and SELECTED-STRING are set as if an ESC had been typed.

B.8. SD_POP_UP_MENU2

This call provides the ability to display a simple pop-up menu and accepts function keys as terminators. The user can use the up-arrow and down-arrow keys or the first letter of a selection to position to an item. A newline selects that item and exits the menu while an ESC exits the menu with no selection. The screen area under the pop-up menu is restored upon exiting from the menu. This function differs from SD_POP_UP_MENU in that it allows function keys to successfully select and return a menu option. The value of ESCAPE KEY is set to indicate which terminator was pressed. (The COBOL program can query this value with the “ACCEPT FROM ESCAPE KEY” statement.) Pressing ESC exits the menu with no selection, but it also updates ESCAPE KEY.

The syntax is:

```
CALL "SD_POP_UP_MENU2" USING menu-packet [, label ]
```

Where

menu-packet

Specifies a structure of the following format:

```
01 MENU-PACKET.
   05 MENU-LINE                PIC 9(4) COMP.
   05 MENU-COLUMN              PIC 9(4) COMP.
   05 DEFAULT-ITEM             PIC 9(4) COMP.
   05 SELECTED-ITEM            PIC 9(4) COMP.
   05 SELECTED-STRING          PIC X(30).
   05 MENU-ITEMS-STRING        PIC X(n).
```

label

Specifies a PIC X(n) and defines a label to be placed on the top line of a box underlined and bright. The label must be terminated with a null (LOW-VALUE).

The line and column arguments specify the top-left-line and top-left-column for the box containing the pop-up menu (just as documented under SD_DRAW_BOX). The menu-items-string is the list of options to be displayed in the pop-up menu. This string should contain the string for each selection-item separated by a vertical bar (|) and ending with two vertical bars (||). For example:

```
"First selection|Second selection|Third selection||".
```

The default-item specifies the default item, i.e., the item to which the cursor will be positioned initially. The selected-item and selected-string are the returned item number and string when exiting the pop-up menu. If an ESC was hit, a zero(0) and spaces will be returned.

Within the pop-up menu, the user can use the up-arrow and down-arrow keys to position to the previous or next selection or the first letter of a selection to move to the next selection starting with that letter.

If a default timeout has been specified by SD_SET_ACCEPT_TIMEOUT and no input is entered within that time, the CALL returns with an Exception Status 76 "Device timeout", and the SELECTED-ITEM and SELECTED-STRING are set as if an ESC had been typed. ESCAPE KEY is set to 99.

B.9. SD_READ_CHAR

This call allows the program to read a single keystroke with or without a timeout. Any 7-bit character read is passed through unchanged. If a timeout is given and no keystroke is received within that time frame, the character 128 is returned. If a function key is pressed a value from the following table is returned. Any 8-bit character has its high-order bit stripped, and the 7-bit value is returned.

Key	normal	Shift	Ctrl	Ctrl-Shift
F1	241	225	177	161
F2	242	226	178	162
F3	243	227	179	163
F4	244	228	180	164
F5	245	229	181	165
F6	246	230	182	166
F7	247	231	183	167
F8	248	232	184	168
F9	249	233	185	169
F10	250	234	186	170
F11	251	235	187	171
F12	252	236	188	172
F13	253	237	189	173
F14	254	238	190	174
F15	240	224	176	160
C1	220	216		
C2	221	217		
C3	222	218		
C4	223	219		
right-arow	24	152		
left-arrow	25	153		
up-arrow	23	151		
down-arrow	26	154		
home	8	136		
newline	10			
ESC	27			

The syntax is:

```
CALL "SD_READ_CHAR" USING char-field [, time-out-value ]
```

Where

char-field

Specifies a PIC X(1) and returns the read character if less than 8-bit, a 128 if a timeout occurred, or a number from the above table for a function key.

timeout-value

Specifies a PIC 9(4) COMP specifying the number of seconds to wait before terminating the READ. If not specified, the READ will wait forever. If set to 0 or >= 65535 then the timeout is set to wait forever, if set > 6300 it is set to 6300, otherwise if between 1 - 6300 it is set to that number of seconds.

If a timeout had previously been specified by SD_SET_ACCEPT_TIMEOUT, the new value will override the previous value for this call.

NOTE: The IC_GET_KEY builtin provides a more complete and terminal independent method of reading individual keystrokes.

B.10. SD_REDRAW

This call instructs the SCREEN HANDLER to clear the screen and redisplay the entire contents of the current image-buffer.

The syntax is:

```
CALL "SD_REDRAW"
```

B.11. SD_REMOVE_WINDOW

This call restores the image-buffer that is on the top of the image-buffer stack. SD_REMOVE_WINDOW effectively removes all data that had been displayed since the last SD_NEW_WINDOW call and replaces it with the data that was on the screen at that time.

A STOP RUN or a CALL PROGRAM to a new program will always clear all pushed image buffers.

The syntax is:

```
CALL "SD_REMOVE_WINDOW"
```

B.12. SD_RETURN_INPUT

This call provides the ability to place keystrokes into the terminal's input buffer such that the next ACCEPT will read them as if they had been typed from the keyboard.

Up to 256 characters can be returned, provided the internal input buffer is empty, i.e., characters are not still left from previous SD_RETURN_INPUT calls. If all the data will not fit into the input buffer, an Exception Status 241 is returned.

A STOP RUN will empty the input buffer.

The syntax is:

```
CALL "SD_RETURN_INPUT" USING data-string, string-size
```

Where

data-string

Specifies a PIC X(n) and holds data to be placed into the input buffer for this terminal. This item cannot be larger than 256 bytes. All entered data is treated like it came from a DG terminal. I.E., to enter a function key enter the 2-byte DG sequence even when on a non-DG terminal.

string-size

Specifies a PIC 9(4) COMP that specifies the number of bytes to use out of data-string. It must be less than or equal to the size of data-string.

To enter data into an empty 10 character field, the following could be used:

```
MOVE "1234567890" TO DATA-STRING.
MOVE 10 TO DATA-ARRAY(11).
MOVE 11 TO STRING-SIZE.
CALL "SD_RETURN_INPUT" USING DATA-STRING, STRING-SIZE.
```

Where

DATA-STRING

Specifies a PIC X(256).

DATA-ARRAY

Specifies a PIC 99 COMP array defined over DATA-STRING.

STRING-SIZE

Specifies a PIC 9(4) COMP.

Which would enter the characters "1234567890" followed by a newline into the next ACCEPT from the keyboard.

NOTE: The SD_RETURN_INPUT function is useful for returning information from a hotkey program into the field from which the hotkey was launched.

B.13. SD_SET_ACCEPT_TIMEOUT

This call provides the ability to set a default timeout value for all subsequent SCREEN HANDLER calls that read from the keyboard. These include SD_POP_UP_MENU, SD_POP_UP_MENU2, SD_READ_CHAR, SD_MESSAGE, SD_ERROR_MESSAGE, and SD_SYS_ERROR_MESSAGE. If no input is done for the specified number of seconds the input will be terminated with Exception Status set to 76. To disable timeout, a value of 65535 must be provided as the time-out-value. The initial default timeout value is forever (i.e., 65535).

This timeout only affects SCREEN HANDLER reads, normal I/O is not affected. The standard IC_SET_TIMEOUT does not affect any SCREEN HANDLER reads.

The syntax is:

```
CALL "SD_SET_ACCEPT_TIMEOUT" USING time-out-value
```

Where

time-out-value

Specifies a PIC 9(4) COMP specifying the number of seconds to wait before terminating a read. If set to 0 or >= 65535 then the timeout is set to wait forever, if set > 6300 it is set to 6300, otherwise if between 1 - 6300 it is set to that number of seconds.

NOTE: This call serves a different function than the SCREEN DEMON call.

B.14. SD_SYS_ERROR_MESSAGE

This call provides the ability to display a system error message as defined by the system. The message is displayed in a box with an optional user message and an operator acknowledge is sought. The *error-code* should be a valid Exception Status.

The syntax is:

```
CALL "SD_SYS_ERROR_MESSAGE" USING error-code [, msg-string
    [, top-left-line, top-left-column ]]
```

Where

error-code
: Specifies a PIC 9(4) COMP and should be a valid system Exception Status code.

msg-string
: Specifies a PIC X(n) and contains a message to be displayed within a box. The msg-string must be terminated with a null (LOW-VALUE). A bar | symbol will cause the message to wrap to a new line.

top-left-line
: Specifies a PIC 9(4) COMP and defines the top left line position of a box or line. If <= 2, it will be centered.

top-left-column
: Specifies a PIC 9(4) COMP and defines the top left column position of a box or line. If <= 0, it will be centered.

PART TWO - DEVELOPER'S GUIDE

X. INTRODUCTION TO THE DEVELOPER'S GUIDE

A. Overview

ICOBOL provides the ability to compile and execute COBOL programs in the UNIX and Windows environments. This allows the developer to use the most cost-effective platforms (UNIX or Windows) for both program development and program installation.

B. Operating Environment

B.1. General Concepts

The **ICOBOL** system has been designed to provide an application operating environment that works as consistently as possible among several different operating system environments. This consistency is expressed in a few key concepts that have their roots in the UNIX and Windows operating systems. If you are using one of these operating systems, the concepts may already be familiar to you.

B.1.1 Communication with the Operating System

The first concept is that programs communicate with their operating environment through three input/output streams or files: standard input (stdin), standard output (stdout), and standard error (stderr). Programs can read data to be processed from stdin, process it in some way, and write the results to stdout. They report errors to stderr. By default, most systems connect stdin to the console keyboard and both stdout and stderr to the console display.

Many utilities, especially in the COBOL environment, must process complex data files that do not fit this simple model and so they do not often use stdin for the data to process. However, the stdout and stderr files are still very useful. They allow the utility to logically separate error reporting from reporting the results of processing. For example, the ICSTAT utility reports statistics about an ICISAM files. It reports these statistics to stdout. If an error occurs, for example one of the command arguments does not exist, the error is reported to stderr.

B.1.2 I-O Redirection

The second concept is the ability to redirect I-O files from the default files to another file or device. The UNIX and Windows systems provide a very simple way to redirect these standard files in the command processor by using the special characters `<' and `>'. When stdout is redirected to a file, it provides a simple mechanism to capture the output of a utility. See your operating system command processor documentation for more on this concept.

B.1.3 Environment Variables

The third major concept is the ability to customize the operation of specific programs by setting information in items called Environment Variables. Environment variables have a name and a value like program variables or data items. The difference is that these variables are managed by the command processor. The utility programs can ask the operating system whether a particular environment variable is set or not, and what its value is. They are most often used to set default operating options, or the locations of important files. For example, all **ICOBOL** command-line programs look for the environment variable ICROOT as the base directory for finding the system files nd help files. UNIX and Windows both provide environment variables. ICROOT and other common environment variables used by **ICOBOL** are described in more detail beginning on page 696.

Environment variables are maintained in the command processor (or shell). Environment variables are set up with a command like:

On Windows	On UNIX
`SET ICROOT=C:\Program Files\Icobol`	`ICROOT=/usr/cobol360`

On UNIX, the **ICOBOL** software is installed in a directory with the name cobol*nnn*, where *nnn* corresponds to the revision level. For example, **ICOBOL** Revision 3.60 will be in a directory named cobol360 by default. This directory can be installed wherever is most appropriate or convenient for your system and can be renamed as needed. On Windows, the **ICOBOL** software is installed in a directory with the name icobol in the program files directory by default..

The main directory contains: all of the command-line programs, the readme file(s), and supplied COBOL executable programs. One subdirectory is called *help*. The help subdirectory contains help (.hf) files, for all the command-line programs defined as <command>.hf. Interactive programs have their own subdirectories under the help directory. There may be additional directories with other miscellaneous files, see the appropriate readme file(s) for a list of all the actual files.

Main Directory	Sub-Directories	Description
cobol<*rev*>	- cgicobol	Cgiruntime, scripts, examples
	- examples	Various examples
	- help	Help files (.hf)
	- link_kit	(Dev) Interface to install user-written subroutines into runtime
	- messages	Sample Message text files
	- print	Printer translation files (.pti)
	- term	Terminal description files (.tdi)
	- tcs	ThinClient server files
	- user_lib	(Dev) Interface to ICISAM files (ICAPI)for non-COBOL use
- main executables and needed files		

FIGURE 9. **ICOBOL** Directory Structure (UNIX)

Main Directory	Sub-Directories	Description
icobol	- cgicobol	Cgiruntime, scripts, examples
	- examples	Various examples
	- help	Help files (.hf)
	- link_kit	(Dev) Interface to install user-written subroutines into runtime
	- messages	Sample Message text files
	- print	Printer translation files (.pti)
	- qpr	(Dev) Gui-printer development (Formprint)(ICQPRW)
	- sentinel	Rainbow sentinel device files
	- sp2	(Dev) Gui-screen development (ICSP2)
	- term	Terminal description files (.tdi)
	- tcs	ThinClient server files
	- user_lib	(Dev) Interface to ICISAM files (ICAPI)for non-COBOL use
- main executables, .dlls, and needed files		

FIGURE 10. **ICOBOL** Directory Structure (Windows)

Command-line programs require the corresponding help file to be available in order to display their help text. If it is not available, an error message will be displayed that it could not find the help file. There are two methods for finding the help file: by using the ICROOT environment variable or by passing a partial pathname to the operating system.

B.3. ICEXEC Control Program

The **ICOBOL** system uses a control program called ICEXEC to coordinate multi-user access to system resources. The runtime system (ICRUN) requires the ICEXEC program to be running in order to operate. All other **ICOBOL** executables can operate with or without ICEXEC.

On UNIX, ICEXEC is required to provide an exclusive open capability since UNIX does not provide that capability. When ICEXEC is not running, an exclusive open is emulated by posting a write-lock on the whole file. A non-exclusive open posts a read-lock on the whole file. Thus, two programs can detect whether a file is opened or open-exclusively by using this mechanism. Care should be exercised when moving from no-ICEXEC to ICEXEC-running, as utilities that started in the no-ICEXEC mode will keep running in that mode until they terminate.

B.4. ICPERMIT License Program

The license manager, ICPERMIT, is used to provide licensing information to any executable that requires an authorization either on a single machine or over a TCP/IP-based network. This includes the **ICOBOL** compiler (ICOBOL), the runtime (ICRUN), ICNETD surrogates, the ICsp2 editor, the ICQPRW editor, sp2 runtimes, ICODBC driver, programs built with the user library, the ICIDE, and the remote logging surrogate (ICLOGS). ICPERMIT must be running and authorizing the proper license before any of these programs can operate. Otherwise, an error message occurs, stating the program could not connect with the license server and so is not authorized to execute.

C. Command-line Conventions

Another aspect of providing a consistent system across multiple operating platforms is in the command-line interface. The command-line programs use a common command-line syntax across all platforms, and they adhere to the following standard conventions:

C.1. Switches

1) all switches are composed of a single letter or digit preceded by a hyphen (-) (or optionally a forward slash (/) on Windows);
2) the switches are order independent;
3) the switches ARE case sensitive;
4) lower-case switches imply an action or modification of an action, e.g., `-h' for help;
5) UPPER-CASE switches imply an action with a required argument that must follow with an intervening space, e.g., `-A audit.log' for setting up an auditfile called audit.log.
6) multiple lower-case switches can be combined with one hyphen, e.g., `-aew' for `-a -e -w'.

C.2. Conventions for Defining Syntax

The following shows how the various conventions for defining syntax are represented in the **ICOBOL** documentation:

Convention	Meaning
[ ]	Brackets enclose optional portions of a format. One of the options contained within the brackets may be explicitly specified or that portion may be omitted.
{ }	Braces enclosing a portion of a format means that one of the options contained within the braces must be specified.
|	Bar will be used to separate choices when multiple choices are allowed.
...	Ellipsis indicates that the previous item can be repeated one or more times.
italic-lower-case	Indicates a generic term representing a value that is defined as indicated.

TABLE 38. Common Command-line Syntax Conventions

C.3. Filename Case (upper or lower)

UNIX systems support case-sensitive filenames as opposed to Windows, where they are case-insensitive. All released **ICOBOL** on UNIX files are lower-case, which is in keeping with most UNIX systems. By default, the **ICOBOL** on UNIX runtime will convert all COBOL filenames, including program names, to lower-case before looking up that file in UNIX. Although **ICOBOL** on UNIX can support UPPER-CASE only or mixed-case, we recommend using only one case for filenames to ease portability to case-insensitive environments.

With this in mind, this document will still use upper-case names in the text for specific programs but will always use lower-case in examples and when showing what needs to be entered from the keyboard to run a program.

On UNIX, all examples assume the Bourne shell is being run.

D. Common Switches

D.1. Overall

There are several switches that are common to all **ICOBOL** command-line programs except for ICINFO. These are described in detail in the following sections and will be referenced later, in the discussions of each program.. The **ICOBOL** command-line switch processor scans all the command-line switches, checking for errors. Any errors

display an abbreviated startup banner (the program name and revision) to stdout before displaying the error message to stderr and then exiting with a non-zero exit code. If there are no errors to terminate processing prematurely, the common switches are processed. First, if the Help switch is given, an abbreviated startup banner and help text are displayed to stdout after which the program exits normally (i.e., no other switches or arguments are processed). Next, if the Audit switch is given, auditing is enabled. Finally, the Quiet switch, if given, is processed. The program then begins its specific processing by emitting a startup banner, consisting of the program name, revision level, system, and the copyright notice. When it finishes processing, it will emit a trailer message indicating that it is done.

D.2. Audit Switch

The Audit switch will be shown in the syntax as:

-a[:a|b|d|p|t|u] | -A *file*|*dir*[:a|b|d|p|t|u]

Where

- a Append. Do not truncate the file, just append to the current file.
- b Backup. If a previous log file (.lg) exists, rename it to *.lgb and then open a new .lg file. **On UNIX, this will break hard links.**
- d Date. Add date in the form of _YYYYMMDD before the .lg extension.
- p PID. Add pid in the form of _NNNN before the .lg extension.
- t Time. Add time in the form of _YYYYMMDDHHmmsshh before the .lg extension. (YYYY-year, MM-month, DD-day of the month, HH-hour, mm-minute, ss-second, hh-hundredths of seconds.)
- u Username. Add username in the form _name before the .lg extension.

NOTE:

1) On Windows, the option "-A c:a" will be treated as open file "c" in append mode in the current directory. Previously this would have been open file "a" in the current directory of drive C:. To get the old behavior, enter

   "-A c:.\a"

The audit flags (a,b,d,p,t,u) instruct the Audit processing to take a different action then the default for the audit file. The default action is the same as usual, truncate the file to zero on startup.

Note that:

- -a Audit to the default file for this command.
- -A *file* Audit to the specified file.
- -A *dir* Audit to default file in the specified directory.

Audit files contain a copy of any output that was sent to either stdout or stderr, in the same order as it was emitted at execution time (i.e., it may be interspersed). The programs handle this internally, so stdout and stderr can still be redirected. The audit file can be specified to use the default name in the current directory (-a), a user specified name (-A *file*), or the default name in a specified directory (-A *dir*). An audit file is always created if it does not already exist. If it does exist, it is truncated to zero, unless the ‘a’ option on the audit switch is used (e.g., ‘-a:a’).

The default audit file name is <command>.lg.

D.3. Quiet Switch

The Quiet switch will be shown in the syntax as:

-q

The Quiet switch works by suppressing all output that is emitted to stdout. The most obvious effect is that it suppresses the usual banner and trailer messages that are emitted to stdout as the program starts and terminates. Because it is suppressing stdout, the Quiet switch may also suppress other parts of the usual output.

D.4. Help Switch

The Help switch will be shown in the syntax as:

-h|-?

The Help switch displays a summary of the command-line syntax, the switches and what they do, and the applicable environment variables.

E. Filename Extensions

ICOBOL requires that extensions for certain types of files match those in the following table except for those marked *defacto*. Those marked *defacto* are only the most commonly used extensions for these purposes and are not required. All **ICOBOL** release files will conform to these *defacto* standards.

d Those extensions marked as this sentence is marked are extensions in some older revision of **ICOBOL** or ICHOST
d but are handled in some special cases by current **ICOBOL** utilities.

Common extensions used by **ICOBOL** include:

.cd	Old ICHOST COBOL program file
.cf	Old Configuration file (pre-3.30)
.cfi	Configuration file
.cl	Library file
.co	COBOL Source program (ANSI card format) (defacto)
.cx	COBOL Program file
.er	Error file (defacto)
.fa	File attribute file
.fp	Failsafe protection file
.hf	**ICOBOL** help files
.icp	ICIDE project files
.lg	Audit / Log file (defacto)
.lgb	previous Audit / Log file
.lk	Link file
.ls	List file (defacto)
.ms	Message file
.od,.nt	Pair of files, ICPACK data and index temporary files
.pd,.dd	Pair of files, older revision COBOL program file (program and data) (pre-ICOBOL 2)
.pq	Printer control file
.pt	Old Printer translation file (pre-3.30)
.pti	Printer translation file (.ini format)
.sd	ICRUN Sort data file (temporary)
.sr	COBOL Source program (free-form format) (defacto)
.st	ICRUN Sort tag file (temporary)
.sy	COBOL Symbol table file
.td	Old Terminal description file (pre-3.30)
.tdi	Terminal description file (.ini format)
.tmp	Temporary file (defacto)
.xd,.nx	Pair of files, COBOL ICISAM file (date and index portion)
.xdb	ICODBC database definition file (.ini format)
.xdt	ICODBC table definition file (.ini format)
.xl	Log file
.xlg	Generation log file

TABLE 39. Common Filename Extensions used by **ICOBOL**

On UNIX, all **ICOBOL** utilities support mixed-case filenames. If a utility needs to add an extension, e.g., .xd/.nx, etc., it searches back from the end of the simple filename for the first alphabetic character. If it finds an upper-case alphabetic, it will use an upper-case extension, otherwise a lower-case extension is used. For example "iccheck DATAbase1" and "iccheck 12345" would use the lower-case extensions `.xd' and `.nx' for the ICISAM file, while "iccheck dataBASE52" would use the upper-case extensions `.XD' and `.NX'.

F. Exit Codes

All command-line programs return exit codes that provide an indication of the success or failure of the program. These are returned through the appropriate OS-specific mechanism (e.g., into ERRORLEVEL on Windows and the exit code on UNIX). In general, the following codes will be returned:

Exit code	Description
0	The program completed without errors.
1	The program ran, but some items it processed had errors. For example, ICCHECK checked a series of files, and some of them were corrupt.
2	The program was running, but was terminated by an operator interrupt or external abort.
3	The program was running, but was terminated by some fatal internal error. For example, the compiler was running but detected that its virtual memory manager had run out of memory unexpectedly.
4	There were command-line errors and so the program did not perform any of the requested function(s).
5	The user was not authorized to execute the program or perform a requested operation, so the program did not run.
6	The program experienced an error during its initialization phrase and could not execute. For example, it could not allocate sufficient memory to perform its function.
7	Help was requested
8-9	Reserved for future `common' errors.
10	These codes are specific to each program and will be documented with each program.
NOTE: All of the programs support exit codes 0 through 9 with the meaning described above.	

G. Common Environment Variables

G.1. Overall

There are several common environment entries that most command-line programs use. These are described in detail in this section, which will be referenced in each section describing the **ICOBOL** command-line programs. Other environment variables that are more program specific will be described under each **ICOBOL** command-line program.

All **ICOBOL** command-line programs accept an environment variable specific to themselves called uppercase-command-line-program-name. These specific environment variables can be used to set up options that are always used.

G.2. ICROOT

ICROOT specifies the **ICOBOL** root directory. ICROOT is used to find needed files and subdirectories like the help directory, print directory, and the term directory.

The syntax is:

```
ICROOT=dir
```

Where

dir

Specifies the directory where to find the **ICOBOL** help and term directories. Usually this should be set the current revision directory.

If ICROOT is not set, the current directory is used.

G.3. ICTMPDIR

ICTMPDIR specifies a directory to which programs may write any temporary files.

The syntax is:

```
ICTMPDIR=dir
```

Where

dir

Specifies a valid pathname for the directory in which any needed temporary files are to be written.

If ICTMPDIR is not set, the current directory is used.

Some of the programs that look for the ICTMPDIR environment variable are the **ICOBOL** compiler, ICLIB, ICRUN, ICSORT, and ICREV.

On Windows in particular, ICTMPDIR should be used when the current directory is a network drive to set the temporary directory to be on a local drive to improve performance.

G.4. Executable-Name Environment Variable

All command-line utilities support an environment variable of the same name as the utility in upper-case. For example, the 'iccheck' utility will recognize the variable ICCHECK. The environment variable can contain command line options for the utility which will be processed prior to any options actually present on the command line. If such an environment variable is present, the utility will display the complete set of options at startup.

G.5. TZ (Windows only)

On Windows, *TZ* specifies the time zone and number of hours past Greenwich mean time (GMT) for this location.

The syntax is:

```
TZ=tttn[ttt]
```

Where

ttt

Specifies a time zone of three letters. The second time zone should be given if daylight-saving time applies at this location.

n

Specifies a positive (west) or negative (east) integer number of hours difference from Greenwich mean time (GMT). Up to two digits can be specified.

If no TZ is specified, **ICOBOL** assumes all times are Greenwich mean time (GMT). If the second time zone is specified, **ICOBOL** assumes that daylight-saving time starts and stops based on the same schedule as used in the USA.

An example for Raleigh, North Carolina, USA would be:

```
SET TZ=EST5EDT
```

TZ is used for the command-line programs to accurately report date and time, and to accurately set date and time information in file headers. It sets the time zone and number of hours past Greenwich mean time (GMT) for this location.

XI. COMPILER (ICOBOL)

A. Overview

The **ICOBOL** Compiler (ICOBOL) is available for the UNIX and Windows environments. The compiler works the same in all environments except as stated in this manual. The following sections describe the requirements for the various operating environments.

The **ICOBOL** compiler provides the following features:

- a number of compile-time optimizations, including the detection and elimination of unreferenced data and unused code
- warnings about non-standard features or misuse of **ANSI 85** features when compiling for ANSI COBOL 74
- the ability to select the **ICOBOL** dialect (**ANSI 74**, **ANSI 85**, or ***VXCOBOL***)
- enhanced compilation performance
- the ability to create a cross-reference listing

The **ICOBOL** compiler requires an **ICOBOL** Development license to be available from the license manager (ICPERMIT).

To execute the **ICOBOL** compiler, an ICPERMIT must be installed and be running. Please see your Installing and Configuring **ICOBOL** on UNIX, or Installing and Configuring **ICOBOL** on Windows manuals on how to install and use ICPERMIT to allow the **ICOBOL** compiler to be authorized.

The **ICOBOL** compiler generates a COBOL executable file with the .cx extension. This .cx file when used in conjunction with the runtime system (icrun) allows the COBOL program to be executed in any environment in which the runtime is available.

B. Syntax

The syntax for the **ICOBOL** compiler is:

```
icobol [-a[:aflag]|-A file|dir[:aflag]] [-B 1|2|4] [-c] [-C copydir]...
   [-D ic|vx|85] [-e|-E erdir] [-F f|c] [-G {6|a|b|d|e|g|h|i|n|p|q|s}...]
   [-h|-?] [-H cnt] [-i] [-I {g|m|p|x}...] [-l|-L lsdir] [-M dddir]
   [-N {h|p|s|u}...] [-o|-O rev] [-P cxdir] [-q] [-R rev] [-s] [-w]
   [-X "string"] [-Z sydir] { infile }...
```

Where

-a[:*aflag*] or -A *file*|*dir*[:*aflag*] (Audit)
: Enables auditing (default icobol.lg). Where *aflag* is a|b|d|p|t|u. *Aflags* are a-append, b-backup, d-date, p-pid, t-time, u-username.

-B 1|2|4 (Byte alignment)
: Where to align 01 & 77 level items. Options are: 1-byte, 2-byte, or 4-byte. Default is 2-byte.

-c (Copy source directory)
: Add the directory of the main source file to the COPY list.

-C *copydir* (Copy directory)
: Add *copydir* to COPY searchlist. A maximum of 16 directories can be added this way.

-D ic|vx|85 (Dialact)
: Select **ICOBOL** dialect. Dialects are:
  ic=icobol (traditional **ANSI 74** with extensions),
  vx=icobolvx (***VXCOBOL***),

85=strict (**ANSI 85**).

Default is ic.

-e | -E *erdir* (Error)

Specify error file. Redirect messages to *infile*.er for -e, or to *erdir/infile*.er for -E.

-F c|f (Format source)

Select source format. Options are c=card or f=free-form. Default is f.

-G {6|a|b|d|e|g|h|i|n|p|q|s}... (General)

General switch allows various options to be specified. Multiple options may be specified. Options are:

6=V6 ICISAM
a=ANSI (***VXCOBOL***)
b=COMP size check by bytes (**ANSI 85 and *VXCOBOL***)
d=compile debug lines
e=ISO screen behavior
g=GO TO from/to/among declaratives is warning
h=require ANSI SEARCH ALL rules (***VXCOBOL***)
i=imply DUPLICATES
n=allow <nnn> lits
p=COMP size check by PIC (**ANSI 74**)
q=allow ***ISQL*** support
s=single-key is ICISAM (***VXCOBOL***)

-h | -? (Help)

Display help text.

-H *cnt* (Hard error limit)

Halt compile of each program after *cnt* errors. Default is to compile till end-of-file or a Fatal Error.

-i (Info)

Put out messages of category "information"

-I {g|m|p|x}... (Include in listing)

Include in listing options. Multiple options may be specified. Options are:

g=use global line numbers,
m=show metacode & pc,
p=show only the pc,
x=cross reference.

-l | -L *lsdir* (Listing file)

Specify listing file. Produce a listing in *infile*.ls for -l, or to *lsdir*/*infile*.ls for -L.

-M *dddir* (Make ICODBC data definition files)

Create ICODBC definition files: *dddir**.xd[t|b]. Works with the ICODBC options (-X *string*) switch.

-N {h|p|s|u}... (NO)

NO options. Multiple options may be specified. Options are:

h=No - to $ translation
p=No check or recovery for missing period
s=No space needed in comma or semicolon separators
u=No USE, INVALID KEY or AT END required.

-o|-O *rev* (OEM version)

Set OEM version in .cx to be the compiler's version(-o) or 'rev' (limit 8 characters).

-P *cxdir* (Program files)

Specify location of .cx program files in *cxdir**infile*.cx

-q (Quiet)

Specify quiet operation.

-R *rev* (Revision)

Specify the code revision to be compiled. Valid revisions are 1 (3.0x), 2 (3.2x), 3 (3.4x), or 4 (3.5x). The default is 4.

-s (Stats)

Put out statistics.

-w (Warnings)

Put out Warning messages.

-X "*string*" (ICODBC options)

Options for ICODBC definition file creation.

-Z *sydir* (Debug)

Causes the symbol files needed for debugging to be generated. The program symbol file (infile.sy) will be created in the directory given by *sydir*. The path for the symbol file(s) will also need to be passed to the runtime using the same switch value or -z for the current directory.

infile

is one or more COBOL source files or a template representing source files to compile.

Options for -X "string" above:

-F 1|2|3|4 (ICODBC Format)

Column format:

1=exactly as in source,
2=exactly as in source with '_' replacing hyphen,
3=initial caps after hyphens with hyphens removed,
4=initial caps after hyphens with '_' replacing hyphen. (default is 3)

-G p (ICODBC General)

General options: p=COMP items have precision based on size.

-I f (ICODBC Include)

Include options: f=filler items are included.

-L *min*:*max* (ICODBC Level numbers)

Include only items with level numbers between *min* and *max*.

-n (ICODBC Not overwrite)

Do not overwrite existing .xdb and .xdt files.

-N {g|n|r|s}... (ICODBC No)

NO options:

g=no group items are included,
n=no RENAMES (level 66) items included,
r=no REDEFINES items included,
s=no secondary record definitions included.

{-P *old*[:*new*]}... (ICODBC Prefix)

Replace the prefix '*old*-' with '*new*-' in column names. If '*new*' is not specified, remove '*old*-' . Multiple ICODBC Prefix (-P) switches may be specified.

B.1. Rules

(a) The compile switches may be specified in any order.

(b) For the General switch (-G) and No switch (-N), one or more of the option values may be specified and they may be specified in any order.

(c) On Windows, the `/' can be used in place of the `-' as long as it is used consistently throughout the command-line.

(d) For *infile*, if the source name does not have an extension and a file by that name is NOT found, an extension of the appropriate case is appended. First, a `.co' or `.CO' extension (`.cob' or `.COB' if using ***VXCOBOL***) is appended and sought, but if that name is NOT found, a `.sr' or `.SR' extension is appended to the original filename and sought.

NOTE: If both foo.co and foo.sr exist in the same directory, specifying

```
icobol foo
```

will only compile 'foo.co' since the .co extension is sought first.

(e) On UNIX, the extension case is determined based on the last alphabetic character in the simple part of the filename. If the last alphabetic character is upper-case, then the extension will be upper-case, otherwise a lower-case extension will be used. The recommendation is to always use lowercase filenames.

(f) If multiple source files are given, each file is compiled separately as though it was the only source file given.

(g) The input filename can specify a template. The valid template characters are `?' and `*' and match any one character or series of characters, respectively.

(h) If the **ICOBOL** compiler detects an error while compiling, no program file (.CX) is generated and the compiler returns with a non-zero exit code when it terminates. If there is an existing program file, it is not modified.

(i) On UNIX, copy files are always sought in lower-case. If files must be converted from upper-case to lower-case the makelow script, in the examples sub-directory of the release, can be used.

B.2. Environment Variables

The **ICOBOL** compiler looks for the following environment variables in addition to ICROOT and ICTMPDIR:

ICOBOL — sets standard switches. The contents of ICOBOL are treated like switches from the command line that are processed before the command line. The environment variable may only contain switches, no input file arguments. Since the contents of the environment variable contains spaces, it must be enclosed in quotes in most UNIX shells.

For example to compile using the strict **ANSI 85** dialect (-D 85), always produce a listing (-l), include warnings (-w), and search *copy_dir* for COPY files, the environment variable ICOBOL can be set as follows:

On Windows:

```
> SET ICOBOL=-D 85 -l -w -C copy_dir
```

On UNIX with the Bourne shell:

```
$ ICOBOL="-D 85 -l -w -C copy_dir"
$ export ICOBOL
```

This setup of the ICOBOL environment variable can be included in your AUTOEXEC.BAT, in the 'Environment' tab of the System Properties sheet, or in your .profile file on UNIX.

C. Switches

C.1. Overview

All **ICOBOL** compiler switches start with a dash `-' followed by the switch with no spaces. Each individual switch must be separated by a space or spaces from the other switches. Switches are case-sensitive. On Windows , the slash `/' can be used in place of the dash as long as it is used consistently throughout the command line.

The general standard for switches is that a lower-case switch is only an ON or OFF switch. An UPPER-CASE switch implies that an additional argument, delimited by spaces, follows this switch.

In addition to the following switches, which are unique to the **ICOBOL** compiler, also see the Common Switches section, beginning on page 692.

C.2. Byte Alignment Switch (-B 1|2|4)

This switch instructs the compiler to align 01 level and 77 level items on a specific boundary. The valid values are 1, 2, and 4. A value of 1 causes the compiler to allocate 01 or 77 level items on a byte boundary, i.e., there is no specific alignment. A value of 2, the default, causes the compiler to allocate the items on a 2-byte (word) boundary. A value of 4 causes the compiler to allocate the items on a 4-byte (double-word) boundary. As a general rule, the default value is recommended unless a specific alignment is needed in order to interface with a linked-in routine.

C.3. COPY Sourcedir Switch (-c)

This switch specifies to the compiler to add the source directory for the main source file to the list of directories that will be searched for COPY files if the file is NOT found in the current working directory. This directory is added IN FRONT of the directories specified by the -C dir switch.

C.4. COPY Path Switch (-C *copydir*)

This switch specifies to the compiler that, in addition to the current directory, the specified *copydir* directory will be searched for any COPY files with non-full pathnames (simple or relative). I.E., both the names *"source1"* and *"dir1\source1"* would be sought using the extra copydir's, but *"\dir1\source1"* would not since it has a full-pathname. Up to sixteen directories can be specified in this manner with the -C switch in front of each directory. For example:

```
-C directory1 -C directory2 -C directory3 -C directory4
```

would specify four additional directories to search for COPY files after looking in the current directory. The directories will be searched in the order specified on the command line.

The name specified as the input file is not sought along the COPY Path set of directories.

C.5. Dialect Switch (-D ic|vx|85)

This switch selects the **ICOBOL** dialect. Valid dialects are:

ic
: The fundamental dialect. It is consistent with traditional **ICOBOL**, and uses **ANSI 74** file status codes and file-handling semantics. This is the default.

vx
: This dialect is consistent with the syntax and semantics used by Data General's AOS/VS COBOL and by Envyr Corporation's ***VXCOBOL***.

85
: This is the stricter **ANSI 85** dialect. It is consistent with **ICOBOL**2 code compiled with the (now obsolete) -M 85 option. It uses **ANSI 85** file status codes and file handling semantics.

C.6. Error File Switch (-e | -E *erdir*)

The -e and -E switches are mutually exclusive.

The -e switch specifies that the name of the error file is to be the name of the source program (*infile*) with the `.er' extension (i.e., *infile.er*). If *infile* has an extension, the extension is removed before `.er' is appended. When used with multiple source files or a template, each individual source program will have its own error file.

The -E *erdir* switch specifies that the error file should be the source file with the `.er' extension and that the file should be placed in the directory specified by *erdir*.

C.7. Format Switch (-F c|f)

This switch specifies the format of the source program being compiled.

Where

- c is ANSI Card format
- f is Free-form format (also known as CRT format)

All COPY files in the program must have the same format as the program that uses them.

C.8. General Switch (-G {6|a|b|d|e|g|h|i|k|n|p|q|s}...)

This switch provides a mechanism to enable a particular enhanced feature of the **ICOBOL** compiler. The switch options that are available are:

- 6 (Version 6 ICISAM) Use ICISAM version 5/6 limitations (4 alternate keys, 100 byte keys, and 4096 byte records) instead of version 7 limitations. This option is useful ONLY if you are using ICISAM version 5 or 6..
- a (ANSI) (***VXCOBOL***). Causes COMP items to be stored based on picture, IS NUMERIC test is strict ANSI (item must be USAGE DISPLAY and spaces are not allowed), and OPEN OUTPUT of a sequential file will delete and recreate.
- b (COMP size check by bytes) (**ANSI 85 and *VXCOBOL***) This is the default for **ANSI 74**. By default, **ANSI 85 and *VXCOBOL*** size checks by picture.
- d (With DEBUGGING) causes the compiler compile all debugging lines. If not given, debugging lines are treated as comment lines.
- e (ISO screen behavior) causes the compiler to invoke ISO screen behavior. This option is useful when migrating to **ICOBOL** from certain other COBOL products. When specified on the compilation, runtime behavior is altered as follows: (1) BLANK LINE erases the entire line, (2) ERASE LINE erases from the cursor to the end of the line, and (3) ERASE SCREEN erases from the cursor to the end of the screen.
- g (GO TO from/to/among declaratives is warning) Normally this is an error and we recommend against the use of this option.
- h (Require ANSI SEARCH ALL rules) (***VXCOBOL***) Requires that the SEARCH ALL statement conform to ANSI syntax. Without this switch, SEARCH ALL syntax is identical to SEARCH.
- i (Imply DUPLICATES) Specifies that the ALTERNATE KEY clause should ALWAYS imply the WITH DUPLICATES phrase.
- n (Numbers) (***VXCOBOL***) allow the constructs <nnn> or <onnn> or (***ANSI 74/85***) allow the constructs <nnn>, <onnn>, <dnnn>, and <xnn> in nonnumeric literals to specify a byte value represented by the nnn or nn numbers. In the case of <*nnn*> and <o*nnn*>, *nnn* represents an octal value, in <d*nnn*> *nnn* represents a decimal value, and in <x*nn*> *nn* represents a hex value. Upper and lower case `o', `d', and `x' can be used to specify octal, decimal, or hex. In hex mode, upper and lower case `a' - `f' can be used. The value for any byte must be in the range 0 - 255 (decimal). For octal and decimal no more than three digits can be specified and for hex no more than two digits can be specified. <1> is treated as <001>. The construct << can be used to enter a single < when the General number switch (-G n) has been specified. Only one byte can be specified per <> pair.
- p (COMP size check by PIC) (***ANSI 74***) This is the default for ***ANSI 85* and *VXCOBOL***. ***ANSI 74*** size check by bytes.
- q (***ISQL*** Support) Allow Integrated SQL support, including SQL data types and new SQL statements. Enables the ***ISQL*** feature-set.
- s (Single-key is ICISAM) (***VXCOBOL***) For ***VXCOBOL***, the default for a single-key indexed file is to use INFOS. Specifying this switch will cause ICISAM files to be used.

C.9. Hard Error Limit Switch (-H *cnt*)

This switch provides a mechanism to instruct the compiler to stop compiling a file after a certain number of errors are encountered. Normally the compiler continues to process the file until a Fatal Error is encountered or the end-of-file is reached. Valid values for *cnt* can be from 1 to 65535.

For example, if "-H 10" were given on the compile line then after the tenth error is encountered a message would be given that the maximum number of errors has been reached and that this compile is terminating.

Cnt is only for a single compile. If multiple compiles are being done, the error counter is reset at the start of each compilation.

C.10. Information Switch (-i)

This switch causes the compiler to display all information messages. The default is to not display information messages.

C.11. Include listing options Switch (-I {g|m|p|x}...)

This switch allows the programmer to select from the following options to include in the compiler output listing. One or more options may be specified in any order.

Where

- g Specifies that global line numbers be included in the output listing. By default, each copy file starts a new set of line numbers for that source.
- m Specifies that metacode and pc be included in the output listing.
- p Specifies that the pc be included in the output listing.
- x Specifies that a cross-reference be included in the output listing.

If not specified, none of the above is done. If the Listing File Switch is not specified, it is implied.

C.12. Listing File Switch (-l | -L *lsdir*)

The -l and -L switches are mutually exclusive.

The -l switch specifies that the name of the list file is to be the name of the source program (*infile*) with the `.ls' extension (i.e., *infile.ls*). When used with multiple source files or a template, each individual source program will have its own list file.

The -L *lsdir* switch specifies that the list file should be the source file with the `.ls' extension and that the file should be placed in the directory specified by *lsdir*. When used with multiple source files or a template each individual source program will have its own list file in the *lsdir* directory.

The listing file will show any dialect, format source, or compile options. Each source line will be shown with its line number, whether the line came from a COPY file (c), a flag character (<, >, or *), and a space preceding each actual source line.

The results of a COPY ... REPLACING statement are clearly shown. Lines (or parts of lines) which are removed are indicated with a < next to the line number. Lines (or parts of lines) which are being inserted are marked with a > next to the line number. For each replacement, the listing will show the text word or words being removed (<) and those, if any, being inserted (>).

The following example shows a COPY file included without any replacement (at line 23) and the same copy file (included at line 25) with 3 items replaced. Note that the positions of the inserted text are displayed at the location in which they appeared in the COPY statement.

```
23 <     COPY "TESTFILE.FD".
 1c  FD TESTFILE.
 2c  01 TESTFILE-REC.
 3c     04 FILLER                 PIC X(25).
 4c     04 TESTFILE-STUFF         PIC X(25).
24
25 <     COPY "TESTFILE.FD" REPLACING TESTFILE BY TESTFILE2
26 <             TESTFILE-REC BY TESTFILE2-REC
27 <             TESTFILE-STUFF BY TESTFILE2-STUFF.
 1c  FD
 1c<    TESTFILE
   >                                              TESTFILE2
 1c             .
 2c  01
 2c<    TESTFILE-REC
   >                             TESTFILE2-REC
 2c                .
 3c     04 FILLER                 PIC X(25).
 4c     04
 4c<       TESTFILE-STUFF
    >                               TESTFILE2-STUFF
 4c                                PIC X(25).
```

C.13. Make ICODBC Data Definition Files Switch (-M *dddir*)

This switch creates ICODBC definition files: *dddir*\ *.xdt and *dddir**.xdb. The -X *string* switch is used to pass options for creating ICODBC definition files. See the ICODBC Section later in this chapter (starting on page 717), for more information on this support. This option requires symbol files so if no Debug Switch is specified, the symbol files (.sy) are placed in the same *dddir* directory.

C.14. No Switch (-N {h|p|s|u}...)

This switch provides a mechanism to disable particular default features of the **ICOBOL** compiler. Valid option values for the No switch are:

- h (No dollar signs) do not replace "-" with "$" for a generated external name as specified in the Language Reference manual, in the Default Filenames table that is included in the section describing the ASSIGN clause of the SELECT statement.
- p (No check or recovery for missing Periods). In certain cases, the compiler attempts to recover from missing periods. This option suppresses that behavior.
- s (No check for Space) no space needed in comma or semicolon separators.
- u (No **U**SE, INVALID KEY, or AT END required). Certain statements require a coded method of error checking. Use this option only if you code error checks in-line after each statement.

To disable any of these features, use the No switch (-N) followed by a space and then the switch values, in any order, of the features to be disabled (without spaces).

For example, ' -N hp':

(1) will NOT replace "-" with "$" in generated external filenames, and
(2) will NOT check for or fix missing periods.

C.15. OEM Version Switch (-o | -O *rev*)

The -o switch instructs the compiler to set the OEM version in the .cx file to be the compiler's version.

The -O switch instructs the compiler to set the OEM version in the .cx file to be '*rev*' (limited to 3 characters).

C.16. Program Output File Switch (-P *cxdir*)

This switch specifies that the .cx program output file(s) should be placed in the directory specified by *cxdir*.

C.17. Revision Switch (-R 1|2|3|4)

The -R 1 switch instructs the compiler to allow only syntax that generates code of the specified .CX revision or less. The compiler will not allow syntax that generates code above the revision level specified. This switch is useful to generate program files to run on older systems. The compiler will set the .CX revision to the specified value when generating code.

The -R 1 switch instructs the compiler to produce revision 1 .CX files (**ICOBOL** 3.00).

The -R 2 switch instructs the compiler to produce revision 2 .CX files (**ICOBOL** 3.20). Revision 2 supports new functionality for ACCEPT and DISPLAY.

The -R 3 switch instructs the compiler to produce revision 3 .CX files (**ICOBOL** 3.40). Revision 3 supports the ***ISQL*** data types and statements (when used with the -G q switch) along with some minor performance enhancement opcodes. Programs compiled at this level require at least a 3.40 runtime in order to execute, even if no new opcodes are generated. If you must be able to run a program on a previous version of the runtime make sure that the revision is set correctly.

The -R 4 switch instructs the compiler to produce revision 4 cx files **(ICOBOL** 3.50) Revision 4 cx files support up to 255 operands in a FETCH statement. This will now allow 255 columns to be fetched. The previous limit was 100.

If no -R switch is given, -R 4 is the default.

C.18. Statistics Switch (-s)

This switch instructs the compiler to put out statistics that include the a) start and stop time, b) number of lines and lines per minute, c) the number of errors, warnings, and information messages encountered, and d) a blank line for each individual source compiled.

C.19. Warnings Switch (-w)

This switch instructs the compiler to put out Warning messages. The default is no Warnings.

C.20. ICODBC Options Switch (-X "*string*")

Options for ICODBC definition files are in "*string*". Valid options are:

-F 1|2|3|4 (ICODBC Column format)
This option specifies how column names are to be created from the COBOL data names. Options are:
1=exactly as in source,
2=exactly as in source with '_' replacing hyphen,
3=initial caps after hyphens with hyphens removed,
4=initial caps after hyphens with '_' replacing hyphen
(default is 3)

-G p (ICODBC General options)
p=COMP items have precision based on size, rather than picture

-I f (ICODBC Include options)
f=filler items are included (i.e., a column definition is created for each FILLER item)

-L *min*:*max* (ICODBC Level)

Only include column definitions for data items with level numbers between *min* and *max*: 1 <= *min* <= *max* <= 49. (Level 66 items must be explicitly excluded with the -N n option.) If no -L is specified, all levels are included.

-n (ICODBC No overwrite)

Do not overwrite existing .xdb and .xdt files

-N {g|n|r|s}... (ICODBC NO options)

Specifies data items which are NOT to be included in the column definitions. Valid options are:

g=no group items are included,
n=no RENAMES (level 66) items included,
r=no REDEFINES items included,
s=no secondary record definitions included

{-P old[:new]}... (ICODBC Prefix)

Replace the prefix 'old-' with 'new-' in column names. If 'new' is not specified, remove 'old-'. Multiple ICODBC Prefix (-P) switches may be specified.

These options are only used if the Make ICODBC Definition files switch (-M) is given. See the ICODBC Section later in this chapter (starting on page 717), for more information on this support.

C.21. Debug Switch (-Z *sydir*)

This switch instructs the compiler to generate the needed symbol file(s) for use when debugging. The symbol file (infile.sy) will be generated in the directory *sydir*. The symbol file information must be specified to the runtime using its -Z or -z switch also.

D. Messages

D.1. Overview

The **ICOBOL** compiler generates four levels of messages. These are:

(1) Fatal errors, which cause the compilation to halt.
(2) Errors, which cause the compiler to attempt to continue the compilation but no program files will be generated.
(3) Warnings, which will generally imply a construct that:
a) will not compile under 1.xx **ICOBOL** compilers,
b) is not standard in comparison to the **ANSI COBOL 85** standard, or
c) is ignored or has behavior which might not be expected.
Warnings will not suppress the creation of a program file.
(4) Information messages which will help the programmer to clean up his program. For example to indicate that a data item is never referenced or a piece of code is never executed.

D.1.1 Format

Messages from the compiler are shown in three different formats depending on where the message is being placed: the screen (stdout), the error file, or the listing file.

When coming to the screen (stdout) messages are displayed in a one line format as shown below :

```
** source-file (line-number, col-num): Msg-type: Msg-text
```

Where

source-file
is the fully resolved name of the appropriate source filename

line-num
is the local line number in the *source-file* of the condition.

col-num
is the column of the start of the token that caused the condition.

Msg-type
Is Fatal, Error, Warning, or Info.

Msg-text
Is the actual text for this particular message.

When going to the error file, messages are displayed in a five line format as shown below:

```
Line # of filename
<actual COBOL line from the source file>
a circumflex (^) is positioned at the place which caused the message to be generated
Msg-type:  Msg text
  (blank line)
```

Where

#
Is the appropriate local line number.

filename
Is the fully resolved name of the appropriate source file.

Msg-type
Is Fatal, Error, Warning, or Info.

Msg-text
Is the actual text for this particular message.

When going to the listing file, messages are displayed after the affected line of text in a three-line format as shown below:

```
<actual COBOL line from the source file>
a circumflex (^) is positioned at the place which caused the message to be generated
Msg-type:  Msg text
        (blank line)
```

Where

Msg-type
Is Fatal, Error, Warning, or Info.

Msg-text
Is the actual text for this particular message.

A blank line is always inserted to more easily allow the messages to be viewed.

D.1.2 Examples

Two examples of the messages are given below:

Example 1 to screen:

```
** C:\test200\logon.sr (1662,1): Info:   This item is never referenced.
```

Example 2 to screen:

```
** C:\test200\logon.sr (1185,24): Info:  Code was generated the same as 1.xx ICOBOL (which
resets the TALLY variable to zero).
```

Example 1 to error file:

```
Line 1662 of C:\test200\logon.sr
DETERMINE-TERMINAL.
^
Info:   This item is never referenced.
```

Example 2 to error file:

```
Line 1185 of C:\test200\logon.sr
INSPECT F-STR TALLYING LOW-CTR FOR ALL LOW-VALUES.
                       ^
Info: Code was generated the same as 1.xx ICOBOL (which resets the TALLY variable to zero).
```

D.1.3. Rules

(1) Information messages are displayed only when the Info switch (-i) is given.

(2) Warning messages are displayed only when the Warning switch (-w) is given.

(3) Fatal Errors and Errors are always displayed.

D.2. Error Messages

The **ICOBOL** compiler provides a level of messages for those features of **ICOBOL** that are in error. Generally errors result from improperly coded COBOL code where either a syntax rule or general rule has been violated.

There are some errors that can occur on a file that compiles correctly with previous (usually Data General) **ICOBOL** compilers. These generally are places where the previous **ICOBOL** compiler did not check for the error. This compiler will give an error in these cases and will not generate a program file. These must be fixed in order to compile successfully.

Some examples of these types of error are given below with the Message text that you will see on the Error line followed by a description of what causes the message:

1. This option may only be used with a group, TO clause, or USING clause.

   Some previous compilers allowed the AUTO clause on a FROM clause, although it makes no sense and was prohibited in the documentation.

2. The JUSTIFIED clause may only be specified with a FROM clause or USING clause.

   In the SCREEN Section, previous compilers ignored the JUSTIFIED clause on items with the TO clause even though it was prohibited in the documentation.

3. The JUSTIFIED clause may not be specified for a numeric or edited item.

   Some previous compilers ignored the JUSTIFIED clause on a numeric or edited item even though it was prohibited in the documentation and by the **ANSI COBOL 74** and **ANSI COBOL 85** standards.

4. A GO TO statement may not branch between declarative and non-declarative procedures.

   Previous compilers allowed this construct, even though it was prohibited in the documentation and by the **ANSI COBOL 74** and **ANSI COBOL 85** standards. This construct can eventually lead to a "Perform stack overflow" error at runtime, since a perform is pushed onto the stack when the declarative section is executed, but is never popped off again. (This error can be turned into a warning with the -G w switch, but this is not recommended.)

5. Syntax error. (on a PICTURE in a level 88)

   Previous compilers ignored the PICTURE clause on a level 88 even though it is prohibited in the documentation and by the **ANSI COBOL 74** and **ANSI COBOL 85** standards. The PICTURE clause should be removed.

6. This item must refer to an elementary integer data item.

   Previous compilers allowed an alphanumeric item to be used in a GO TO DEPENDING ON, even though it was prohibited in the documentation and by the **ANSI COBOL 74** and **ANSI COBOL 85** standards. This must be changed to an integer.

D.3. Warning Messages

When not running in **ANSI 85** mode (-D 85), the **ICOBOL** compiler provides a level of warning messages for those features of **ICOBOL** that do not meet the **ANSI COBOL 85** standard. These warnings are provided to encourage the programmer to clean-up and/or fix these areas such that the code will still work in the **ICOBOL** environment but meet **ANSI 85** requirements.

Some examples of these types of warnings are given below with the Message text that you will see on the Warning line, followed by a description of what causes the warning:

1. This item must be an elementary item, not a group item.

   In the flagged statement this particular item must be an elementary item. **ICOBOL** allows group items here if they are only 1-byte in length.

2. The RECORDING MODE clause is valid only for a SEQUENTIAL file. (Self explanatory.)

3. The composite of operands is greater than 18 digits.

   Previous compilers did not detect when more than 18 digits of total precision are used in an ADD, SUBTRACT, MULTIPLY, or DIVIDE statement. This compiler detects this. These should either be fixed or changed to COMPUTE statements, which do not check the precision.

   Example: An ADD of a PIC 9.9 to a PIC 9(18) would create a 19 digit precision. (i.e., PIC 9(18).9).

Previous compilers treated index-names and index data items just like a PIC 9(5) COMPUTATIONAL item. As a result, index-names and index data items can be totally misused from what the standard allows. The next several messages all pertain to the use (or misuse) of index-names, index data items, and/or the SET and MOVE statements.

4. The value must be an integer value greater than zero.

   Previous compilers allowed a SET index-name to 0. This is prohibited by the **ANSI COBOL 74** and **ANSI COBOL 85** standards since 0 is NEVER a valid occurrence number.

5. This index-name is not listed in a corresponding INDEXED BY list.

   Previous compilers allowed any index-name to be used to subscript any table. The **ANSI COBOL 74** and **ANSI COBOL 85** standards only allow the index-name to be used with the table with which it is associated in the INDEXED BY phrase.

6. The operand is the wrong class or type for the operation.

   The **ANSI COBOL 74** and **ANSI COBOL 85** standards define index-names and index data items to be separate types that are NOT numeric. The SET statement only allows specific combinations of index-name, index data item, and numeric data items or literals.

7. This is an invalid use of an index-name or index data item.

   Previous compilers allowed index-names and index data items to be used in arithmetic and MOVE statements. This is prohibited by the **ANSI COBOL 74** and **ANSI COBOL 85** standards.

8. A numeric item or index-name is required.

   This message usually appears in a PERFORM VARYING using an index data item, which previous compilers allowed, but which is prohibited by the **ANSI COBOL 74** and **ANSI COBOL 85** standards.

9. This item must refer to an elementary integer data item.

   Previous compilers allowed index-names and index data items to be used in a GO TO DEPENDING ON, even though it was prohibited in the documentation and by the **ANSI COBOL 74** and **ANSI COBOL 85** standards.

D.4. Information Messages

The **ICOBOL** compiler provides a level of messages that inform the programmer about certain aspects of the code or data of which he may not otherwise be aware. These messages also may indicate what the compiler is doing about a particular data item or code sequence. In most cases, these messages are about unreferenced data items and unexecutable code.

Some examples of these types of messages are given below, with the Message text that you will see on the Info line followed by a description of what causes the message:

1. This item is never referenced.

This message indicates that the level 01 data name (and every sub-item) or 77 data name or the paragraph name is never referenced. The compiler will not include these items in the program file.

2. This section is unreachable; the entire section has been eliminated.

3. This paragraph is unreachable; the entire paragraph has been eliminated.

4. This statement is unreachable; it (and possibly others following) has been eliminated.

5. This paragraph is the end of a PERFORM range, but the end of the paragraph is unreachable.

6. This word is a reserved word in some other compatibility mode.

   This message is given when a user-defined word has been defined that will conflict with the ***VXCOBOL*** Reserved Word list or with a Reserved Word in another COBOL compiler. Changing this word to a new name will allow for an easier future migration.

The first five of these messages say the particular piece of code can never be executed and thus is being eliminated from the program file. These messages can be used as a guide to detect unexecutable portions of code.

In addition, #5 implies that you could use a GO TO statement rather than a PERFORM, since the PERFORM will never return.

The following messages are generated as part of detecting invalid code versus the **ANSI COBOL 74** or **ANSI COBOL 85** standards. These messages can be moved to warnings by using the General Bad code switch (-G b).

1. Code was generated the same as 1.xx **ICOBOL**.

   On storing into a Signed COMPUTATIONAL item, **ICOBOL** does not account for the sign bit when detecting size error.

   Example: For a PIC S9(2) COMPUTATIONAL item (i.e., 1 byte) a store of 129 will succeed, but give the value -127. It should have been a size error since the value 129 will not fit in the item.

NOTE: This does not occur when the -G s switch is used, since size check is then based on the number of digits, not the binary value.

2. Code was generated the same as 1.xx **ICOBOL** (which generates incorrect code).

   This message is generated in the following cases:

   (a) on a comparison between a numeric integer and a figurative constant other than ZEROS (i.e., HIGH-VALUES, LOW-VALUES), 1.xx **ICOBOL** generates an alphanumeric to alphanumeric comparison, and the proper operation is to generate a numeric to alphanumeric comparison.
   (b) on a MOVE of a figurative constant other than ZEROS (i.e., HIGH-VALUES or LOW-VALUES) to a numeric or numeric edited item 1.xx **ICOBOL** generates an alphanumeric move. The proper operation is to generate a numeric to numeric or numeric edited move, treating the alphanumeric items as an unsigned integer.

3. Code was generated the same as 1.xx **ICOBOL** (which resets the TALLY variable to zero)

   ICOBOL (in ANSI 74 mode) always resets the tallying counter in an INSPECT statement to zero before it starts tallying. The standard says that the tallying counter is ONLY incremented, i.e., if a zero was in the variable when the INSPECT started it would execute just like 1.xx **ICOBOL**, otherwise the variable would come out with a different result.

4. Code was generated the same as 1.xx **ICOBOL** (which ignores S in the picture)

   ICOBOL accepts an S in the PICTURE clause of Screen Section entries and ignores it unless the SIGN IS clause is also specified. If the SIGN IS clause is absent, it is preferable to either remove the S, thereby making the entry unsigned, or to use the plus (+) or minus (-) PICTURE characters.

E. Example Output

Two examples of what the output of an **ICOBOL** compiler invocation looks like are given below:

Example 1

```
>icobol logon
icobol Revision 3.40 (Windows)
Copyright (C) 1987-2004, Envyr Corporation.All Rights Reserved.
Compiling C:\test200\logon.sr
1 file/argument was processed
No files/arguments had errors
icobol is finished.
>
```

Example 2

```
>icobol -s logon
icobol Revision 3.40
Copyright (C) 1987-2004,Envyr Corporation.All Rights Reserved.
Compiling C:\test200\logon.sr
Start: Aug-15-2000 07:34:34.62 Stop: Aug-15-2000 07:34:35.61
2719 lines compiled in 0.99 seconds (164787 lines per minute)
No errors, 1 warning, 16 info messages

1 file/argument was processed
No file/arguments had errors
icobol is finished.
>
```

F. Cross Reference Output

The cross reference output at attached at the end of the listing file and will have the following format:

```
Start of Cross Reference

IDENTIFICATION DIVISION Symbols:
...

ENVIRONMENT DIVISION (CONFIGURATION SECTION) Symbols:
...

ENVIRONMENT DIVISION (INPUT-OUTPUT SECTION) Symbols:
...

DATA DIVISION (FILE SECTION) Symbols:
...

DATA DIVISION (WORKING-STORAGE SECTION) Symbols:
...

DATA DIVISION (LINKAGE SECTION) Symbols:
...

DATA DIVISION (SCREEN SECTION) Symbols:
...

PROCEDURE DIVISION Symbols:
...

End of Cross Reference
```

Each symbol is displayed in the following format:

SYMBOLNAME symbol-type, symbol-type-info,
level number (when needed), address (in data) or pc (in code), size (bytes), and occurs (for tables), segment count, also count, and occurs for key-names.

[*sourcename*] xxx <*howused*>

Where

sourcename
Is the simple filename to which the following line numbers belong and is only given when there are COPY files.

symbol-type and
symbol-type-info
Are defined below in Table 38.

xxx
Is the line number in the source where this entry is used.

<*howused*>

How the symbol is used at this linenumber can include any of the following:

(dead)	Usage occurs in dead code
(implied)	Usage is implied by an operation
Begin	Procedure referenced as beginning of PERFORM range
CLOSE	File referenced in a CLOSE statement
Definition	Item is defined.
DELETE FILE	File referenced in a DELETE FILE statement
End	Procedure referenced as end of PERFORM range
GO TO	Procedure referenced in GO TO
Modified	Data item is modified and then the statement type (ACCEPT, ...)
OPEN	File referenced in an OPEN statement
READ	File referenced in a READ statement
Referenced	Data item is referenced and then the statement type (ACCEPT, ...)
REWRITE	File referenced in a REWRITE statement
Used	Item is used in another item's definition
WRITE	File referenced in a WRITE statement

The Used is given for an item used for example with a REDEFINES, a SCREEN TO/FROM/USING clause, a SCREEN LINE/COL, FILE STATUS clause in the FD, file-name in SELECT statement, KEY clause and ALTERNATE KEY clause.

The definition is the first line shown and then the remaining numbers are in the same order the source was processed.

symbol-type	symbol-type-info
file-name	SEQUENTIAL, RELATIVE, INDEXED
data-name	group, alphabetic, alphanumeric, numeric-edited, numeric integer [USAGE COMP [3|5]], numeric [USAGE COMP [3|5]], POINTER
screen-name	group, alphabetic, alphanumeric, numeric-edited
section-name	[declaratives] pc
paragraph-name	[declaratives] pc
program-name	
index-name	
key-name	
condition-name	

TABLE 40. Cross Reference Symbol Types

Filenames; Primary-, Alternate-, and Relative- Keys; and File Status symbols that are used in a SELECT show the line number of the FD instead of the SELECT.

G. ICODBC Support

By using the Make ICODBC Definition Files switch (-M) the compiler can be used to create a Database Definition File (.XDB) and one or more Table Definition Files (.XDT) for use with the ICODBC Driver. The compiler creates a single Database Definition File (.XDB) and a Table Definition File (.XDT) for each Indexed File that is contained in the corresponding source program.

As each indexed file described in the program is processed, a column definition will be created for every data item described in the record definition or definitions. This will be a complete picture of the indexed file, and via the ICODBC Driver, it will enable an ODBC-enabled application (Microsoft Access, Crystal Reports, etc.) to select, read, and format any data in the indexed file. However, this may be more capability than is required and the .XDT and .XDB files can be edited with a text editor to remove and/or modify any columns that are not needed.

Several options for ICODBC definition files are supported via the -X switch to assist in narrowing the number of columns created for each table and formatting options that allow table names and column names to be created from identifiers in the COBOL program in a variety of forms.

Using the Make ICODBC Definition Files switch does not necessarily create what you would like as a finished set of Database Definition Files or Table Definition Files. However, it does remove the drudgery of mapping data types and of determining the positions and lengths of each data item required as a column in the table. The output from this option should be considered as a good starting point from which to create your definition files with a text editor.

For each source file that includes indexed file definitions, a single Database Definition File (.XDB) and one or more Table Definition File (.XDT) will be created. Each of these files is created in the directory specified by the -M switch. If the -n switch is given in the -X switch, then an error will be generated if it needs to create a file and the file already exists. If the -n switch is NOT given in the -X switch, any existing file(s) will be overwritten.

The name of the Database Definition File (.xdb) is derived from the simple name of the source file and adding the .XDB extension. The name of each Table Definition File (.xdt) is derived from information in the original COBOL program in the following manner:

(1) If the indexed file's external filename was specified as a literal, then a .XDT extension is appended to the simple name of the file specified by the literal.
(2) If the indexed file's external filename was specified as an identifier and that identifier has a VALUE clause, then a .XDT extension is appended to the simple name of the file specified by the contents of the VALUE clause.
(3) Otherwise, the internal (COBOL) filename is used with every hyphen (-) being converted to a dollar sign ($).

Table and column names are created from identifier names. There are two steps involved in the name creation. The first step is to apply prefix replacement. The second step is to apply the formatting option.

When ICODBC Prefix replacement (-P switch) is specified, each prefix specified is compared against each data item in the record. If the prefix matches, it is replaced with the new prefix specified or removed if no new prefix was specified. If it does not match, the next prefix specified on the command line is compared to the data item and so on. Only the first matching prefix is applied. For example, if the command line specified the switch -P AR:AP and the data item AR-TOTAL was a field in the data record being processed, AR-TOTAL would be converted to AP-TOTAL. If the command line option was -P AR, the AR-TOTAL would be converted to TOTAL. Any replacement is done prior to application of formatting options.

If an ICODBC Formatting option (-F switch) is specified, each data item from the record is formatted according to fixed rules as described above in the syntax. Consider the following examples starting with the data item AR-TOTAL-ON-ORDER:

-F value	Column Name Generated	With -P AR:ACCOUNT also
1	AR-TOTAL-ON-ORDER	ACCOUNT-TOTAL-ON-ORDER
2	AR_TOTAL_ON_ORDER	ACCOUNT_TOTAL_ON_ORDER
3	ArTotalOnOrder	AccountTotalOnOrder
4	Ar_Total_On_Order	Account_Total_On_Order

The following rules apply with regard to the inclusion or exclusion of data items when columns are being generated:

(1) FILLER items are included only if requested with the -I f switch. When included, their column name is generated as if the FILLER item had been named ICMAKEDB-FILLER-*n* where *n* is an integer increasing in value by one for each FILLER encountered.
(2) The primary key for the file is always included. This includes all segments for a suffixed key. No option may override this rule.
(3) Secondary record definitions, group items, data items with a REDEFINES clause, data items with a RENAMES clause, and data items with level numbers outside of a specific range may be excluded by specifying one of the -N switch options or the -L switch. These items are included unless explicitly excluded with a switch.

(4) No data item which has an OCCURS clause or which is subordinate to an OCCURS clause will be included. No option may override this rule.

(5) No item whose PICTURE includes the P character will be included. No option may override this rule.

For each column, a section in the .XDT file is created containing the data type, position in the record, and length of the data. For numeric types it will also generate the precision and scale of the data item. For any column which is an alternate record key, segment of an alternate record key or an ALSO key, a Suppress directive will be generated using the same character defined in the SUPPRESS WHEN clause for the alternate key.

ICOBOL data types are mapped to those used by the ICODBC Driver according to the following table.

Data Description	Type	Length	Precision	Scale
PIC A(*n*)	ALPHABETIC	*n*	n/a	n/a
PIC X(*n*)	ALPHANUMERIC	*n*	n/a	n/a
group item	ALPHANUMERIC	varies	n/a	n/a
alphanumeric edited items	ALPHANUMERIC	varies	n/a	n/a
numeric edited items	ALPHANUMERIC	varies	n/a	n/a
PIC X(*n*) or group used in a key or key-segment with sub-ordinated items of non-DISPLAY usage	BYTE	*n*	n/a	n/a
PIC 9(*l*)V9(*r*) USAGE DISPLAY	UNSIGNED DISPLAY	*l*+*r*	*l*+*r*	*r*
PIC S9(*l*)V9(*r*) USAGE DISPLAY	DISPLAY	*l*+*r*	*l*+*r*	*r*
PIC S9(*l*)V9(*r*) USAGE DISPLAY SIGN LEADING	LEADING OVERPUNCH	*l*+*r*	*l*+*r*	*r*
PIC S9(*l*)V9(*r*) USAGE DISPLAY SIGN LEADING SEPARATE	LEADING SEPARATE	*l*+*r*+1	*l*+*r*	*r*
PIC S9(*l*)V9(*r*) USAGE DISPLAY SIGN TRAILING	TRAILING OVERPUNCH	*l*+*r*	*l*+*r*	*r*
PIC S9(*l*)V9(*r*) USAGE DISPLAY SIGN TRAILING SEPARATE	TRAILING SEPARATE	*l*+*r*+1	*l*+*r*	*r*
PIC 9(*l*)V9(*r*) USAGE COMP	UNSIGNED COMP	varies	*l*+*r*	*r*
PIC 9(*l*)V9(*r*) USAGE COMP-3	UNSIGNED COMP-3	varies	*l*+*r*	*r*
PIC 9(*l*)V9(*r*) USAGE COMP-5	UNSIGNED COMP-5	varies	*l*+*r*	*r*
PIC 9(*l*)V9(*r*) USAGE BINARY	UNSIGNED COMP	varies	*l*+*r*	*r*
PIC 9(*l*)V9(*r*) USAGE PACKED	UNSIGNED COMP-3	varies	*l*+*r*	*r*
PIC S9(*l*)V9(*r*) USAGE COMP	COMP	varies	*l*+*r*	*r*
PIC S9(*l*)V9(*r*) USAGE COMP-3	COMP-3	varies	*l*+*r*	*r*
PIC S9(*l*)V9(*r*) USAGE COMP-5	COMP-5	varies	*l*+*r*	*r*
PIC S9(*l*)V9(*r*) USAGE BINARY	COMP	varies	*l*+*r*	*r*
PIC S9(*l*)V9(*r*) USAGE PACKED	COMP-3	varies	*l*+*r*	*r*
USAGE INDEX	COMP	4	9	0
USAGE POINTER	COMP-5	4	10	0

TABLE 41. **ICOBOL** Data Types to ODBC Data Types

More on ICODBC can be found in the ICODBC Driver Chapter starting on page 771.

XII. DEBUGGING

A. Introduction

The **ICOBOL** runtime system incorporates a high-level (source code) debugger. This high-level debugger is available whenever a symbol file (.sy) can be accessed for a particular COBOL program. Sources are available in this mode if the source file(s) can be accessed. The **ICOBOL** compiler generates symbol files through the use of the -Z switch. To start the runtime in debug mode use the -z or -Z switch.

ICOBOL programs are compiled into a pseudo-code (or p-code) that corresponds to the types of operations found in the COBOL language. The runtime system `executes' this pseudo-code when it runs the COBOL programs.

The debugger supports the following features:

- Breakpoints can be set on a particular type of instruction or operation. Several examples are: *Break Perform* will set a breakpoint on all PERFORM statements; *Break I-o* will set a breakpoint on all non-screen I/O; and *Break I-o Screen* will set a breakpoint on all screen I/O.
- Breakpoints can be set at the start or end of particular programs. Thus, if an application has hundreds of programs, but only a certain program has a problem, the debugger can be set to stop only when a particular program is executed allowing just that program to be debugged.
- Breakpoints are remembered for individual programs. If a program is canceled or is exited, the breakpoints are remembered. Thus, if the program is ever reactivated by a CALL or CALL PROGRAM, the breakpoints are reset before the program begins running.
- The current status of active and inactive CALLs and PERFORMs can be displayed.
- The number of open files, their names, file type, and open mode can be displayed.
- The debugger keeps two screen images, the debug screen and the COBOL screen. The debugger automatically switches to the COBOL screen when the COBOL program is run and back to the debug screen when re-entering the debugger. While in the debugger you can switch to the COBOL screen and back again to view the output from the COBOL program. You can also select a portion of the COBOL screen to always have displayed in the debugger screen.

The **ICOBOL** compiler generates individual symbol files (.sy) for each compiled program for debugging by using the -Z switch (-Z). The symbol file includes all line offsets that have code, all information about data items, and the names of source files used in the compilation. The debugger requires the symbol file to support the following additional features:

- Breakpoints can be set at a line number, at the beginning or end of a procedure, and whether a particular data-item has been changed.
- Data-items can be viewed and set to new values using their names.
- Up to 8 breakpoints (with no more than 1KB of data) can be set to test if particular data items change their values. (Some other debuggers refer to this type of breakpoint as a "watchpoint").
- If sources are available, the actual COBOL programs can be viewed as needed to track the program. In addition, the *Find* command can be used to search for a particular <string> in the COBOL source.

Symbol files (.sy) used by the debugger must be in the correct byte order for the machine on which the debugger is running. (i.e., they must have been compiled on the same type of machine (big-endian versus little-endian.)

Sources are sought as specified in the .sy file, in the current directory, or as specified with the -z or -Z switch..

B. Invocation

To start the debugger enter ICRUN with the -Z or -z switch. The -Z *sydir* switch specifies the location of the symbol files(.sy) that were created by the **ICOBOL** compiler using the same switch. If -z is given, the current directory is used. For example, the following would start **ICOBOL** in the debugger and instruct the debugger to look for symbol files in the *symboldir* directory.

```
icrun -Z symboldir
```

If not enabled, the SCREEN OPTIMIZER will automatically be placed in partial mode (ICSCROPT=partial).

C. **Usage**

A snapshot of the default debug screen when loading LOGON whose source files(s) and symbol file are present is shown in SCREEN 1.

```
    1084   END DECLARATIVES
    1085
    1086   MAIN-LOGIC SECTION.
    1087   ONLY-AT-START.
    1088   * PD/DD Revision 7 feature
**> 1089+   ACCEPT ENVIR-STRUCTURE FROM ENVIRONMENT.
    1090+      INSPECT OEM-REV-STRING REPLACING ALL LOW-VALUES BY SPACES.
    1091+      ACCEPT LINE-NUMBER FROM LINE.
    1092+      ACCEPT USERNAME FROM USER NAME.
icrun Revision 3.40 (os)
Stopped at line 1089 in "logon": The initial program is loaded.



> _
```

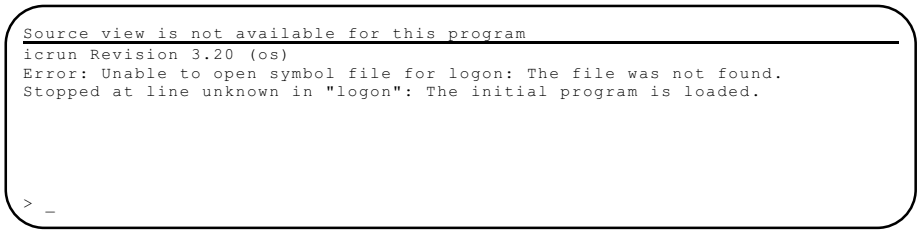
SCREEN 1. Default Debugging SCREEN

A snapshot of the same debug screen with the display view enabled is shown in SCREEN 2.

```
    1084   END DECLARATIVES
    1085
    1086   MAIN-LOGIC SECTION.
    1087   ONLY-AT-START.
    1088   * PD/DD Revision 7 feature
**> 1089+   ACCEPT ENVIR-STRUCTURE FROM ENVIRONMENT.




icrun Revision 3.40 (os)
Stopped at line 1089 in "logon": The initial program is loaded.



> _
```

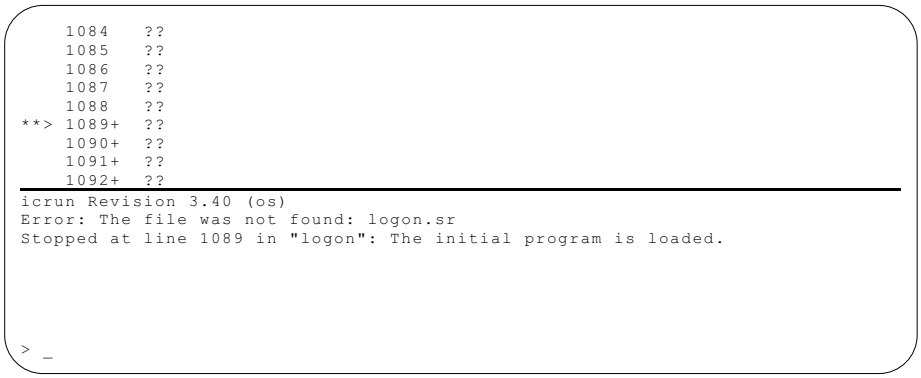
SCREEN 2. Debugging SCREEN (all views enabled)

A snapshot of the same initial debug screen with no symbol file available for LOGON is shown in SCREEN 3.



SCREEN 3. Debugging SCREEN (no symbol file)

A snapshot of the default debug screen when loading LOGON where the symbol file is available but the initial source file is not available is shown in SCREEN 4.



SCREEN 4. Debugging SCREEN (symbols but no source)

The `??' symbols in the source window indicate for this line number the source line is not available.

The debug screen is split into up to four(4) windows. The number of lines within each window is determined dynamically based on the number of lines supported by the console and the number of enabled windows. The command window is always present with two lines.

Source window (the top)

If the symbol file is not found, the source window will shrink to one line, and no debugger commands that use source will be allowed.

If the symbol file is found and the source for the current program can be opened, the actual program source with line numbers is shown. If the source can not be found, a message indicating that will be displayed in the output window and in the source window a `??' will be used to indicated that no source is available for this line. In scrolling through the source a COPY file may be encountered whose source is available and then the actual source lines will be displayed.

If a source line has code associated with it a `+' will be displayed after the line number. Location breakpoints can only be set on lines marked with a `+'. Location breakpoints are indicated by a `B' after the line number. A `==>' to the left of the line number indicates the current execution line. A `-->' to the left of the line number is the current location of the cursor in the source file. A `**>' to the left of the line number indicates that the current execution line and the current line in the source file are the same.

The source window is enabled by default.

The source window is scrollable with the *View* and *Zoom* commands. The *View* and *Zoom* commands can be used even if the window is not enabled.

Display window

The display window shows lines from the current COBOL program screen output.

The display window is not enabled by default, it must be enabled with the *View ON Display* command.

The display window is scrollable (within the content of one screen) with the *View* and *Zoom* commands. The *View* and *Zoom* commands can be used even if the window is not enabled.

Output window

All user commands and debugger responses are shown in the output window. The output window is scrollable by using the *View* and *Zoom* commands, allowing positioning anywhere in the current output from the start of this debug session to the last displayed message.

The output window is the main work area for the debugger and is enabled by default.

Command window (bottom two lines)

The bottom two lines of the debug screen form the command window. The command window is always enabled. The first line holds the debugger prompt `> ', while the second line provides a message text string. The debugger keeps a circular `history' of the last twenty (20) commands that have been entered. These are accessible by using the up- and down-arrow keys. This makes it fairly easy to repeat a sequence of commands. One-character commands are not saved.

Whenever the debugger is entered, an appropriate reason will be displayed in the output window just above the command window. The reason will include the current line number and program name.

Several possible reasons are given below:

The run unit is finished:

The runtime is ready to exit back to the operating system with the incoming reason code. If the reason is cleared, the system will restart with a *Run*. If the reason is not cleared, it will be returned as the error code and the system terminated with a *Run*.

Break:

The program has stopped for a breakpoint. There is no incoming reason code. Exception Status may be set.

Interrupt:

The program has stopped for a program interrupt. The interrupt code is the reason code, it is what will happen with a *Run* or *Step* unless cleared. The Exception Status is not set to this code. An *Error Reset* will clear the reason code.

The initial program is loaded:
Break After "name":
Break Before "name":

The program has stopped for the beginning of a statement for the initial startup, the before or after "name" breakpoint, or single-stepping. There is no incoming reason code.

Break Global Perform (Use):
The program has stopped at the start of a perform operation. There is no incoming reason code.

Break Global eXit:
The program has stopped at the start of an exit perform operation. There is no incoming reason code.

Break Call "name" (program or |):
The program has stopped at the start of a call [|] operation. There is no incoming reason code.

Break Call Program "name" (program or #):
The program has stopped at the start of call program [#] operation. There is no incoming reason code.

Break eXit Program:
The program has stopped at the start of a exit perform operation. There is no incoming reason code.

Break I-o:
Break I-o Screen:
The program has stopped at the start of an I/O or Screen I/O operation. There is no incoming reason code.

Break Stop:
The program has stopped at some type of stop-run situation. The incoming reason code should indicate the reason.

Break Error Call:
The program has stopped for an error from Call to a COBOL program. Exception Status is the same as incoming reason code. A failure will always still be in the original program.

Break Error I-o:
Break Error I-o Screen:
The program has stopped for an I/O or Screen I/O error. Exception Status is the same as incoming reason code. The File Status has been set up according to the error (unless ACCEPT or DISPLAY).

Break Error Stop:
The program has stopped for a fatal error. Exception Status is not set to the fatal error although the reason code is.

On Windows, a Ctrl-C will show an Exception Status 193 "Program terminated by console interrupt".

On UNIX, the UNIX Quit key will show an Exception Status 255 "The process was terminated" here. The UNIX Intr will show an Exception Status 193 "Program terminated by console interrupt".

The incoming reason code is important when using the Break Error features, since it is the error that will be used when resuming execution with the *Run* command, unless cleared with *Error Reset*.

For the Break Call type breakpoints, the reason shows the call argument as a quoted string. If the string is too long to fit, it is followed by an ellipsis (...).

When first started, the debugger is entered after loading the initial program. The message "The initial program `name' is loaded" is displayed in the Output window. For other cases of when the debugger is entered please see the section on the *Break* command.

D. Commands

D.1. Overview

At the debugger prompt the following commands are available:

Audit	Echo commands to a command file
Break	Manage breakpoints
Command	Execute a host operating system command
Dump	Display data values
Error Reset	Reset the incoming reason code
eXecute	Execute a command file
Find	Search for a specific string in a source file
Go	Go to a different pc in the program
Help or *?*	Show help information
Info	Provide information on various items
List	List a text file
Move	Allow data values to be changed
Quit	Quit the debugger and exit *the runtime*
RERUN	Run the program again from the beginning
Run	Run the program
Step	Single-step source into CALL/PERFORM
Type	Formatted breakdown of the contents of data items
View	View a particular window or enabled/disable a particular window
Zoom	Zoom a particular window to full size

All commands and keywords can be abbreviated as indicated by the uppercase letters, typically the first letter of the command or keyword. If a keyword is all uppercase, it cannot be abbreviated. Words that are completely in lowercase and *italic* represent generic items, like a number or a string of text.

The command syntax below uses the following conventions:

{} for required choices and
[ ] for optional choices.

D.2. AUDIT

The *Audit* command controls the echoing of commands to an audit file. It provides a simple mechanism for creating command files that can then be run with the *eXecute* command. You can look at command files with the *List* command.

The syntax is:

```
> Audit { "filename"
        { Extend "filename"
        { Close
```

Where

"*filename*" specifies a text file.

Below is a description for each valid syntax.

Audit "*filename*"	Recreates an existing file or creates a new one to echo debugger commands.
Audit Extend "*filename*"	Appends to an existing file or creates a new one to echo debugger commands.
Audit Close	Closes the audit file.

There can be at most one audit file active at a time. All debugger commands that pass the syntax checks and are executed will be audited to the file except the *Audit Close* command and the *eXecute* command. While the *eXecute* command itself is not audited, the commands from the eXecute file are audited as they are processed.

D.3. BREAK

The *Break* command encompasses all aspects of breakpoint management including setting, deleting, listing, and temporarily disabling and reenabling breakpoints.

Unless specific breakpoints are set the debugger is entered in one of four ways:

1) The debugger is entered after loading the initial program. The message "The initial program `name' is loaded" will be displayed in the Output window.
2) The debugger is entered after the last program terminates. The message `The run unit is finished` will be displayed in the Output window. Entering *Run* or *Quit* will cause the runtime to terminate.
3) If program interrupts are enabled, pressing Ctrl-C (on Windows) or the Intr or Quit keys (on UNIX) while running the COBOL program will cause the debugger to be entered. The message "Interrupt at . . . in `program-name'" will be displayed in the Output window along with the appropriate Exception Status.
4) The debugger is entered after an ICEXEC command Abort has been done on the pid or a `*kill -15*' or `*kill -SIGUSR1*' on the pid. The message "Interrupt at . . . in `program-name'" will be displayed in the Output window along with the appropriate Exception Status.

If the program was in an ACCEPT operation when one of the interrupts in #3 or #4 occurs, the debugger is entered with the reason "Console Interrupt in screen read". Execution will resume in the screen read. Only a *Run* can be done at this point to continue the program.

When breakpoints are enabled, the debugger is usually entered because of a breakpoint. The reason information will show which breakpoint caused entry into the debugger.

The following descriptions are grouped into several categories. Note that location and test breakpoints can only be set for the current active program

The syntax is:

Setting location breakpoints:

```
> Break { AT procedure
          END procedure
          line }                    [ COUNT=counter [ RESET ] ]
```

Setting global breakpoints:

```
> Break { { After } { "name" }
          { Before } { Any }
          Call [ | ]
          Call Program [ # ]
          I-o [ Screen
                Indexed
                Relative
                seQuential
                iNfos
                sqL ]
          Perform
          Stop
          eXit [ Program ] }                [ COUNT=counter [ RESET ] ]
```

Setting data change breakpoint (only one allowed):

```
> Break Test identifier   [ COUNT=counter [ RESET ] ]
```

Setting error breakpoints:

```
> Break Error { exception
                Any
                Call [ | ]
                Call Program [ # ]
                I-o [ Screen
                      Indexed
                      Relative
                      seQuential
                      iNfos
                      sqL ] }                [ COUNT=counter [ RESET ] ]
```

Deleting, Enabling/Disabling, Listing Breakpoints:

```
> Break { Delete } { breaknumber... }
        { ON     } { "name"       }
        { OFF    } { ALL          }

> Break [ List [ breaknumber... ]
               [ "name"       ]
               [ ALL          ]
```

Where

breaknumber	is a specific breakpoint number. Each breakpoint has a unique breakpoint number assigned to it.
counter	is an integer and represents the number of times this breakpoint must be encountered before executing the breakpoint.
exception	is valid exception value to watch for.
identifier	is a valid data-item in this program. The default format is the storage type of the item. (Requires a symbol file.)
line	must be a valid line number in the current program. (Requires a symbol file.)
"*name*"	must be a simple program name enclosed in double-quotes.
procedure	must be a valid section or paragraph name in this COBOL program. (Requires a symbol file.)

Below is a description for each type of breakpoint that can be set with the command on the left and the action on the right.

Location Breakpoints

Break AT *procedure*	Set a breakpoint at the first executable line in the given paragraph or section
Break END *procedure*	Set a breakpoint at the end of the given paragraph or section if it ends a PERFORM or at the first line of the next paragraph for a paragraph that is not an ending paragraph.
Break *line*	Set a breakpoint at the specified linenumber

Location breakpoints are indicated in the source window by a `B' following the line number.

Global Breakpoints

Break After "*name*"	Break after subprogram "name" has finished and returned to the calling program.
Break After Any	Break after every subprogram has finished and returned to its calling program. The Any form overrides all "name" form breakpoints, but the "name" ones are remembered, so if Any is cleared, the "name" breakpoints will be reset.
Break Before "*name*"	Break before program "name" begins execution each time it is started by a CALL or CALL PROGRAM.
Break Before Any	Break before every program or subprogram begins execution. The Any form overrides all "name" form breakpoints, but the "name" ones are remembered, so if Any is cleared, the "name" breakpoints will be reset.
Break Call	Break at every CALL of a subprogram or builtin function.
Break Call |	Break at every CALL of an operating system call.
Break Call Program	Break at every CALL PROGRAM of a COBOL program.
Break Call Program #	Break at every CALL PROGRAM # system call.
Break eXit	Break at every exit from a PERFORM that will exit, i.e., for PERFORM n TIMES, it breaks at the exit from the last of the n times.
Break eXit Program	Break at every EXIT PROGRAM in a subprogram.
Break Perform	Break at every PERFORM statement, including the implicit perform of a USE PROCEDURE, although excluding in-line PERFORMs other than an in-line PERFORM n TIMES.
Break I-o	Break at every I/O operation, excluding ACCEPT and DISPLAY.
Break I-o Screen	Break at every screen I/O operation (ACCEPT and DISPLAY).
Break I-o Indexed	Break at every indexed I/O operation (CLOSE, DELETE, OPEN, READ, REWRITE, START, UNLOCK, and WRITE).
Break I-o Relative	Break at every relative I/O operation (CLOSE, DELETE, OPEN, READ, REWRITE, START, UNLOCK, and WRITE).
Break I-o seQuential	Break at every sequential I/O operation (CLOSE, OPEN, READ, REWRITE, START, and WRITE).
Break I-o iNfos	Break at every INFOS I/O operation (CLOSE, DELETE, EXPUNGE, OPEN, READ, RETRIEVE, REWRITE, START, SUB-INDEX operations, UNLOCK, and WRITE).
Break I-o sqL	Break at every ***ISQL*** statement (COMMIT, CONNECT, DEALLOCATE, DISCONNECT, EXECUTE, EXECUTE IMMEDIATE, FETCH, PREPARE, ROLLBACK, and SET CONNECTION).

Break Stop	Break at every STOP RUN and at every STOP "lit" in which the user has chosen to stop (i.e., hit ESC).

Data Change Breakpoint

Break Test *identifier*	Break if the storage associated with identifier has changed.

Error Breakpoints

Break Error *exception*	Break after every operation that returns the Exception Status specified by `*exception*'. Only one *Break Error exception* is allowed; exception cannot be 0.
Break Error Any	Break after every operation that returns a non-zero exception code.
Break Error Call	Break after every CALL of a subprogram that returns with an error.
Break Error Call |	Break after every CALL of an operating system call that returns an error.
Break Error Call Program	Break after every CALL PROGRAM of a COBOL program that returns an error.
Break Error Call Program #	Break after every CALL PROGRAM # system call that returns an error.
Break Error I-o	Break after every I/O operation that returns an error, excluding ACCEPT and DISPLAY), including DELETE FILE, EXPUNGE, START, UNDELETE, UNLOCK, and STOP "lit".
Break Error I-o Screen	Break after every screen I/O operation that returns an error (ACCEPT and DISPLAY).
Break Error I-o sqL	Break after any ***ISQL*** statement that returns an error (SQLSTATE not = 0).

Miscellaneous

Break [List [ALL]]	List ALL breakpoints.
Break Delete *breaknumber*	Clears the given breakpoint number.
Break Delete "*name*"	Clears all Location type breakpoints in the program specified by "name".
Break Delete ALL	Clears ALL breakpoints in the debugger!!
Break OFF ALL	Temporarily disable ALL breakpoints.
Break ON ALL	Reenable ALL temporarily disabled breakpoints.
Break OFF *breaknumber*	Temporarily disable the given breakpoint number.

The keywords AT, END, OFF, ON, and ALL cannot be abbreviated.

All breakpoints are persistent. The debugger remembers breakpoints even when a program has been CANCELed from the run-unit or replaced by a CALL PROGRAM. If the program ever becomes part of the run-unit again through a CALL or CALL PROGRAM, the debugger resets the breakpoints. In order to delete breakpoints, use the *Break Delete* command.

Location and Test breakpoints can only be set while in the applicable program.

Up to 8 Break Test breakpoints can be set.

D.4. COMMAND

The *Command* command is used to process commands through to the host operating system command line processor, shell, or CLI.

The syntax is:

```
> Command [ "string" ]
```

Where

string is a particular string of commands to pass to the host command processor to be executed, after which it returns automatically. If no *string* is given, the debugger pushes to the command processor in interactive mode.

When the host operating system command line processor returns to the debugger, the prompt "Press any key to continue" is given. If an error is returned from the command processor an Info message is given showing the actual error returned.

D.5. DUMP

The *Dump* command displays storage for particular data items. It can show the data in one of several formats.

The syntax is:

> Dump [*identifier* [Decimal | Hex | Octal | Alphanumeric]]

Where

identifier is a valid data-item in this program. The default format is the storage type of the item.

Below is a description for several valid syntaxes:

Dump *identifier*	Displays the content of the data-item in a format appropriate to the type of the data-item.
Dump *identifier* Hex	Displays the data at the address and length as specified for the given identifier in hex.
Dump	Display the next data area in the same format as the previous area if the identifier was a table element. The address immediately follows the item from the previous display and the length used is the same.

The Decimal, Hex, or Octal format specifiers display data in a dump format that has the address in the left margin, followed by a numeric decoding of ten-bytes of data in the selected numeric format, followed by a decoding of the ten-bytes as ASCII characters. In the ASCII dump, unprintable characters are shown as periods.

Unsigned numeric items are shown with no sign. Signed numeric items are shown with either a `+' or `-'. In an identifier numeric dump, a decimal point is displayed when appropriate based on the PICTURE and actual value of the item.

The Alphanumeric type will display data as a quoted string, using the <nnn> octal notation for unprintable characters.

If the data item is a table element, you must supply valid subscripts the first time entered. After that a Dump with no arguments will increment the subscript(s) as needed to dump the next value..

D.6. ERROR RESET

The *Error Reset* command is used to clear the incoming reason code, which will be used by the program when it resumes.

The syntax is:

> Error Reset

See the *Step* and *Run* commands for the implications of using Error Reset.

D.7. EXECUTE

The *eXecute* command is used to run a series of debugger commands stored in a text file.

The syntax is:

```
> eXecute "filename"
```

Where

"filename" is a text file that holds valid debugger commands to be executed. The commands from *"filename"* are echoed to the screen as they are executed. If the *Audit* command is active, the *eXecute* command itself is not echoed, but the commands are.

D.8. FIND

The *Find* command is used to find a string of text in the source of the current program. *Find* requires that the source file be available for the current program.

The syntax is:

```
> Find [ First    ]                    [ "string" ]
       [ Last     ]
       [ Next     ]
       [ Previous ]
```

Where

"*string*" is a string of text to search for in the current program source.

The First specifier indicates that the search is to start at the beginning of the program and proceed in the forward direction.

The Last specifier indicates that the search is to start at the end of the program and proceed backwards.

The Next and Previous specifiers start at the location of the last find and proceeds forward or backward respectively.

The string argument is required for the first *Find* command. If not specified on subsequent *Find* commands, the most recently specified value is used.

Find without any arguments repeats the most recent *Find* command.

D.9. GO

The *Go* command moves the COBOL PC to a new location. It does not start execution.

The syntax is:

```
> Go { TO procedure }
   { line      }
   { eXit      }
```

Where

procedure must be a valid section or paragraph name in the current COBOL program. (Requires a symbol file.)

line must be a valid line number in the current COBOL program. (Requires a symbol file.)

Below is a description of valid combinations:

Go TO *procedure*	The current PC is moved to the first executable line in the given procedure.
Go *line*	The current PC is moved to the start of the given linenumber.

Go eXit	The current PC is moved to the return address in the current (topmost) perform, and the perform stack is popped a level.

This command does not cause the program to run, but it does change the location where execution will occur the next time that execution resumes with a *Run* or *Step*.

The view in the source window will be adjusted appropriately.

You should be careful about moving the location from within a PERFORM or into a PERFORM as it could confuse the runtime with its PERFORM stack.

D.10. HELP

The *Help* command provides general help or help for a specific debugger command.

The syntax is:

```
> Help [ command ]
> ? [ command ]
> command ?
```

Where

command	must be a valid debugger command

If no command is given, a general help is given showing all possible debugger commands.

If command is given, detailed help will be displayed for that particular debugger command only.

Help is also available from the command-line by pressing F1. If the command name (or its abbreviation) has already been typed in, help will be provided for that command.

All help information is shown is a separate zoomed help window. While in the zoomed window, the cursor up, cursor down, F2 (page up), and F3 (page down) can be used to position up and down in the window to view the contents.

Pressing ESC will cause the command to exit.

D.11. INFO

The *Info* command is used to display specific information about the state of the program or various program elements.

The syntax is:

```
> Info [ ON qualified-name
       | Call [ count ]
       | Detail
       | Error [ exception ]
       | Open
       | Perform [ count ]
       | eXit [ count ]
       | sqL ]
```

Where

qualified-name	is a valid item in the current COBOL program. (Requires a symbol file.)
count	is an integer value indicating to only show the topmost *count* items of the selected values.
exception	is a valid Exception Status.

Each particular *Info* command provides specific information about the indicated item.

Info On qualified-name provides a description of the item including its class and category, size, and address.

Info Call displays the active call list. If count is given only the topmost count programs will be displayed.

Info Detail provides detailed information about the internal state of the runtime system that provides the following:

A. (line 1) the Program name and dialect compiled with,
B. (line 2) the ESCAPE KEY, the Exception Status, the File Status, the INFOS Status,
C. (Optional) if the exception status has an embedded os error message and extra line showing
"Exception register [INFO on:] OS-err:
Along with any severity of the error will be given.
D. (line 3/4) the number of Performs active (for this program); Open files (for this program); number of Active programs; the number of Inactive programs (for this process), and the program counter (PC);

A sample for *Info Detail* is shown below:

```
Name: ussteel                                    Dialect: icobol
ESC: 00 EXC: 00000 FileStatus: '00' InfosStatus: '000000000000'
Perform: 01  Open: 01  Active: 01  Inactive: 00   PC: 13523
```

Info Error displays the text associated with the current Exception Status or if a particular exception code it given then the text for that exception is displayed.

Info Open displays a list of the files currently open by this program with their name, type, and open mode.

Info Perform displays the Perform stack. If count is given only the topmost count performs will be displayed.

Info eXit displays the inactive call list. If count is given only the topmost count programs will be displayed.

Info sqL displays ***ISQL*** information about active SQL connections. A sample would look like:

```
SQLState: 02000 SQLText: [Envyr Corporation][icrun] No data was affected by..
Connection: TestDSN
    String: TestDSN
    Statement: 1
       Cols: 3 PREPAREd: Y EXECUTEd: Y
```

D.12. LIST

The *List* command is used to list the contents of any text file.

The syntax is:

> List "*filename*" [First | Last | *line*]

Where

"*filename*" is a valid text file to be opened and listed.
line is a valid line number in the given file.

The First, Last, and *line* options provide a default starting position in the file.

The indicated text file is listed in a zoomed list window. The First, Last, and *line* options provide a default starting position in the file. If no default starting position is given the First line is used.

While in a zoomed window the cursor up, cursor down, F2 (page up), and F3 (page down) can be used to position up and down in the window to view the contents.

Pressing ESC will cause the command to exit.

D.13. MOVE

The *Move* command allows storage to be changed to a new value.

The syntax is:

> Move { *identifier* / *literal* } TO *identifier*

Where

identifier	must be a valid identifier in this COBOL program. (Requires the symbol file.)
literal	must be a valid COBOL literal.

The *Move* is just like the MOVE statement in COBOL.

String literals may have imbedded octal, decimal, or hex values using the construct <*nnn*>, <o*nnn*>, <d*nnn*>, and <x*nn*> to specify a byte value represented by the *nnn* or *nn* numbers. In the case of <*nnn*> and <o*nnn*>, *nnn* represents an octal value, in <d*nnn*>, *nnn* represents a decimal value, and in <x*nn*>, *nn* represents a hex value. Upper and lower case `o', `d', and `x' can be used to specify octal, decimal, or hex. In hex mode, upper and lower case `a' - `f' can be used. The value for any byte must be in the range 0 - 255 (decimal). For octal and decimal, no more than three digits can be specified and for hex no more than two digits can be specified. <1> is treated as <001>. The construct << can be used to enter a single <. Only one byte can be specified per <> pair.

D.14. QUIT

The *Quit* command performs an implicit STOP RUN (closing all files) and exits the debugger.

The syntax is:

```
> Quit
```

After a STOP RUN, only *Quit* and *Rerun* are allowed.

D.15. RERUN

The *RERUN* command is used to restart the debug session. An implicit STOP RUN (closing all files) is done and the initial program is reloaded. All breakpoint information is retained, and all the programs that had become part of the memory image are retained in a canceled state.

The syntax is:

```
> RERUN
```

After a STOP RUN, only *Quit* and *RERUN* are allowed.

D.16. RUN

The *Run* command resumes program execution and optionally set a temporary location breakpoint.

The syntax is:

```
> Run [ TO procedure ]
      [      line      ]
```

Where

procedure must be a valid section or paragraph in the current COBOL program. (Requires a symbol file.)
line must be a valid line number in the current COBOL program. (Requires a symbol file.)

A more detailed description for each combination is given below:

Run	Resumes program execution.
Run TO *procedure*	Sets a temporary breakpoint at the first executable line in the given paragraph or section and then Resumes program execution
Run *line*	Sets a temporary breakpoint at the given linenumber and then Resumes program execution.

The *Run* command is used to resume program execution. The program resumes execution with the incoming reason code, as shown in the Output window, unless the *Error Reset* command was used to clear the reason code. In cases where the debugger was entered on a *Break Error* type breakpoint, it is usually better to run with the incoming reason code so that the program continues as it would normally. In some cases, resetting the reason code can lead to erroneous program behavior, e.g., if a READ statement failed, the record area may be corrupt. However, if the program was entered by an external interrupt, e.g., the Intr key, it is often advantageous to reset the error so that the program will continue to run rather than terminating, as is the normal result.

If the debugger had been entered by pressing the Interrupt key, the *Error Reset* command must be used before resuming, or else the program will be terminated with a console interrupt.

If you are in an ACCEPT operation, the ACCEPT will be resumed when *Run* is entered. (The reason is "Console Interrupt within a screen read".)

If the run-unit has terminated, issuing an *Error Reset* followed by *Run* will start the original program over again. This can also be done with a *RERUN*.

If after setting a temporary breakpoint, any other breakpoints are encountered during the course of running other than the temporary breakpoint, the debugger will stop at that breakpoint and the temporary breakpoint will be cleared.

The *Run* command automatically switches to the COBOL screen before starting execution.

D.17. STEP

The *Step* command single steps to the next statement. Step follows CALL and out-of-line PERFORM's to their targets.

The syntax is:

```
> Step [ count ]
```

Where

count is the number of steps to be performed.

If any other breakpoints are encountered during the course of stepping, the debugger will stop at that breakpoint and the *Step* operation will be canceled.

Instead of using the count option it is often better to use the "RUN xxx" form of the *Run* command where xxx is the line-number.

Step	Single steps by one statement. If the current instruction is a CALL or PERFORM the debugger will stop at the first statement or instruction in the target of the CALL or PERFORM.
Step 10	Single steps by 10 statements. If the current statement is a CALL or PERFORM the debugger will stop at the 10th statement inside the CALL or PERFORM.

Step, just like *Run*, continues execution with the incoming reason code unless it has been cleared with the *Error Reset* command. See the *Run* command for additional information on how this affects program behavior.

D.18. TYPE

The *Type* command is used to show the contents of data items. It breaks down group items into their composite data items. *Type* requires that a symbol file be available.

The syntax is:

> Type [*identifier*]

Where

identifier is a valid data-item in the program.

This command displays a formatted breakdown of the contents of *identifier*. The output contains the level number, identifier name, and its current value. For a group item, it shows the contents of each elementary data item in a format appropriate for the data type.

When no arguments are supplied, the results depend on the *identifier* of the previously executed *Type* command. If the preceding *Type* command was a display of a table item, the next element of the table is displayed. The debugger will automatically increment the subscripts. If the preceding *Type* was of a simple data item, the command will return an error.

An example of the output from this command follows:

```
> type group-item
    01 GROUP-ITEM
      05 EMP-NAME
        10 EMP-NAME-FIRST = "Joe       "
        10 EMP-NAME-LAST = "Programmer     "
      05 EMP-AGE = 47
      05 EMP-PHONE = 8005551212
```

D.19. VIEW

The *View* command is used to control which windows are active on the screen and to position to a particular window to browse.

The syntax is:

> View [ON | OFF] [Source | Display | Output] ...

> View Reset

When the debugger initially starts the source, output, and command windows are enabled by default. The *View* command can be used to enable others or disable current windows.

The windows are always positioned in order from the top of the screen, with Source first, Display second, Output third, and Command fourth. The *View ON* and *View OFF* commands can have multiple arguments to change

multiple windows at the same time. When ON or OFF is omitted, only a single window is allowed. A *View* command with no arguments positions to the Output view by default.

When positioned to a window, the cursor up, cursor down, F2 (page up), and F3 (page down) can be used to position up and down in the data displayed in the window. ESC will return you to the command prompt.

The View Reset command resets the source window to be positioned around the current program execution location. It is useful for quickly returning these views to their default state after using the View or Zoom commands to look at some other areas of the program.

D.20. ZOOM

The *Zoom* command is used to zoom a particular window to full size.

The syntax is:

```
> Zoom [ Source  ]
     [ Display ]
     [ Output  ]
```

The *Zoom* command without any argument zooms the Output window. A zoomed window expands to the whole screen minus the 2-line command window.

While in a zoomed window the cursor up, cursor down, F2 (page up), and F3 (page down) can be used to position up and down in the window to view the contents.

Pressing ESC will cause the command to exit.

E. Performance Considerations

When running programs in the debugger, performance is degraded. The degradation is in the 10 to 25% range. Use of the *Break Test* breakpoint causes a dramatic performance degradation as this test must be performed at the start of every opcode to detect changes from a preceding opcode.

Setting ICSCROPT=full instead of ICSCROPT=partial will provide for better screen optimization as the screens will not actually be re-painted unless there is an actual change to the screen. This is especially helpful when using the *Step* command.

F. Quick Reference

? [*command*]
command ?

Audit { "*filename*" | Extend "*filename*" | Close }

Break { AT *procedure* | END *procedure* | *line* } [COUNT=*counter* [RESET]]

Break { { After | Before } { "*name*" | Any } | Call [|] | Call Program [#] | I-o [Screen | Indexed | Relative | seQuential | iNfos | sqL] | Perform | Stop | eXit [Program] } [COUNT=*counter* [RESET]]

Break Test *identifier* [COUNT=*counter* [RESET]]

Break Error { *exception* | Any | Call [|] | Call Program [#] | I-o [Screen | Indexed | Relative | seQuential | iNfos | sqL] | Stop } [COUNT=*counter* [RESET]]

Break [List [*breaknumber*... | "*name*" | ALL]]

Break { Delete | ON | OFF } { *breaknumber*... | "*name*" | ALL }

Command [*"string"*]

Dump [*identifier* [Decimal | Hex | Octal | Alphanumeric]]

Error Reset

eXecute *"filename"*

Find [First | Last | Next | Previous] ["*string*"]

```
Go { TO procedure }
   {    line     }
   {    eXit     }

Help [ command ]

     [ ON qualified-name ]
     [   Call [ count ]  ]
     [      Detail       ]
Info [ Error [ exception ] ]
     [      Open         ]
     [ Perform [ count ] ]
     [   eXit [ count ]  ]
     [      sqL          ]

List "filename" [ First ]
                [ Last  ]
                [ line  ]

Move { identifier } TO identifier
     { literal    }

Quit

RERUN

Run [ TO procedure ]
    [    line      ]

Step [ count ]

Type identifier

View [ ON  ] [ Source  ] ...
     [ OFF ] [ Display ]
             [ Output  ]

View Reset

Zoom [ Source  ]
     [ Display ]
     [ Output  ]
```

Where

breaknumber	is a specific breakpoint number. Each breakpoint has a unique breakpoint number assigned to it.
command	is a valid debugger command.
count	is an integer.
counter	is an integer that represents the number of times this breakpoint must be encountered before executing the breakpoint.
exception	is a valid Exception Status.
"*filename*"	is a text filename enclosed in double-quotes.
identifier	is a valid data-item in the current COBOL program. (Requires a symbol file.)
line	is a valid line number in the current COBOL program. (Requires a symbol file.)
"*name*"	is a simple program name enclosed in double-quotes.
procedure	is a valid section or paragraph name in the current COBOL program. (Requires a symbol file.)
"*string*"	is an alphanumeric string enclosed in double quotes.

XIII. ICREVSET

A. Introduction

The ICREVSET utility is used to set the programmer (or supplier) revision field in a file. This utility is separate from the ICREV utility so that it can be removed from the end-user's system, thereby preventing the end-user from modifying his revision information. This utility can set the programmer revision field in any type of standard-header file.

The OEM Version switch (-o|-O rev) of the compiler can also be used to set the OEM revision of .CX files at file creation.

The ACCEPT FROM ENVIRONMENT statement can be used in a COBOL program to extract the programmer revision field from the executing program.

B. Syntax

The standard syntax is:

```
icrevset [-a[:aflag]|-A file|dir[:aflag]] [-h|-?] [-L file] [-q] [-v] rev
   { infile }...
```

Where

-a[:*aflag*]|-A *path*[:*aflag*] (audit)
: Audit to icdump.lg, aflag: a=append, b=backup, d=datestamp, p=process-id, t=timestamp, u=username

-A path[:flag] (audit)
: Audit to path, or path\icrevset.lg if path is a directory

-h|-? (Help)
: Display help text.

-L *file* (Library)
: Use the specified COBOL library to find the specified files.

-q (Quiet)
: Enables quiet operation.

-v (Verbose)
: The name of each file will be displayed as it is processed, as well as the file type and the prior value of the revision field. Otherwise, a simple summary of the number of files processed is displayed.

rev
: Specifies a string of up to 8 characters containing the text the programmer (or supplier) desires in the programmer revision field.

infile
: Specifies a filename or template. The filenames or templates must specify an extension to determine the type of file to modify. If the filename argument specifies a library file, the revision field is set in the library file itself.

C. General Rules

ICREVSET looks for the common environment variables ICROOT and ICTMPDIR.

On UNIX, when using the library switch, if a template is to be specified for *infile*, it may need to be quoted to prevent it from being expanded by the shell.

XIV. ICDUMP

A. Introduction

The Dump utility (ICDUMP) allows the user (usually the programmer) to dump a .CX file or a .PD file to look at a particular COBOL PC for debugging purposes.

B. Syntax

The syntax is:

```
icdump [-a[:aflag]|-A path[:aflag]] [-c] [-d] [-f] [-h|-?] [-l] [-n] [-q]
    [-r] [-x] { filename }...
```

Where

-a[:*aflag*]|-A *path*[:*aflag*] (audit)
: Audit to icdump.lg, aflag: a=append, b=backup, d=datestamp, p=process-id, t=timestamp, u=username

-A path[:flag] (audit)
: Audit to path, or path\icdump.lg if path is a directory

-c (Code)
: Display a dump of the Code area

-d (Data)
: Display a dump of the Data area

-f (File)
: Display a dump of the file info (Rev 5 CX and later)

-h|-? (Help)
: Display help text

-l (Literal)
: Display a dump of the literal area (Rev 5 CX and later)

-n (No-header)
: Do not display the header dump

-q (Quiet)
: Enable quiet operation.

-r (Reference)
: Display a dump of the reference table

-x (External)
: Display a dump of the external area (Rev 5 CX and later)

filename
: Specifies the file to be dumped.

C. Rules

ICDUMP looks for the common environment variable ICROOT.

D. Example

The example below shows the output from an ICDUMP of f_pi.sr, a simple program. It shows the information from the file headers, the program code, the reference table, and the program data segment.

The command line for this example is:

```
icdump -crdfx f_pi
```

The output for this example is shown in four different frames only for explanatory purposes. The above command line produces one stream of output that includes all of the output.

The first frame shows the ICDUMP banner and the dump of the file headers.

```
icdump Revision 3.60 (Windows)
Copyright (C) 1987-2007, Envyr Corporation.  All rights reserved.

Processing f_pi.cx

Decoding of File Headers

Standard Header:
    ICOBOL Executable File Revision 5.00 (byteswapped)
    Created:  Jun-30-2000 08:01:29.00 by icobol 3.00 (Windows 9X/NT/2000)
    Modified: Jun-30-2000 08:01:29.00 by icobol 3.00 (Windows 9X/NT/2000)
    Supplier Revision: none

File Header:
    Revision:       5.00
    PROGRAM-ID:     F-PI
    Currency Char:  $
    Decimal Char:   .
    Comma Char:     ,
    Is Initial:     No
    COBOL Type:     ANSI-74+
    Source Format:  Free format
    Parse options:  (00000000) None
    Start PC:               1       End PC:               21
    Use Input Beg:          0       Use Input End:         0
    Use Output Beg:         0       Use Output End:        0
    Use IO Beg:             0       Use IO End:            0
    Use Extend Beg:         0       Use Extend End:        0
    Code Start:           268       Code Size:            22
    PicLits Offset:        22
    Init Start:           292       Init Size:             2
    Ref Start:            296       Ref Count:             2
    D-V Start:            312       D-V Count:             0
    File Start:             0       File Count:            0
    Using Start:            0       Using Count:           0
    External Start:         0       External Size:         0
    Data Size:             24       CONTENT Size:          0
    External Count:         0       Total Using:           0
```

EXAMPLE 90. ICDUMP of the Header (default)

This frame shows the dump of the program code segment.

```
Decoding of Program Code
Copyright (C) 1987-2007, Envyr Corporation  All rights reserved.

 PC     Operation          Operands


%00001  Compute x ->y...     FUNCTION PI @0
%00007  Display Data Adv     @0
%00011  Compute x ->y...     FUNCTION PI @1
%00017  Display Data Adv     @1
%00021  STOP RUN
```

EXAMPLE 91. ICDUMP of the Program Code (using the -c switch)

This frame shows the dump of the program reference table.

```
Decoding of Program Reference Table

Ref #   File or Data Type       Type-specific Information
        Address (Relocation Information)

    0   Unsigned Display        len: 18, cnt: 18, l: 1, r: 17
        0 (Data Segment)
    1   Unsigned Display        len: 6, cnt: 6, l: 1, r: 5
        18 (Data Segment)
There are no files defined in this program.
```

EXAMPLE 92. ICDUMP of the Reference Table (using the -r switch)

This frame shows the dump of the program data segment.

```
Decoding of Program Data Segment

 The data segment is 24 bytes long
 Offset      Hex Dump                            Char Dump

        0    00 00 00 00 00 00 00 00 00 00       ..........
       10    00 00 00 00 00 00 00 00 00 00       ..........
       20    00 00 00 00                         ....


There are no external data items
icdump is finished
```

EXAMPLE 93. ICDUMP of the Data (using the -d switch)

XV. RUNTIME (ICRUN)

A. Introduction

The **ICOBOL** runtime (ICRUN) is the environment (or soft machine) provided by the **ICOBOL** product that executes COBOL programs. This chapter describes how the runtime works.

ICRUN insulates the COBOL program from many machine and operating system differences. However, where differences are noted one should try to code for the least common denominator of the features. When that is not possible, use the ACCEPT FROM ENVIRONMENT statement to allow the program to know exactly which machine it is running on.

The command line and environment settings for ICRUN are described in the appropriate Installing and Configuring manuals (for UNIX or Windows).

B. Printer Control Utility

The Printer Control utility is provided when enabled from the configuration file (.cfi). The Printer Control utility provides for the spooling and separate printing of files. The Printer Control utility uses the printer control file to hold the filenames that are currently in the printer control queue. By default, the printer control file is system.pq. The printer control file can handle up to 1024 files based on what the configuration file (.cfi) has allowed. Once that maximum is reached, an OPEN of a file that would have been placed in the printer control file will fail with a File Status 99 (Exception Status 44).

The Printer Control subsystem can be configured to automatically print a file once it has been entered into the printer control file or to allow each file to be queued separately to a printer by a user. To manually queue files to be printed, the IC_PRINT_STAT builtin must be executed to start the Printer Control Utility. To automatically have files printed, the AUTO option must be set in the configuration file (.cfi) for the particular queue.

At startup time, the printer control file is scanned and if an entry or file no longer exists on the disk, the entry is removed from the system.pq file and the printer control queue.

On UNIX

The printer control file is read at ICEXEC startup to load the queue and is kept updated while running.

The UNIX print spooler, *lp*, is used to provide the actual printing of jobs. The following UNIX command is executed to print a job:

lp -dpcqdest -tsimple-filename filename

where pcqdest is the destination defined for the particular printer control queue in the configuration file (.cfi), simple-filename is the simple part of the filename as the title, and then the file to be printed. If the destination field is blank the default queue is used.

On Windows

The Windows print spooler is used to provide the actual printing of jobs. ICEXEC reads the configuration file (.cfi) and determines available print queues and matching Windows printers (except for a default (blank name) which is still setup by the runtime system when it starts.

C. Program Termination

If running in Program mode, **ICOBOL** will return to the shell (or parent process) on any of the following:

1) STOP RUN
2) CALL"IC_HANGUP", "IC_SHUTDOWN"
3) CALL PROGRAM #H, #S, ##U
4) console interrupt
5) Fatal error.

C.1. Two Types of Termination

Two types of COBOL program termination are provided under **ICOBOL** when running in Logon mode.

C.1.1 Return to LOGON as Inactive

The first type of program termination stops the COBOL program and returns control to the program LOGON as an inactive terminal. This is done by:

Action	Status
1) CALL "IC_LOGON"	Set to Inactive
2) STOP RUN and then press ENTER	Set to Stopped Set to Inactive
3) Pressing the Intr key (if allowed by the configuration file (.cfi)) and then press ENTER	Set to Stopped Set to Inactive
4) Program errors that terminate the program, e.g., Fatal COBOL I/O errors, and then press ENTER	Set to Stopped Set to Inactive

NOTE: If a fatal I/O error is encountered and the program terminates, the current Exception Status is displayed right after the COBOL PC as E=nnn.

C.1.2 Return to Parent Process

The second type of program termination terminates the **ICOBOL** process for that terminal and returns the user to the parent process. This is done by one of the following:

```
CALL  "IC_HANGUP"
CALL  "IC_SHUTDOWN"
```

If the parent process was ICEXEC, then ICEXEC will cause the initial logon prompt to be re-displayed.

D. Device Support

D.1. Overview

The mapping from an **ICOBOL** logical device to a particular hardware device can be configured in the configuration file (.cfi) under the Device Configuration menu.

It is recommended that only logical devices be used in a COBOL program so that it will be insulated from a particular operating system and/or a particular hardware configuration.

The **ICOBOL** logical devices are:

- console devices (@CON0, @CON1, @CON2, and up),
- printer devices (@PRN0 through @PRN127),
- printer control queues (@PCQ0 through @PCQ127),
- serial devices (@SER0 through @SER127).

There are eight generic logical devices:

- @NUL always maps to the internal null device,
- @CON always maps to the current console,
- @PTS always maps to the current console with printer pass thru mode enabled for each WRITE operation,
- @PCQ, @PRN, and @SER can be configured, on a per console basis with the PCQ, PRN, or SER entries, to point to any of their respective logical devices, e.g., PCQ=1 is used to set the generic @PCQ to point to @PCQ1. If no entry is specified, the default values for @PCQ, @PRN, and @SER are @PCQ0, @PRN0, and @SER0 respectively. (These can be set as environment variable(s) to override those defined in the configuration file (.cfi).)
- @DATA maps to the current contents of the environment variable DATAFILE at runtime.
- @LIST maps to the current contents of the environment variable LISTFILE at runtime.

D.2. General Rules

(1) If **ICOBOL** detects any operating system name from the table below, it is replaced with the corresponding **ICOBOL** name as shown. This replacement is not done if the name was mapped using the link file.

Operating System Name	ICOBOL Name	Operating System Name	ICOBOL Name
NUL	@NUL	@INPUT	@CON
CON	@CON	@OUTPUT	@CON
$TTI	@CON	@CONSOLE	@CON
$TTO	@CON	@LPT	@PCQ0
PRN	@PRN0	@LPT1	@PCQ1
LPT1	@PRN0	@LPT2	@PCQ2
LPT2	@PRN1	@LPT3	@PCQ3
LPT3	@PRN2	@LPT4	@PCQ4
$LPT	@PRN0	@LPT5	@PCQ5
$LPT1	@PRN1	@LPT6	@PCQ6
AUX	@SER0	@LPT7	@PCQ7
COM1	@SER0	@LPT8	@PCQ8
COM2	@SER1	@LPT9	@PCQ9
COM3	@SER2	...	...
COM4	@SER3	@LPT2048	@OCQ2048
QTY:0	@CON1		
QTY:1	@CON2		
QTY:2	@CON3		
...	...		

TABLE 42. Device Mappings

All the device names in TABLE 42 are mapped in a case insensitive manner. D.G., `con', `CON', `Con', and `cOn' all specify the same device name to **ICOBOL**. To override an **ICOBOL** name specify a pathname, i.e., `=con', `./CON' or `./con', in which case **ICOBOL** will not find its device but will pass the name on through to the operating system.

(2) **ICOBOL** does not open hardware devices it controls with the EXCLUSIVE option unless explicitly set.

(3) The logical filename @NUL is a special internal device. If you use @NUL as an input device, a read will always generate an immediate end-of-file. As an output device, the write operations are simulated, but no data is actually written.

(4) Each logical console device (@CONn) is either enabled or not. If enabled, the logical console device has a character device (**on UNIX** 'from /dev'), a blank for use with the Terminal number switch (-T) or when terminal devices are not defined in the configuration file (.cfi), or a null for use with the IC_DETACH_PROGRAM builtin. Logical consoles also have the ability to Run Programs if the Run Program option is set to Yes. If Run Programs is set to Yes, the Program environment options are used. The line-number returned by the ACCEPT LINE statement of a COBOL program is the number n of the @CONn logical console name. When **ICOBOL** starts if the ttyname for

the console cannot be found in the the configuration file (.cfi) Console table or if it is already in use, it will scan the console table for an entry that is enabled, has a blank device, and is currently not in use as its console.

(5) Each logical printer control queue (@PCQn) has associated with it a standard UNIX print queue (**on UNIX**) or a standard Windows printer (**on Windows**). These must be defined and enabled before **ICOBOL** can make use of the @PCQn devices. Whenever an @PCQn logical device is opened, the output is routed to the particular print queue (**on UNIX**) or printer (**on Windows**) defined for the destination as setup in the configuration file (.cfi).

On UNIX, this is done using the UNIX print spooler (*lp*) and is called intercept spooling. Thus if the COBOL filename opened was `@PCQ1', **ICOBOL** would pipe the written output to

lp -dpcq1destination.

On Windows, this is done using standard Windows print routines to place the file in the Windows printer subsystem and is called intercept spooling.

(6) Each logical printer device (@PRNn) has associated with it some printer options such as form-feed on open and/or close upon printing.

On UNIX, you should be very careful when printing directly to a device that the UNIX print spooler is using as there is no standard way to provide EXCLUSIVE access to that output device. You should either use printer control queues (@PCQn), the Printer Control Utility, or use a device that is not being used by the UNIX print spooler.

(7) The logical serial devices (@SERn) are serial communication ports on which no programs can be run, but serial input and/or output can be performed.

(8) If the hardware for a particular device is not installed in the system, not enabled, or set to None; a File Status 91 is returned on the OPEN.

(9) **On UNIX**, standard UNIX character devices that reside in /dev can be used as appropriate for files.

D.3. Parallel Printer Ports

(1) Parallel printer ports are generally the lp0, lp1, and lp2 devices.

(2) On output to a parallel port, if a timeout value was not specified on the OPEN, the write will try forever. If the timeout and message options are both set on an extended device open, the message displayed on the user's screen will show the actual reason (like offline, out of paper, I/O error, etc.) that is causing the write to wait.

(3) On a CLOSE of a parallel port, if its buffer still has characters waiting to be written, the CLOSE will delay up to 5 minutes to enable the buffer to be flushed. After that time the parallel port will be closed and the buffer reset.

D.4. Serial Ports

(1) The serial ports are generally the tty01, tty02, . . . or tty1a, tty1b, . . . devices.

(2) If an OPEN specifies a device with modem control enabled, the OPEN will wait until Data Carrier Detect (DCD) is detected before preceding (this could wait forever). For all other operations, if DCD is not detected, a File Status 30 (Exception Status 122) will be returned.

(3) If hardware errors, such as parity, are detected, a File Status 30 (Exception Status 13) is returned for the operation in progress.

(4) Program lines cannot be opened by a program on another terminal. The other program will get a File Status 94 (file is exclusively opened).

(5) On output to a serial line that is not the current console, if a timeout value was not specified on the OPEN, the write will try forever. A timeout option along with a possible message option can be specified as an extended device open option on the OPEN to change this behavior.

(6) On a CLOSE of a serial line, if a timeout value was not specified on the OPEN, the CLOSE will try forever. If a timeout had been specified the CLOSE will complete in that time, the line closed, and the buffer reset.

E. Filenaming Conventions

E.1. Internal Filenames

An *internal filename* is assigned to an external file by using the SELECT clause in a COBOL program. The I/O statements in the program then refer to this file by its internal name, as in OPEN INPUT FILE-ONE.

E.2. External Filenames

An *external filename* is the name by which a file is known to the operating system and/or **ICOBOL** environment. This section, describes how **ICOBOL** handles COBOL external filenames. Also see the sections in the **ICOBOL** Language Reference that discuss the COBOL builtins IC_RENAME_FILE, IC_GET_DISK_SPACE, IC_DIR_LIST, IC_MOVE_FILE_DATA).

In the SELECT clause of a COBOL program, the external filename can be specified. If no external filename is specified for a SELECT, the **ICOBOL** compiler generates a default external filename based on the ASSIGN TO <device> clause as defined in Table 1 on page 100 Default External Filenames.

ICOBOL considers the following as **legal** characters in a filename:

Characters	Description
a-z	Lower-case letters
A-Z	Upper-case letters
0-9	Digits
.	Period
_	Underscore
$	Dollar sign
-	Hyphen
!	Exclamation
%	Percent
&	Ampersand
{}	Left- and right-brace
( )	Left- and right-parenthesis
~	Tilde

TABLE 43. Legal characters in a filename

ICOBOL treats the following characters as **illegal** characters in a filename:

Character	Description
` '	Open- and close-single-quote
"	Double quote
[ ]	Left- and right-bracket
*	Asterisk
#	Pound-sign
+	Plus-sign
|	Vertical-bar
< >	Left- and right-angle-bracket
;	Semicolon
?	Question-mark
ALSO: Embedded spaces (only if the -N e option is used), characters less than space, and characters greater than tilde.	

TABLE 44. Illegal Characters in a Filename

In certain contexts, the following characters are allowed in a filename:

Character	Description
equal (=)	As the first character of a filename, an equal is replaced with the current directory
caret (^)	As the first character of a filename, a caret is replaced with the parent directory
colon (:)	All occurrences are converted to the appropriate directory separator - `\' on Windows - `/' on UNIX **NOTE:** On Windows, : is not converted if immediately following a single letter at the beginning of the name. (Drive-letter)
backslash (\)	All occurrences are converted to the appropriate directory separator - `\' on Windows - `/' on UNIX)
forwardslash (/)	Treated as a directory separator and converted to the appropriate directory separator - `\' on Windows - `/' on UNIX; except when given on a program name when single-character program switches are stripped off the program name by scanning from the end
at-sign (@)	As the first character of an **ICOBOL** logical device name specifier

TABLE 45. Characters Allowed in a Filename, in Certain Contexts

E.2.1 Rules

(1) **ICOBOL** always trims leading and trailing spaces before any other processing is done to a filename.

(2) A simple filename cannot be longer than 255 characters, and the pathname cannot be longer than 255 on Windows and 1023 on UNIX. If it is, an error is given.

E.2.2 Program names

E.2.2.1 Overview

The simple portion of a program name in **ICOBOL** must be 30 characters or less; otherwise, an error is given.

In addition to length, there are other things **ICOBOL** considers when processing a program name, and the processing is different for each of the operations that use program name:

- CALL statement
- CALL PROGRAM statement
- CANCEL statement
- IC_DETACH_PROGRAM builtin

The table below defines processes that are used in some of the operations listed above, as **ICOBOL** processes a program name. The section that follows the table will describe how **ICOBOL** processes a program name for each of the operations and will use the name of the definition from the table (e.g., *Strip program switches*) to simplify the explanation.

Strip program switches:

When searching for program switches, the runtime first checks to see if the switches were delimited by a space, in which case the switches are stripped to the right of the space with no other embedded space allowed. For example, "/usr/a/b /c/d" would treat c and d as switches with the program name being "/usr/a/b". If not delimited by a space, **ICOBOL** scans backward from the end of the string picking off "/character" pairs until either no more valid pairs exist or the beginning of the name. For example, "/usr/a/b/c/d" would treat a, b, c, and d as switches with the program name being "/usr". See page 304 for a complete description on program switch processing.

CALL check:

When searching for an active or inactive program during a CALL, **ICOBOL** uses the simple part of the filename in a case-insensitive fashion. If there is already an active program with that name, it is an error (recursion). If there is already an inactive program with that name, the program is activated. Otherwise, **ICOBOL** uses the name to activate a new program.

Check ICCODEPATH:

ICCODEPATH specifies a list of directories and/or COBOL library files in which to look for COBOL programs with simple names. If ICCODEPATH is not specified, the simple name is passed to the operating system. Which will look in the current directory. If ICCODEPATH is specified, each directory and/or COBOL library is searched sequentially to find the given program file.

E.2.2.2 CALL Statement

Program name is processed as follows for a CALL statement:

1) **On UNIX**, check if this is a call to the operating system (`|' as the first character) and process accordingly;
2) Strip program switches;
3) **On UNIX**, convert the name to the case specified by **ICOBOL** (default lower-case);
4) Search for name in the link file and replace with new name if found;
5) If a simple name;
   a) check for user-defined subroutines (calls added with the Link Kit) or builtins and process if found;
   c) CALL check;
   d) check for invalid characters;
   e) append `.cx' and check ICCODEPATH. If not found, give an error.
6) If not a simple name;
   a) CALL check;
   b) check for invalid characters;
   c) append `.cx', resolve the name, and look up the file in the operating system. If not found, give an error.

E.2.2.3 CALL PROGRAM Statement

Program name is processed as follows for a CALL PROGRAM statement:

1) Check if this is a system call (prefix of # or ##), process if so
2) Strip program switches
3) On UNIX, convert the name to the case specified by **ICOBOL** (default lower-case)
4) Search for name in the link file and replace with new name if found
5) Check for invalid characters
6) If a simple name, append `.cx' and check ICCODEPATH. If not found, give an error.
7) If not a simple name, append `.cx', resolve the name, and look up the file in the operating system. If not found, give an error.

E.2.2.4 CANCEL Statement

Program name is processed as follows for a CANCEL statement:

1) Strip program switches
2) **On UNIX**, convert the name to the case specified by **ICOBOL** (default lower-case)
3) Search for name in the link file and replace with new name if found
4) Extract the simple name and check to see if there is already an active program with that name, if so give an error (active), next check to see if there is already an inactive program with that name, if so, CANCEL the program
5) Otherwise ignore.

E.2.2.5 IC_DETACH_PROGRAM builtin

Program name is processed as follows for an IC_DETACH_PROGRAM builtin:

1) Extract the program part of the name, (up to the first space),
2) Strip program switches
3) **On UNIX**, convert the name to the case specified by **ICOBOL** (default lower-case)
4) Search for name in the link file and replace with new name if found
5) Check for invalid characters
6) If a simple name, append `.cx' and check ICCODEPATH. If not found, give an error.
7) If not a simple name, append `.cx', resolve the name, and look up the file in the operating system. If not found, give an error.

E.2.3 Sequential and ICISAM Filenames

E.2.3.1 Overview

There are several different operations for which **ICOBOL** needs to process a filename for a sequential or ICISAM file, and for each operation the process is different. Here are the operations that use a sequential or ICISAM filename:

- OPEN Statement
- DELETE FILE Statement, along with the IC_DIR_LIST, IC_GET_DISK_SPACE, IC_MOVE_FILE_DATA builtins
- IC_RENAME builtin

The table below defines a process that is used in some of the operations listed above, as **ICOBOL** processes a sequential or ICISAM filename. The section that follows the table will describe how **ICOBOL** processes a filename for each of the operations and will use the name of the definition from the table (i.e., *Check ICDATAPATH*) to simplify the explanation.

Check ICDATAPATH:

ICDATAPATH specifies a list of directories and/or COBOL library files in which to look for COBOL data files with simple names. If ICDATAPATH is not specified, the simple name is passed to the operating system, which will look in the current directory. If ICDATAPATH is specified, each directory and/or COBOL library is searched sequentially to find the given file. If not found, and the creation attribute is specified, the file will always be created in the current directory regardless of the ICDATAPATH.

E.2.3.2 OPEN Statement

Filename is processed as follows for an OPEN statement:

1) On UNIX, check for a pipe open ('|' as the first character); if so, process the OPEN.
2) Strip any extended open options (i.e., the comma-separated list)
3) On UNIX, convert name to the case specified by **ICOBOL** (default lower-case)
4) Search for name in the link file and, if found, replace with new name, otherwise, map all RDOS, AOS/VS, and MS-DOS names to their **ICOBOL** logical name as defined in TABLE 42 on page 749.
5) On UNIX, check for a pipe open ('|' as the first character) if so process the OPEN.
6) Check for invalid characters
7) If a simple name, check ICDATAPATH appending ICISAM extensions if needed. If not found, give an error or create in the current directory as required by the OPEN.
8) If not a simple name, append ICISAM extensions if needed, resolve the name, and look up the file in the operating system. If not found, give an error or create in the specified directory as required by the OPEN.
9) Process the OPEN.

NOTE: A pipe open does not allow extended open options. Extended open options are discussed beginning on page 756.

E.2.3.3 DELETE FILE Statement along with IC_DIR_LIST, IC_GET_DISK_SPACE, IC_MOVE_FILE_DATA builtins

Filename is processed as follows for a DELETE FILE statement and the above named builtins and system calls:

1) On UNIX, convert name to the case specified by **ICOBOL** (default lower-case)
2) Search for name in the link file and replace with new name if found
3) Check for invalid characters
4) Append ICISAM extensions if needed, resolve the name, and look up the file in the operating system. If not found, give an error.
5) Process as specified.

E.2.3.4 IC_RENAME builtin

Filename is processed as follows for IC_RENAME builtin:

1) On UNIX, convert the name to the case specified by **ICOBOL** (default lower-case)
2) Check for invalid characters
3) Rename.

E.2.3.5 IC_DETACH_PROGRAM builtin (for the output file)

Filename is processed as follows for IC_DETACH_PROGRAM builtin:

1) If specified, extract the output filename
2) On UNIX, convert the name to the case specified by **ICOBOL** (default lower-case)
3) Search for name in the link file and, if found, replace with new name, otherwise, map all RDOS, AOS/VS, and MS-DOS names to their **ICOBOL** logical name as defined in TABLE 42 on page 749.
4) Check for invalid characters
5) Resolve name to a fully qualified name
6) Process the OPEN.

F. Extended OPEN options

F.1. Overview

Extended open options are available to allow specification of certain items at open time that may not be known when the COBOL program is written. The extended open options are a comma-separated list of options that allow the COBOL programmer to tailor the reads, writes, rewrites, and closes based on information known only at runtime. Within the extended open options, spaces are ignored.

The extended open options are specific to the file organization (sequential, relative, indexed or infos) and are described in the following sections. **ICOBOL** checks the options for validity and will return an error for invalid options or option value.

F.2. Extended Sequential Open

The extended open options for sequential files can be further sub-divided into the following three categories:

OPEN OPTION	DESCRIPTION
Extended Device Open	For all opens that resolve to direct access to a hardware device.
Extended PCQ Open	For all opens that will either be explicitly (OPEN "@PCQn") or implicitly (ASSIGN TO PRINTER or PRINTER-1) placed into the printer control file.
Extended Disk Open	All other cases.

TABLE 46. Three Categories of Extended Open for Sequential Files

F.2.1 (Sequential) Extended Device Open

Extended device open options are allowed for all opens that resolve to direct access to a hardware device. The attributes for any hardware device, including the current console, can be reset at open by using the extended device open options.

F.2.1.1 ***ANSI 74*** and ***ANSI 85*** syntax is:

```
device [,t=timeout] [,e=retries] [,m=y|n] [,r=record-size] [,b=baud]
   [,p=n|o|e] [,d=8|7] [,s=1|2] [,f=b|n|i|o]
```

F.2.1.2 ***VXCOBOL*** syntax:

```
device [,t=timeout] [,b=baud] [,p=n|o|e] [,d=8|7] [,s=1|2] [,f=b|n|i|o]
```

Where

device

Is any name that resolves to direct access to a hardware device. These can include the logical console, serial, and printer devices of @CONn, @SERn, or @PRNn.

t=*timeout*

Sets the timeout in tenths of seconds. This is the maximum inter-character time to wait for the device to respond before returning an exception on the I/O operation. For a CLOSE, it is the amount of time to wait to flush buffers before closing the line and resetting the buffers. Valid values are 65535 to wait forever, and 0 - 63000 for that number of tenths of seconds. A File Status 9T (Exception Status 76 "Device timeout") will be returned if the timeout is taken.

e=*retries* (***ANSI 74*** and ***ANSI 85***)

Sets a retry count when writing data-sensitive records. Valid values are 0 - 63. If any one of the Exception Status values above occurs on the write, the specified number of retries will be performed before the exception is returned to the COBOL program.

m=y|n (***ANSI 74*** and ***ANSI 85***)

Specifies that when performing an exception retry to display a message (with a beep) indicating which exception occurred followed by "Retrying. . ." on the bottom of the user's screen. If the exception condition is overcome before the retry count is exhausted, the message is erased. Otherwise, a "Retry failure. . ." message will be displayed.

r=*record-size* (***ANSI 74*** and ***ANSI 85***)

Overrides the record size specified at compile time for this FD. The new record-size must be less than or equal to that specified in the COBOL FD. May not be specified for an EXTERNAL file. For variable-length files the maximum record size is set.

b=*baud*

Sets the baud to one of following legal values, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56000.

p=n|o|e
: Sets the parity to none, odd, or even

d=8|7
: Sets the number of data bits to 8 or 7

s=1|2
: Sets the number of stop bits to 1 or 2

f=b|n|i|o
: Sets both Software Input Flow Control (SIFC) and Software Output Flow Control (SOFC), neither SIFC nor SOFC, SIFC only, or SOFC only, respectively. When used on a program line, the console interrupt option will be disabled.

F.2.1.3 Rules

(1) If an attribute is not given, its value is taken from the default set by the operating system.

(2) If a particular option is not applicable for the final hardware device, it is ignored, e.g., baud rate for a parallel port.

(3) The timeout (t=), retries (e=), message (m=), and record-size (r=) are set on a per COBOL FD basis.

(4) The remaining options, baud (b=), parity (p=), data-bits (d=), stop-bits (s=) and flow control (f=), are the hardware options and will affect all subsequent I/O on this device until a close is done, at which time the device is reset to its default value. All closes of a device will reset it to its default state. Thus if you re-open your console with a new baud rate, the new baud will be in effect until you close the file. Also if you re-open your console with a flow-control option set the console interrupt option will be disabled until you close the file. For example "@con,f=n" will open the current console with the current baud, parity, data-bits, and stop-bits while disabling console interrupt and having both software flow control (SIFC and SOFC) options disabled. This is useful for doing file transfer via a terminal emulator.

(5) A STOP RUN or any other program termination will do a CLOSE, forcing a reset of the options on the current console to the default state.

(6) Mdm Ctl (modem control) and Hrd OFS (hardware output flow control) options are never reset due to an extended device open.

(7) An example of the message retry count would be "...,t=3000,e=10" which specifies that the timeout be set to 5 minutes and the retry count set to 10. Thus before the program gets an error the WRITE will delay for 10 times 5 minutes or 50 minutes. While retrying, if the WRITE can continue, the message will be erased.

F.2.2 (Sequential) Extended PCQ Open (***ANSI 74*** and ***ANSI 85***)

For all opens that will have entries either explicitly, with an OPEN "@PCQn", or implicitly, using an ASSIGN TO PRINTER or PRINTER-1, placed into the printer control file, allow extended pcq open options to be specified.

The syntax is:

```
filename [,r=record-size] [,i=position] [,q=n] [,d=k|r|d] [,p=priority]
   [,c=copies] [,n=y|n] [,a=y|n]
```

Where

filename
: Is any disk filename or @PCQn.

r=*record-size*
: Overrides the record size specified at compile time for this FD. The new record-size must be less than or equal to that specified at compile time.

i=*position*
: Overrides the initial position to start performing I/O operations. With this option you can start reading at any byte in the file. For example, "tmp,i=1024" would allow you to start reading or writing at the 1024th byte. Zero is the beginning of file. If not specified, the default is beginning-of-file for all OPENs except OPEN EXTEND, in which case it is end-of-file. This option may not be specified for an EXTERNAL file.

q=*n*
: Places this file in the printer control file with its default queue set to n. n must be an enabled printer control queue (0-2047) or else an error will be raised - file status 91, Exception Status 81 "Device is not available or does not exist". (The QUEUE IS phrase of the file control entry may be used for the same purpose.)

d=k|r|d
: Sets the disposition option to keep (k), remove (r), or delete (d) when this file is placed into the printer control file.

p=*priority*
: Sets the priority to this value when this file is placed into the printer control file. Valid values are 1 - 255.

c=*copies*
: Sets the copies option to this value when this file is placed into the printer control file.

n=y|n
: Sets the notify option to yes or no when this file is placed into the printer control file.

a=y|n
: Sets the auto print option to yes or no when this file is placed into the printer control file.

F.2.3 (Sequential) Extended Disk Open (***ANSI 74*** and ***ANSI 85***)

The extended disk open for all other sequential files has the following syntax:

```
filename [,r=record-size] [,i=position]
```

Where

filename
: Is any disk filename.

r=*record-size*
: Overrides the record size specified at compile time for this FD. The new record-size must be less than or equal to that specified at compile time. This option may not be specified for an EXTERNAL file. For variable-length files the maximum record size is set.

i=*position*
: Overrides the initial position to start performing I/O operations. With this option you can start reading at any byte in the file. For example, "tmp,i=1024" would allow you to start reading or writing at the 1024th byte. Zero is the beginning of file. If not specified, the default is beginning-of-file for all OPENs except OPEN EXTEND, in which case it is end-of-file.

F.3. Extended Relative Open (***ANSI 74*** and ***ANSI 85***)

The extended open for relative files has the following syntax:

```
filename [,v=5|6|7] [,p=y|n] [,r=record-size]
```

Where

filename
: Is any disk filename.

v=5|6|7
: Specifies that if this particular file is to be created it should be created as an ICISAM version 5, 6, or 7 file. Versions 5 and 6 may only be specified for **ANSI 74**.

p=y|n
: Specifies the delete-is-physical attribute for version 7 files. This attribute is only valid for version 7 files and the default is n(o), if not specified. This option may not be specified for an EXTERNAL file.

r=*record-size*

Overrides the record size specified at compile time for this FD. The new record-size must be less than or equal to that specified at compile time. This option may not be specified for an EXTERNAL file. For variable-length files the maximum record size is set.

If an existing file is being opened, all of the specified new parameters must match the attributes of the current file.

F.4. Extended Indexed Open

The extended open for indexed files has the following syntax:

ANSI 74 and ANSI 85

```
filename [,v=5|6|7] [,b=c|b|i] [,r=record-size] [,n=number-keys] [,p=y|n]
   [,o=offset]... [,l=length]... [,d=y|n]... [,u=y|n]...
```

VXCOBOL

```
filename [,b=c|b|i]
```

Where

filename

Is any disk filename.

b=c|i|b

Specifies the record manager to use. To use ICISAM (which is always available) use i. **On Windows** to use BTRIEVE, use `b'. **On UNIX** to use C-ISAM, use `c'. The default is that specified to the runtime with the Brand switch (-B) when it started. BTRIEVE is only available **on Windows** when a valid BTRIEVE engine is present and a BTRIEVE option is provided to the runtime license. C-ISAM is only available **on UNIX** when a C-ISAM option is provided to the runtime license.

v=5|6|7

Specifies that if this particular file is to be created it should be created as an ICISAM version 5, 6, or 7 file. Versions 5 and 6 may only be specified for **ANSI 74**.

r=*record-size*

Overrides the record size specified at compile time for this FD. The new record-size must be less than or equal to that specified at compile time. This option may not be specified for an EXTERNAL file. For variable-length files the maximum record size is set.

n=*keys*

Overrides the number of keys specified at compile time for this FD. The new number of keys must be less than or equal to that specified at compile time. This option may not be specified for an EXTERNAL file.

p=y|n

Specifies the delete-is-physical attribute for version 7 files. This attribute is only valid for version 7 files and the default is n(o), if not specified. This option may not be specified for an EXTERNAL file.

o=*offset*

Overrides the offset for each of the keys specified in the order primary, alternate-1, alternate-2, etc. Offset is zero(0) based, thus o=0 means the first byte in the record. The new offset must be within the old record size and within the new record size, if specified. This option may not be specified for an EXTERNAL file.

l=*length*

Overrides the length of each of the keys in the specified order primary, alternate-1, alternate-2, etc. The length must not be larger than 100 and the key area must be completely contained within the old and new record areas. This option may not be specified for an EXTERNAL file.

d=y|n

Overrides whether duplicates are allowed or not for a key. No duplicates set are only allowed for version 7 indexed files. This option may not be specified for an EXTERNAL file.

u=y|n

Specifies whether to convert all key entries for this key to upper-case. The default is no. Upper-case is only allowed for version 7 indexed files. This option may not be specified for an EXTERNAL file.

NOTES:

1. The o, l, d, and u options can be repeated for the number of keys specified by the n option, or by the number of keys defined in the FD at compile time.

2. The o, l, d, and u options may also be specified as:

   [,o=offset,l=length,d=y|n,u=y|n]...

   It is easier to see what is being done with this syntax.

3. The offset (o) and length (l) options may not be used with keys whose file control entry specifies the PLUS, ALSO, or OCCURS clause.

With these options, it is possible to write programs that do not know the format of a particular file. The format can be entered or read from a table to allow a generic program to read an ICISAM file and create a report. The first key offset, length, duplicate is the primary key, the second key offset, length, duplicate, is the first alternate, the third set is the second alternate, and so on. There is NO re-ordering of the alternate keys like the compiler performs.

If an existing file is being opened, all of the specified new parameters must match the attributes of the current file.

G. ICISAM Information

G.1. Overview

ICOBOL supports three versions of indexed and relative files (5, 6, and 7). All ICISAM files consist of two separate files. The .XD file contains a header along with all the actual data. The .NX file contains a header along with all the index b-trees for the specified keys to look up records in the .XD part of the file. These two files are both required to successfully use an ICISAM file.

All ICISAM files are created as version 7 files unless the extended open option is used to explicitly specify version 5 or 6.

G.2. ICISAM Versions

Version 7 indexed files are compatible with revisions 3.30 and higher of *ICHOST*. Indexed blocks are allocated as 2048-byte entries. The headers in the .XD and the .NX contain duplicated data allowing for verification at open time and allow the entire file to be rebuilt using only the .XD portion. Version 7 indexed files support the ability to physically delete records, i.e., a DELETE places the record on a reuse chain such that the next WRITE will use that record position rather than allocate a new record area. UNDELETEs cannot be performed when a record has been physically deleted. The default for revision 7 indexed file is for delete-is-physical to be disabled.

Version 7 indexed files support up to 16 alternate keys, each alternate can allow or not allow duplicates, have a maximum record size of 16384 bytes (16KB), and a maximum key size of 255 bytes. Version 7 indexed files also allow a particular key path to be set to only add and lookup key entries in upper-case. When set to upper-case mode, all key entries for this key are converted to upper-case before being added or looked up in the index. Version 7 indexed files also support descending keys, suffixed keys, multiple locations per key, suppressed key values, etc. Version 7 indexed files keep a deleted record count in the header.

Version 7 indexed files and relative files are the recommended versions to use unless a compatibility issue is involved. By default, **ICOBOL**3 creates version 7 files but can access any version. **ANSI 74** can create versions 5 and 6 with the ,v= extended open option. The ICCREATE utility can also be used to create files of any version.

ICISAM version 5 files are compatible with pre-1.50 versions of **ICOBOL**. Index blocks are allocated as 512-byte entries. Version 5 files are limited to a 32MB .NX file. Since index blocks are so small, the maximum number of index levels of 6 could be hit with large keys. Version 5 files should only be used for compatibility.

ICISAM version 6 files are compatible with post-1.50 versions of **ICOBOL**. Index blocks are allocated as 2048-byte entries. Hitting the maximum number of index levels is much harder even with large keys. Version 6 relative files also have a relative indicator in the header versus the version 5 relative files which looked like an indexed file.

Both version 5 and version 6 indexed files support up to 4 alternate keys, always allow duplicates on alternate keys, have a maximum record size of 4096 bytes (4KB), and a maximum key size of 100 bytes. All deletes are logical deletes, i.e., a record can be undeleted. Version 6 files keep a deleted record count in the header.

Both version 5 and 6 relative files support up to 65535 entries using a 2-byte relative key that can go from 0 to 65534. Internally these are implemented as a single-key indexed file whose key is not in the record specified by the user. Version 7 relative files use a 4-byte relative key that can go from 0 to 4,294,967,294.

NOTE: ***VXCOBOL*** always uses ICISAM for ANSI alternate keyed indexed files and for relative files. ***VXCOBOL*** uses ICISAM for single-keyed files if the -G s switch is specified to **ICOBOL** when using the ***VXCOBOL*** dialect (-D vx).

G.3. ICISAM Reliability

ICOBOL's file reliability system helps to insure the logical structure of ICISAM files.

The .XD header of each ICISAM file contains two flag bits, one for the .XD file and one for the .NX file. For an open ICISAM file, the appropriate flag is set by **ICOBOL** when that portion of the ICISAM file has been modified and the modification has not been flushed to disk. These reliability flags are only cleared by **ICOBOL** when it is sure the disk image for the file is logically correct. This is done whenever the file is CLOSE'd by any program, an index root node splits, or a WRITE or REWRITE with the IMMEDIATE option is performed.

If for some reason the system terminates while either one or both of the reliability flags are set, neither **ICOBOL** nor most of its utilities will be allowed to OPEN the ICISAM file. The ICCHECK utility must be run on the file to determine if there really is a problem and if so what the problem is. If no problem exists, ICCHECK will clear the reliability flags.

G.4. ICISAM Key Ordering

The ordering of alternate keys that ALLOW DUPLICATES can be slightly different between version 5 and 6 ICISAM files and a version 7 ICISAM file. This order is used for sequential access (READ NEXT and READ PREVIOUS) when using that alternate key path.

In ICISAM versions 5 and 6, the order of alternate keys with the same key is the order in which the record itself is positioned in the .XD portion of the file. This is generally the order in which the record was "first" written to the file.

In ICISAM version 7, the order of alternate keys with the same key is the order in which the key-itself was written to the file. It has nothing to do with the order in which the record physically resides in the data portion of the file.

The order for both version 5/6 and version 7 will be the same for records that have been written to the file for the first time. The order may be different if REWRITE's are done to the record CHANGING the particular alternate key value. Version 7 will position this changed value at the end of the duplicate key path for that alternate.

ICREORG's on any ICISAM file has the possibility to CHANGE the order that alternate keys with the same value will be positioned in the alternate key path since the record and the key will have been rewritten in the order specified by the ICREORG. (Default is primary key order which is not necessarily the original order.)

Example

Record-A (primary=1 alternate=dave)
Record-B (primary=3 alternate=mary)
Record-C (primary=5 alternate=dave)
Record-D (primary=7 alternate=albert)
Record-E (primary=9 alternate=mary)

**** EXAMPLE 1 ****

Write records A, B, C, D, E (to an empty file)
Read next on alternate (version 6): records D, A, C, B, E
Read next on alternate (version 7): records D, A, C, B, E

Rewrite record D changing alternate to mary
Read next on alternate (version 6): records A, C, B, D, E
Read next on alternate (version 7): records A, C, B, E, D

ICREORG the above version 7 file to a new version 7 file using just the defaults will re-order the alternate to the version 6 way.

```
icreorg filea fileb
```

Read next on alternate (version 6): records A, C, B, D, E (using fileb)
Read next on alternate (version 7): records A, C, B, D, E (using fileb)

**** EXAMPLE 2 ****

Write records C, A, B, D, E (to an empty file)
Read next on alternate (version 6): records D, C, A, B, E
Read next on alternate (version 7): records D, C, A, B, E

Rewrite record D changing alternate to mary
Read next on alternate (version 6): records C, A, B, D, E
Read next on alternate (version 7): records C, A, B, E, D

ICREORG the above version 7 file to a new version 7 file using just the defaults will re-order the alternate to the version 6 way.

```
icreorg filea fileb
```

Read next on alternate (version 6): records A, C, B, D, E (using fileb)
Read next on alternate (version 7): records A, C, B, D, E (using fileb)

In **ICOBOL 3**, to create a version 5 or 6 file the extended open option of ",v=5" or ",v=6" must be added to the filename at OPEN.

H. Notes and Warnings

Many early versions of **ICOBOL** detect SIZE ERROR incorrectly when the receiving item is SIGNED COMP (and the ANSI switch was NOT used on the 1.xx Interactive COBOL compiler). Current versions of **ICOBOL** determine SIZE ERROR based on whether the binary value of the absolute value of the result will fit in the number of bytes. For example, a PIC S99 COMP takes 1-byte and can store (in binary) -128 to 127.

On UNIX

ICOBOL handles the following UNIX signals with the given action:

UNIX signal	Action	
SIGHUP (01)	Terminate **ICOBOL**	
SIGINT (02)	**ICOBOL** console interrupt	UNIX Intr key (usually Ctrl-Del)
SIGQUIT(03)	Terminate **ICOBOL**	UNIX Quit key (usually Ctrl-\)
SIGPIPE(13)	Terminate **ICOBOL**	
SIGTERM(15)	Terminate **ICOBOL**	
SIGUSR1(16)	Abort the **ICOBOL** program	
SIGPWR (19)	Terminate **ICOBOL**	

The UNIX Intr and Quit keys can be viewed or changed to different values by using the UNIX *stty* command.

I. UNIX Pipe Opens

I.1. Overview

An OPEN INPUT and an OPEN OUTPUT/EXTEND can accept pipelines to and from the UNIX shell when opening sequential files. The format for specifying a pipeline as a filename is:

```
"|>command"      (pipe to a command      - OPEN OUTPUT/EXTEND)
"|<command"      (pipe from a command    - OPEN INPUT)
```

Where

command

Is any valid Bourne shell syntax.

I.2. Rules

(1) If a file is sequentially organized, you can direct OPEN to open a pipe to either standard input or standard output of the command. When the runtime system encounters the pipe command format while a COBOL OPEN statement is executing, the system opens a pipe to command. The shell then interprets command as though you had entered the string sh -c command. Each pipe file requires the creation of an additional process.

(2) If the first two characters of the pipeline are "|>", then the OPEN mode must be OUTPUT or EXTEND and the output written to the COBOL file by the runtime system becomes standard input of command. If the first two characters of the pipeline are "|<", then the OPEN mode must be INPUT and the standard output of command becomes the data read from the COBOL file by the runtime system. If you don't use the correct open mode, the OPEN fails with a FILE STATUS 91.

(3) When you open a pipe to or from a shell command, the shell looks for the command in the PATH variable, not ICCODEPATH.

(4) The DELETE FILE statement has no effect in these cases. When you close a file that has been opened this way, the CLOSE statement halts the writing to the pipe and tells the runtime system to wait for the process on the other end of the pipe to terminate.

(5) This syntax is allowed on the right hand side in the linkfile produced by ICLINK. Thus, the following syntax can be used to open a pipe to the UNIX print spooler.

In the linktextfile:

```
$lpt  |>lp -dprinter1 -onobanner
```

J. BTRIEVE Support (Windows only)

J.1. Overview

On Windows only, the runtime system and ICREORG support BTRIEVE as an alternative to ICISAM for implementing the COBOL INDEXED file organization.

In order to use BTRIEVE, a BTRIEVE option must be added to the **ICOBOL** Runtime License and either a BTRIEVE Requestor and/or a BTRIEVE Client Engine must be running on the workstation to allow the runtime to support BTRIEVE files. In addition, if a BTRIEVE Requestor is used, a BTRIEVE server must be available on the specified server.

If you are running with a Novell network, the BTRIEVE Requestor (TSR) and BTRIEVE Server NLM are provided as part of the Novell release. You must ensure that BTRIEVE support has been enabled on the server and the BTRIEVE Requestor has been loaded on the workstation.

If you wish to run BTRIEVE locally or on another network, you must purchase the appropriate BTRIEVE Client package. This can be done by contacting Pervasive Software (www.pervasive.com) (Austin, TX, USA) at 1-800-287-4383. Customers outside the continental U.S. and Canada can call 512-231-6000 (FAX 512-794-1763).

Other than ICREORG, no other ICISAM utility can be used on BTRIEVE files. Instead the appropriate BTRIEVE utility should be used.

All BTRIEVE support has been directed at the functionality that is available in BTRIEVE version 6.15.

J.2. Runtime

When the runtime starts, if the BTRIEVE option is available from ICPERMIT, the runtime checks to see if a BTRIEVE Requestor and/or Client are available. If not available, a Warning is given and, if available, an Info message is given.

The runtime makes it possible to use both ICISAM and BTRIEVE simultaneously, with ICISAM being the default record manager. The use of BTRIEVE can be requested for a particular file by attaching the extended open option to specify the file brand (",b=b") to the end of the filename. Extended open options are discussed beginning on page 756.

The runtime system can also be made to default to BTRIEVE, thus eliminating the need for the extended open option, by specifying the default file Brand switch (-B b) on the ICRUN starting command line. In this mode, a BTRIEVE Requestor and/or Client must be available or else an error is given and the runtime terminates. The use of ICISAM can be requested for a particular file by attaching the extended open option to specify the file brand (",b=i") to the end of the filename.

J.2.1 Rules

(1) To open an existing BTRIEVE file, its file specifications must match an indexed structure that **ICOBOL** can handle (i.e., you cannot open a file whose number of keys exceed the 17 supported by **ICOBOL**).

(2) The runtime system does not common BTRIEVE file opens. If a main program and a sub-program both have the file open, two opens will be posted to BTRIEVE.

(3) When creating a BTRIEVE file the default page size can be specified by using the extended index open option x, which specifies the data element size in 512-byte blocks. If not specified, a default of 4 (2048-bytes) is used.

(4) The runtime system maps the most commonly occurring BTRIEVE status codes to existing Exception Status and File Status codes (e.g., 9:"The operation encountered an end-of-file mark" maps to EXCEPTION 37, FILE STATUS 10). However, the following Exception Status codes are specific to BTRIEVE:

EXCEPTION STATUS	FILE STATUS	BTRIEVE STATUS	ERROR MESSAGE
166	23	10	Attempt to modify an unmodifiable key.
167	94	80	Attempt to rewrite a record which has been modified since it was read.
168	94	78	Attempt to perform an operation which would lead to a deadlock situation.
416	30	N/A	BTRIEVE initialization failed
417		20	The BTRIEVE Record Manager or Requester is inactive.
418	30	53	The BTRIEVE Record Manager or Requester interface is invalid.
419	30	N/A	BTRIEVE does not implement the required capability.
420	30	33,70,75,89	BTRIEVE returned a reserved status code.
421	30	1,2,6,7,14-17, 19,23-25,30-32, 34,36-45,48-50, 52,54-56,60-69, 71-74,76,79,83, 92,93,98, 102-107	BTRIEVE returned a generic status code.
422	30	N/A	BTRIEVE returned an undefined status code.

TABLE 47. BTRIEVE-specific Exception Status Codes

(5) There are several notable semantic differences in the runtime system between BTRIEVE and ICISAM:

a. UNDELETE file-name RECORD is not supported and generates Exception Status 418.

b. For files in random or dynamic access mode, REWRITE will behave as if the file were in the sequential access mode with regard to the rule that the value of the primary record key of the record to be replaced must be equal to the value of the primary record key of the last record read from the file.

c. When the file has been OPEN'ed for INPUT only, you cannot read records that have been locked on another handle.

d. Filenames are not expanded to their full network name (or UNC name) or truncated to the MS-DOS 8.3 restrictions.

e. BTRIEVE file opens are not commoned. If a main program and a sub-program both have the file open, two opens will be posted to BTRIEVE.

(6) The runtime system uses ICDATAPATH to find BTRIEVE files.

J.3. ICREORG

ICREORG supports BTRIEVE files for both Input and Output by using the "b" file-type.

J.4. ICNETD

ICNETD does NOT support BTRIEVE files.

K. C-ISAM Support (UNIX only)

K.1. Overview

On UNIX only, the runtime system and ICREORG support C-ISAM as an alternative to ICISAM for implementing the COBOL INDEXED file organization.

In order to use C-ISAM, a C-ISAM option must be added to the **ICOBOL** Runtime License and a valid C-ISAM license should be in hand.

If you wish to run C-ISAM, you must purchase the appropriate C-ISAM package. This can be done by contacting INFORMIX Software, Inc. (www.informix.com) (Menlo Park, CA. USA) at 1-415-926-6300.

K.2. Runtime

The runtime makes it possible to use both ICISAM and C-ISAM simultaneously, with ICISAM being the default record manager. The use of C-ISAM can be requested for a particular file by attaching the extended open option to specify the file brand (",b=c") to the end of the filename.

The runtime system can also be made to default to C-ISAM, thus eliminating the need for the extended open option, by specifying the default file Brand switch (-B c) on the ICRUN starting command line. The use of ICISAM can be requested for a particular file by attaching the extended open option to specify the file brand (",b=i") to the end of the filename.

K.2.1 Rules

(1) To open an existing C-ISAM file, its file specifications must match an indexed structure that **ICOBOL** can handle (i.e., you cannot open a file whose number of keys exceed the 17 supported by **ICOBOL**).

(2) There is one notable semantic difference in the runtime system between C-ISAM and ICISAM:

a) UNDELETE file-name RECORD is not supported and generates Exception Status 418.

(3) The runtime system does NOT use ICDATAPATH to find C-ISAM files.

K.3. ICREORG

ICREORG supports C-ISAM files for both Input and Output by using the "c" file-type.

K.4. ICNETD

ICNETD does NOT support C-ISAM files.

L. HOT KEYS

L.1. Introduction

Hot keys are available in Interactive COBOL to allow a specific program (a hotkey program) to be run when a particular key is pressed without changing the currently running program. For example, to provide a pop-up calculator or calendar.

Hot keys are defined in the terminal description file (.tdi). Enter the Configure Keyboard selection under Terminal Descriptions and change the Type for an input key to "Hot Key Function" and the Code to the particular hotkey program to be run. Available hotkey programs are "hotkey00" thru "hotkey99".

Keys that are described as "Hot Key Function" can never be seen by an application. For this reason, although there are no restrictions imposed by Interactive COBOL, printable characters, standard delimiter keys (newline, carriage-return, ESC, Tab (used by Print Utility), F1 - F3, and screen edit keys (Ctrl-A, Ctrl-R, Ctrl-V, etc.) should be avoided. Particular function keys needed by the current COBOL application should also be avoided.

If the particular hotkey program is not available or otherwise gets an error while loading, a beep will be given.

To link a particular program to a hotkey program you can either rename the program to the particular hotkey name, "hotkey00", "hotkey01", etc. or use the linking facility (ICLINK) to provide for runtime linking of the hotkey program name to the actual COBOL program.

L.2. Construction

Hotkey programs are most useful when they use the SCREEN HANDLER functions to save and restore screens, even though hotkey programs do not require the SCREEN HANDLER. Hotkey programs should be designed to detect whether the SCREEN HANDLER is running and perform the appropriate functions.

We suggest that the hotkey program should perform an SD_NEW_WINDOW when it first starts and an SD_REMOVE_WINDOW just before it exits. It can then freely use the screen to interact with the user. When it exits, the screen will be restored. The hotkey program can also be used to perform lookups and return data to the ACCEPT field with the SD_RETURN_INPUT call.

If your application uses an initial program to allow the user to logon via a username and password type scheme, you should make sure that any hotkey program that is installed disallows its use if the user has not properly logged on.

Hotkey programs should also insure that they do not do a STOP RUN or CALL PROGRAM or get a Fatal Error since that will stop the entire run unit. A hotkey program should be written like a CALL subprogram such that it will always return to its calling program.

The builtin functions IC_ENABLE_HOTKEY and IC_DISABLE_HOTKEY provide the ability to selectively allow or disallow access to a hotkey program.

L.3. Restrictions

Hot keys cannot be used while in a builtin or system call.

Hotkey programs must abide by the same subprogram rules (recursion) as normal subprograms.

Hot keys are only recognized during an ACCEPT operation on the current console.

If the hotkey program was started initially by a hot key then it is automatically CANCEL'ed on exit. If the hotkey program was already loaded (via a CALL) it is not CANCEL'ed on exit. I.E., a hotkey program will always start in its initial state if it was not previously loaded with a CALL.

If a hotkey program CALL's a program that was not already loaded it will not be automatically CANCEL'ed when it exits. The hotkey program must explicitly cancel subprograms that it initiates.

Up to 100 unique hotkey programs can be configured, "hotkey00" thru "hotkey99".

There is no mechanism to pass parameters to a hotkey program.

Hotkey programs can be any program (including builtins) that can be CALL'ed from a COBOL program, but not an operating system program although you can build a COBOL stub program that in turn calls an operating system program.

L.3. Example

The program sysserve in the examples directory in the Runtime release is a sample HotKey program that provides a System Services screen. If the SCREEN HANDLER is available it uses it to save the initial screen that will be restored when it exits. If no SCREEN HANDLER is available the screen is blanked when it exits.

XVI. ICODBC Driver

A. Introduction

The Interactive COBOL ODBC Driver (ICODBC32.DLL) for Windows is a 32-bit ODBC Driver that provides an ODBC-compliant interface via icodbc32.dll. It is accessible from 32-bit ODBC-enabled applications.

The Interactive COBOL ODBC Driver (ICODBC.SO) for UNIX is a 32-bit ODBC Driver that provides an ODBC-compliant interface via icodbc.so. This interface can be used with the unixODBC Driver Manager. One program in particular that makes use of this interface is the JDBC-ODBC Bridge under the Java Runtime.

ICODBC is a fully functional ODBC Driver (SQL-92 Entry level compliant) providing access to Interactive COBOL data records stored in INDEXED ORGANIZATION files. Through this mechanism it is possible for ODBC-enabled programs (e.g., Crystal Reports, Visual Basic, PowerBuilder, Microsoft Access, etc...) to use SQL to access legacy Interactive COBOL INDEXED file data records as if they were rows of a table in an SQL relational database.

B. General Information

On Windows, for the purpose of buffering file data within an application, multiple opens of the same local/redirected file are commoned by using the lower case rendition of the file name supplied to the driver. It is important to always specify the identical filename (or alias) and not a different alias to refer to a particular file. This applies to database and table definition files as well as data (INDEXED) file names. On UNIX, the inode number is used so the name is not important for buffering.

The ICODBC Driver optionally connects to the shared memory area created and initialized by the Interactive COBOL System Executive Program (ICEXEC). Multi-user file sharing, buffering, and record locking are handled more efficiently through this mechanism as opposed to a stand-alone (single-user) environment.

C. Using the Driver

In order for the ICODBC driver to provide for the INDEXED to SQL data translation, the application builder must supply information regarding the database, the tables comprising the database, and the rows and columns comprising each table. Simply speaking, an INDEXED file can be viewed as a table (or tables) comprised of a set of rows (records), each one specifying a value for a column (field).

The ICODBC driver utilizes two ASCII text files, which are formatted according to Microsoft Windows initialization (.ini) file conventions, to describe the appropriate view of a database and the tables which it contains. The two files are the .xdb (Database definition) file, which describes a database; and the .xdt (Table Definition) file, which describes a table. **Note that the key names are case-sensitive, and must appear in the file exactly as specified below.**

For a given database, the .xdb file explicitly specifies the number of tables comprising that database, defining the name, the INDEXED file, and the Table definition file for each one. Although there is an obvious relationship between an INDEXED file and a Table Definition file, there is no forced association required by the driver. Thus, it is possible to describe different databases using the same Table Definitions paired with different instances of INDEXED files.

For a table, the .xdt file explicitly specifies the number of columns comprising the table, defining the name, the position, size, and type of the data field corresponding to each. Although there may be obvious relationships between the data fields of a record and the columns of a table, the driver does not enforce a particular correspondence. Thus it is possible to describe different columns of a table using the same data field, or to describe the columns of a table using only some of the data fields available. In many cases it may be necessary to have one column that is the whole row (record), along with individual columns that may or may not duplicate other columns in the row (record).

D. Creating .XDB and XDT Files

The **ICOBOL** compiler (when started with the Make ICODBC Definition Files switch (-M)) can be used to create a preliminary Database Definition file (.xdb) and Table Definition File(s) (.xdt) as it compiles a source program. These preliminary files can provide a starting point for tailoring the definitions of your database and tables. The ICODBC Options switch (-X *string*) on the compiler can be used to set specific ICODBC generation options. See page 707 in the Compiler Chapter for more on the compiler ICODBC Options switch.

For the Database Definition file (.xdb), the compiler will create the [Database] section with the NumTables key and then will generate the [Tables] section and individual [<table-names>] sections based on the number of Indexed files found in the program.

For the Table Definition file(s) (.xdt), the compiler will create the [Table] section with NumColumns, MaxRecordSize, and MinRecordSize keys, the [Primary Key] section, the [Columns] section, and finally the [<column-names>] sections with Type, Position, Length keys along with any other key that is needed for the particular Type, for each of the columns that were detected in the record definition of the COBOL program..

In the descriptions below, note that the characters '[' and ']' are required (they are not part of an optional definition). These definition "Section Names" in the file. Within Sections are "keys" which have a value associated with them. Leading spaces are ignored and blank lines are ignored. **Bold lines are required.**

The Database Definition File (.xdb)

XDB Syntax

```
[Database]
NumTables=<number-of-tables>
OpenMode=<open-mode>
BaseYear=<base-year-value>
BaseYearPivot=<base-year-pivot-value>
EpochYear=<epoch-year-value>
EpochDay=<epoch-day-value>
EpochTick=<epoch-tick-value>
ProxyDate=<date-value>
ProxyTime=<time-value>
ProxyTimestamp=<timestamp-value>

[Tables]
<table-name-1>
...
<table-name-n>

[<table-name-1>]
TableFile=<table-definition-file-name>
DataFile=<data-file-name>
...

[<table-name-n>]
TableFile=<table-definition-file-name>
DataFile=<data-file-name>
```

XDB General Rules

* "NumTables" key is required and the value of <*number-of-tables*> must match the number of table names listed in the [Tables] section and the number of [<*table-name-i*>] sections.

* "OpenMode" key is optional and specifies the open mode for the data files comprising the database. The value of <*open-mode*> must be one of either "INPUT", "OUTPUT", "I-O", or "EXTEND". If this key is not specified, a default value of "INPUT" is implied. If the value of <*open-mode*> is either "OUTPUT", "I-O", or "EXTEND", the "WRITE" option must be present in the license authorizing the driver. This value can be overridden by the presence of an "OpenMode" key in the [Table] section of the individual Table Definition files (see below).

* "BaseYear" key is optional and specifies the century year to be added to the two or three digit year values of the DAY and DATE data types described below. The value of <*base-year-value*> must be a valid numeric edited string literal, and must specify a year greater than or equal to 1600 (up to 32700) that is a century (i.e., divisible by 100). If this key is not specified, a default value of 1900 is implied.

* "BaseYearPivot" key is optional and specifies the two-digit year value of the DAY and DATE data types described below, to which, if less than, a century (i.e., 100 years) will be added, in addition to the value of <*base-year-value*>. The value of <*base-year-pivot-value*> must be a valid numeric edited string literal, and must be greater than 0 and less than 99. If this key is not specified, a default value of 0 is implied.

* "EpochYear" key is optional and specifies the starting year of time (as represented by a zero value) for the corresponding epoch data types described below. The value of <*epoch-year-value*> must be a valid numeric edited string literal, and must specify a year greater than (up to 32767) or equal to 1601. If this key is not specified, a default value of 1601 is implied.

* "EpochDay" key is optional and specifies the starting day of time (as represented by a zero value) for the corresponding epoch data types described below. The value of <*epoch-day-value*> must be a valid numeric edited string literal, and must specify a day greater than or equal to 1 and less than or equal to 365 (or 366 if <*epoch-year-value*> represents a leap year). If this key is not specified, a default value of '01-01' (i.e., January 1) is implied.

* "EpochTick" key is optional and specifies the discrete unit of time that passes between single value increments of epoch data types described below. The value of <*epoch-tick-value*> must be must be one of either "SECOND", "BISECOND", "MINUTE", or "DAY". If this key is not specified, a default value of "SECOND" is implied.

* "ProxyDate" key is optional and specifies a particular value to be substituted for otherwise invalid values when retrieving SQL_DATE data. The value of <*date-value*> must be of the form 'yyyy-mm-dd' (e.g. "0001-01-01").

* "ProxyTime" key is optional and specifies a particular value to be substituted for otherwise invalid values for SQL_TIME data. The value of <*time-value*> must be of the form 'hh:mm:ss' (e.g. "00:00:00").

* "ProxyTimestamp" key is optional and specifies a particular value to be substituted for otherwise invalid values for SQL_TIMESTAMP data. The value of <*timestamp-value*> must be of the form 'yyyy-mm-dd hh:mm:ss.ff' (e.g. "0001-01-01 00:00:00.00").

* There must be an identically named section for each <*table-name-i*> specified in the [Tables] section. At least one Table is required.

* <*table-definition-file-name*> must be a valid pathname specifying a valid Table Definition (.xdt) file, although the ".xdt" extension must not be present. It may be a 'relative' (as opposed to 'absolute') pathname, in which case the pathname specifier for the Database Definition (.xdb) file in the Data Source will be automatically prefixed to it. It may also be a URL specification as documented for ICNETD in the Interactive COBOL Utilities Manual. It may contain a single variable name reference to be substituted with an assigned value when a connection to the database is established. Variable names are delimited by a leading and trailing percent character ('%').

* <*data-file-name*> must be a valid pathname specifying an **ICOBOL** INDEXED file. It may be a 'relative' (as opposed to 'absolute') pathname, in which case the pathname specifier for the Database Definition (.xdb) file in the Data Source will be automatically prefixed to it. It may also be a URL specification as documented for ICNETD in the Interactive COBOL Utilities Manual. It may contain a single variable name reference to be substituted with an assigned value when a connection to the database is established. Variable names are delimited by a leading and trailing percent character ('%').

XDT Syntax

```
[Table]
NumColumns=<number-of-columns>
MaxRecordSize=<maximum-data-record-size>
MinRecordSize=<minimum-data-record-size>
OpenMode=<open-mode>
PrimaryKeyName=<primary-key-name>
NumSelectors=<number-of-selectors>

[Primary Key]
<column-name-p1>
...
<column-name-pN>

[Columns]
<column-name-1>
...
<column-name-n>

[<column-name-1>]
Type=<data-storage-type>
Position=<data-byte-position>
Length=<data-byte-length>
Precision=<data-digits-of-precision>
Scale=<data-digits-of-scale>
Picture=<data-storage-picture>
Suppress=<data-byte-suppress-when-value>
Padding=<data-byte-padding-value>
Default=<data-value>

    ...

[<column-name-n>]
Type=<data-storage-type>
Position=<data-byte-position>
Length=<data-byte-length>
Precision=<data-digits-of-precision>
Scale=<data-digits-of-scale>
Picture=<data-storage-picture>
Suppress=<data-byte-suppress-when-value>
Padding=<data-byte-padding-value>
Default=<data-value>

[Selector]
Type=<data-storage-type>
Position=<data-byte-position>
Length=<data-byte-length>
Precision=<data-digits-of-precision>
Scale=<data-digits-of-scale>
Value=<data-value>
Relation=<data-record-selector-relation>

[Selectors]
<selector-name-1>
...
<selector-name-n>

[<selector-name-1>]
Type=<data-storage-type>
Position=<data-byte-position>
Length=<data-byte-length>
Precision=<data-digits-of-precision>
Scale=<data-digits-of-scale>
Value=<data-value>
Relation=<data-record-selector-relation>

    ...
```

[<*selector-name-n*>]
Type=<*data-storage-type*>
Position=<*data-byte-position*>
Length=<*data-byte-length*>
Precision=<*data-digits-of-precision*>
Scale=<*data-digits-of-scale*>
Value=<*data-value*>
Relation=<*data-record-selector-relation*>

[Foreign Keys]
<*foreign-key-table-name-f1*>=<*foreign-key-name-1*>
...
<*foreign-key-table-name-fN*>=<*foreign-key-name-n*>

[<*foreign-key-name-j*>]
<*column-name-fj0*>
...
<*column-name-fjN*>

General Rules

* "NumColumns" key is required and specifies the number of columns in the table. The value of <*number-of-columns*> must match the number of column names listed in the [Columns] section and the number of [<*column-name-i*>] sections.

* <*maximum-data-record-size*> and <*minimum-data- record-size*> values must respectively match the actual maximum and minimum record sizes of the INDEXED file; and they must be the same if the records are fixed-length.

* "OpenMode" key is optional and specifies the open mode for the data file of the table. The value of <*open-mode*> must be one of either "INPUT", "OUTPUT", "I-O", or "EXTEND". If this key is not specified, the value specified by the "OpenMode" key in the [Database] section of the Database Definition file is implied.

* "PrimaryKeyName" key is optional and specifies the primary key for the purposes of foreign key reference.

* "NumSelectors" key is optional and specifies the number of record selectors for the table. A record selector specifies a subset of the records in the INDEXED file which are to be considered as rows in the table. The value of <*number-of-selectors*> must match the number of selector names listed in the [Selectors] section and the number of [<*selector-name-i*>] sections.

* [Primary Key] section is optional and specifies the column(s) of the table which comprise the primary key. This section must be specified, and all of the columns specified must be either "ALPHABETIC" or "ALPHANUMERIC", in order for the Microsoft Jet Database Engine (see above) to be able to create a dynaset over rows of the table. In general specifying this column will allow faster access to the data.

* [Foreign Keys] section is optional and identifies the tables whose primary keys are referenced by foreign keys from the table.

* There must be an identically named <*table-name-j*> key in the [Tables] section of the Database Definition (.xdb) file be for each <*foreign-key-table-name-i*>.

* There must be an identically named section in the Table Definition (.xdt) file for each <*foreign-key-name-i*>. These sections serve to identify the columns of the table which comprise the foreign key.

* [Selector] section is optional and specifies the simple definition of only one record selector for the table. It may not be present when the "NumSelectors" key is present.

* There must be an identically named section for each <*column- name-i*> specified in the [Columns] section.

* "Type" key is required for the [Selector], [<*selector-name-i*>], or the [<*column-name-i*>] section if specified, and specifies the data storage type of the item.

* <*data-storage-type*> value must be one of either "BYTE", "ALPHABETIC", "ALPHANUMERIC", "DISPLAY", "TRAILING OVERPUNCH", "TRAILING SEPARATE", "LEADING OVERPUNCH", "LEADING SEPARATE", "UNSIGNED DISPLAY", "COMP", "UNSIGNED COMP", "COMP-3", "UNSIGNED COMP-3", "COMP-5", "UNSIGNED COMP-5", "DAY", "COMP DAY", "DATE", "COMP DATE", "COMP DATE GROUP", "TIME", "COMP TIME", "COMP TIME GROUP", "FULLDATE", "EPOCH TIMESTAMP", or "COMP EPOCH TIMESTAMP", matching the **ICOBOL** data type of the corresponding field.

* "Value" key is required for the [Selector] or [<*selector-name-i*>] section if specified, and specifies the value to be used in determining the desired subset.

* <*data-value*> value may be any character string literal (unquoted) if the <*data-storage-type*> of the item is "ALPHABETIC" or "ALPHANUMERIC". Otherwise, if the <*data-storage-type*> of the item is "BYTE", the value of <*data-value*> must be a valid hexadecimal string literal (unquoted). Otherwise, if the <*data-storage-type*> of the item is "DISPLAY", "TRAILING OVERPUNCH", "TRAILING SEPARATE", "LEADING OVERPUNCH", "LEADING SEPARATE", "UNSIGNED DISPLAY", "COMP", "UNSIGNED COMP", "COMP-3", "UNSIGNED COMP-3", "COMP-5", or "UNSIGNED COMP-5", the value of <*data-value*> must be a valid numeric edited string literal. Otherwise, if the <*data-storage-type*> of the item is "DAY", "COMP DAY", "DATE", "COMP DATE", or "COMP DATE GROUP", the value of <*data-value*> must be a character string literal (unquoted) of the form 'yyyy-mm-dd'. Otherwise, if the <*data-storage-type*> of the item is "TIME", "COMP TIME", or "COMP TIME GROUP", the value of <*data-value*> must be a character string literal (unquoted) of the form 'hh:mm:ss'. Otherwise, if the <*data-storage-type*> of the item is "FULLDATE", "EPOCH TIMESTAMP", or "COMP EPOCH TIMESTAMP", the value of <*data-value*> must be a character string literal (unquoted) of the form 'yyyy-mm-dd hh:mm:ss.ff'.

* "Relation" key is optional for the [Selector] or [<*selector-name-i*>] section if specified, and specifies the a comparison operation to be applied in determining the desired subset. The value of <*data-record-selector-relation*> must be one of either "EQ", "NE", "GT", "GE", "LT", or "LE" and specifies the relationship between the value of the selector in a record and <data-record-selector-value> which must be true for a record to be included in the desired subset. If this key is not specified, a default value of "EQ" is implied (i.e., records for which the value of their selector field is equal to the value of <*data-record-selector-value*> are included in the subset).

* "Position" key is required for the [Selector], [<*selector-name-i*>], or the [<*column-name-i*>] section if specified, and specifies the data byte position of the item.

* <*data-byte-position*> value must be the byte position (one-based) within a record to the data field corresponding to the column.

* "Length" key is required for either the [Selector] or the [<*column-name-i*>] section if specified, and specifies the data byte length of the item.

* <*data-byte-length*> value must be the length in bytes within a record of the data field corresponding to the column.

* For a section where the value of <*data-storage-type*> is "BYTE", "ALPHABETIC" or "ALPHANUMERIC", neither the "Precision" key nor the "Scale" key may be present.

* For a section where the value of <*data-storage-type*> is not "BYTE", "ALPHABETIC" or "ALPHANUMERIC", the "Precision" key must be present.

* For a section where the value of <*data-storage-type*> is "DAY", "COMP DAY", "DATE", "COMP DATE", "COMP DATE GROUP", "TIME", "COMP TIME", "COMP TIME GROUP", "FULLDATE", "EPOCH TIMESTAMP", or "COMP EPOCH TIMESTAMP" the "Scale" key must not be present.

* For a section where the value of <*data-storage-type*> is "COMP DATE GROUP" or "COMP TIME GROUP", the value of <*data-digits-of-precision*> must be the total number of decimal digits in all the elementary items of the group.

* <*data-digits-of-precision*> value must be the total number of decimal digits to the left and right of the decimal point defined for the data field corresponding to the column. (e.g., if the field is defined as "PIC 9999V99", the value is 6).

* <*data-digits-of-scale*> value must be the number of decimal digits to the right of the decimal point defined for the data field corresponding to the column. (e.g., if the field is defined as "PIC 9999V99", the value is 2).

* "Picture" key is optional and may be present only for a section where the value of <*data-storage-type*> is "DATE", "COMP DATE" or "COMP DATE GROUP". The value of <*data-storage-picture*> must be one of either "YYYYMMDD", "YYYYDDMM", "MMDDYYYY", "MMYYYYDD", "DDMMYYYY", "DDYYYYMM", "CCYYMMDD", "CCYYDDMM", "MMDDCCYY", "MMCCYYDD", "DDMMCCYY", or "DDCCYYMM" if the value of <*data-digits-of-precision*> is 8; it must be one of either "YYYMMDD", "YYYDDMM", "MMDDYYY", "MMYYYDD", "DDMMYYY", or "DDYYYMM" if the value is 7; and it must be one of either "YYMMDD", "YYDDMM", "MMDDYY", "MMYYDD", "DDMMYY", or "DDYYMM" if the value is 6.

* "Suppress" key is optional; but if present, the value of <*data-byte-suppress-when-value*> must be the numeric value (0 to 255) which when present in all bytes of the data field indicates that the value of the column is considered empty or null.

* "Padding" key is optional; but if present, the value of <*data-byte-padding-value*> must be the numeric value (0 to 255) which will be used to pad the value of the column to its full length of <*data-byte-length*> when a shorter value is specified.

* For a section where the value of <*data-storage-type*> is not "ALPHABETIC" or "ALPHANUMERIC", the "Padding" key must not be present.

* "Default" key is optional and specifies the default value to be stored for the column if no value is provided as part of the INSERT statement.

  NOTE: If you plan on adding records (rows) to a database then consider setting this value especially for DATE / TIME columns as on an INSERT sometimes a "null" record is inserted and then the values that the user had specified are individually PUT into the columns, thus the INSERT would fail with an Invalid data-type-value for a DATE column if the "Default" is NOT specified.

Example XDB

```
[Database]
NumTables=4
OpenMode=I-O
; AOS/VS uses biseconds since 1968-01-01 00:00:00
;    EpochYear=1968
;    EpochTick=BISECOND
; UNIX uses seconds since 1970-01-01 00:00:00
;    EpochYear=1970
;    EpochTick=SECOND
; MacOS uses seconds since 1903-01-01 00:00:00
;    EpochYear=1904
;    EpochTick=SECOND
; CBS uses days since 1876-12-31 00:00:00
;    EpochYear=1876
;    EpochDay=366
;    EpochTick=DAY

[Tables]
Customers
```

```
Companies
Orders
Products

[Customers]
TableFile=c:\application\odbcdesc\anycust
DataFile=c:\application\livedata\%this%cust

[Companies]
TableFile=c:\application\odbcdesc\anycomp
DataFile=c:\application\livedata\%this%comps

[Orders]
TableFile=c:\application\odbcdesc\anyorder
DataFile=c:\application\livedata\%this%order

[Products]
TableFile=c:\application\odbcdesc\product
DataFile=c:\application\livedata\products
```

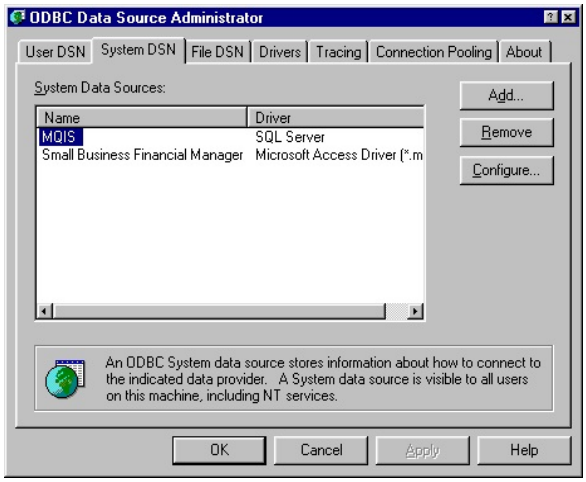
Example XDT

```
[Table]
NumColumns=6
MinRecordSize=100
MaxRecordSize=100
PrimaryKeyName=CustomerKey

[Columns]
CustomerId
Company
Address
City
State
ZipCode

[Primary Key]
CustomerId

[Foreign Keys]
Companies=CompanyKey

[CompanyKey]
Company

[CustomerId]
Type=UNSIGNED DISPLAY
Position=1
Length=10
Precision=10
Scale=0

[Company]
Type=ALPHANUMERIC
Position=11
Length=20

[Address]
Type=ALPHANUMERIC
Position=31
Length=40

[City]
Type=ALPHANUMERIC
Position=71
Length=20

[State]
Type=ALPHABETIC
Position=91
Length=2

[ZipCode]
```

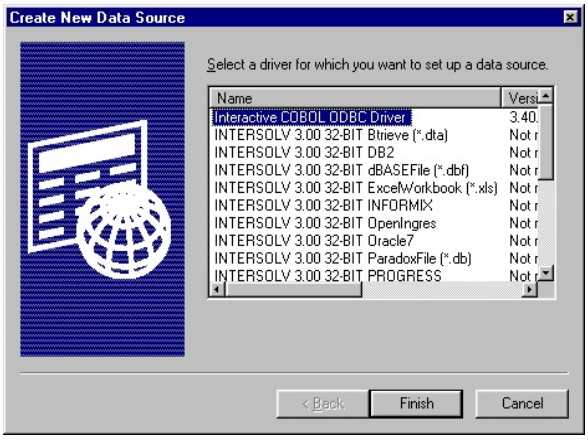
```
Type=UNSIGNED DISPLAY
Position=93
Length=5
Precision=5
Scale=0
```

E. Managing Data Sources (On Windows)

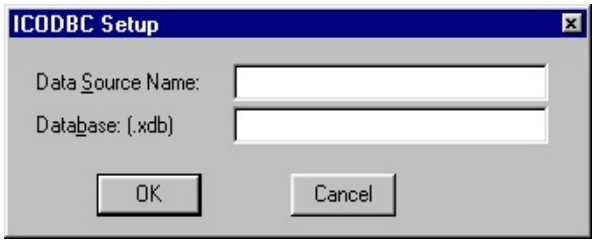
1. Run the 32-bit ODBC Administrator. Typically, you should be able to do this by double-clicking the 32-bit ODBC Administrator icon in the Control Panel, or by selecting the 32-bit ODBC Administrator from the programs folder created by the driver installation procedure.



2. To ADD (i.e., create) a new data source from any of the DSN panels (User, System, or File) in the ODBC Data Source Administrator dialog box, click the "Add" button. The dialog box below will be shown.



From this Create New Data Source dialog box, select the "Interactive COBOL ODBC Driver" and click the "Finish" button. The Setup dialog below will be shown.



From the ICODBC Setup dialog box you may specify the name of the data source and enter the name of the database to be associated with it. The Database field must contain the absolute (i.e., fully qualified) pathname of a Database definition (.xdb) file WITHOUT the ".xdb" extension. It may be a URL specification as documented for the Interactive COBOL Runtime System. Additionally, it may contain a comma-list of value assignments to be used to dynamically substitute for variable names referenced in table definition or data file names when establishing a connection to the database. Click OK. The ODBC Data Source Administrator dialog box will be reactivated. See below for a description of the Database definition (.xdb) file.

3. To DELETE (i.e., remove) a data source from the Data Sources dialog box, select the one you want to eliminate and click the "Delete" button.

4. To CHANGE the setup for a data source from the Data Sources dialog box, select the one you want to eliminate and click the "Configure" button. From the ICODBC Setup dialog box you may now change the name of the data source and/or the name of the database to be associated with it. See the description above for adding a new data source.

5. Click Close.

Example

Data Source name:	ABC Company Database
Database: (.xdb)	c:\application\odbcdesc\anycust,this=ABC

From your favorite ODBC-enabled program there will typically be a SQL/ODBC option available when you select the data to access. Choosing this option will allow you to connect to any one of the data sources available to you. For details, please read the documentation and/or online Help for the particular program you are using.

F. Managing Data Sources (On UNIX)

The ICODBC Driver on Unix is intended to provide an ODBC interface to those UNIX applications that can make use of a shared object interface to ODBC. One of those type applications is the unixODBC project. unixODBC is available from www.unixodbc.org and must be installed before using the ICODBC Unix driver (icodbc.so).

UnixODBC is not so much an end user program, but rather an intermediary between a program and one or more databases. In this case the database is ICISAM files.

The isql program that comes with unixODBC can be used to perform simple connections and queries to test that ICODBC is installed correctly.

Getting Started with unixODBC

UnixODBC is available in source code only. This means that you download a tar file (or zipped tar file) from http://www.unixODBC.org, extract it, compile it, and install it.

Installing unixODBC

As mentioned before, get the source tar file from http://www.unixodbc.org. As root, move the tar file to /op or /usr/local or where ever you want the source to reside. If zipped, then unzip the file. Untar the file and run the following commands from the command line in the unixODBC source directory:

```
./configure
make
make install
```

Assuming that you have all the libraries and tools that it needs, you should be breezing through this compile. UnixODBC takes quite a while to compile, actually all of these packages do. Relax and enjoy it. After installing you will probably have to set the path for shared objects (LD_LIBRARY_PATH on Linux).

Installing

Prerequisets:

On UNIX, there is no ODBC Administrator so the data-sources must be configured in a text file.

System versus User

ODBC distingushes between two types of ini files. System ini files are designed to be accessable but not modifable by any user, and user files are private to a particular user, and may be modified by that user. The system files are odbcinst.ini and odbc.ini (note no leading dot), and the user file is ~/.odbc.ini in each user's home directory (note the leading dot).

The system file odbcinst.ini contains information about ODBC drivers available to all users, and the odbc.ini file contains information about DSN's available to all users. These "System DSN's" are useful for application such as web servers that may not be running as a real user and so will not have a home directory to contain a .odbc.ini file.

A good example of this is Apache and PHP with ODBC support. When the http server is first started it calls SQLAllocEnv as root. It then at a later time changes to the specified user (in my case nobody) and calls SQLConnect. If the DSN's was not a system DSN then this fails.

The ~/.odbc.ini in the user's home directory are "User DSN's". These are only useful for cases of testing or when you do not need to share datasets.

The unixODBC library uses the odbcinst.ini file to administrator the driver manager. Again this file is in a .ini format and has the following format.

odbcinst.ini

This ini file simply lists all installed drivers. It is located in /usr/local/etc/odbcinst.ini. The syntax is simple; a name followed by a property which tells us the drivers file name.

For example;

```
[Sybase 11]
Comment = Super Duper Sybase Server
Driver =  /usr/lib/libsybase.so
Setup = /usr/lib/libsybaseS.so
FileUsage = 1
```

The Driver file name (i.e., /usr/lib/libsybase.so) should be unique. The friendly name (i.e., Sybase 11) must also be unique.

The Setup property points to a shared object containing functions to be called by ODBC Config. ODBC Config will call this share to get driver specific property names during data source configuration. If ODBC Config can not find/use this file it will assume some defaults such as; Data Source Name, Host, and default Database. (Setup is NOT SUPPORTED BY ICODBC at this time.)

One can modify this file either using the ODBCINST shared object, by using the command line equivalent odbcinst, or a standard UNIX editor.

The odbcinst command can be used to add ICODBC to this file.

Enter the following into a temp file:

```
[ICODBC]
Comment   = Interactive COBOL ISAM ODBC Driver for Linux
Driver    = /usr/lib/icodbc.so
FileUsage = 1
```

Now invoke odbcinst with the following arguments assuming you have created a file template_icodbc:

```
odbcinst -i -d -f template_icodbc
```

The args to odbcinst are as follows:

-i (install)
-d (driver name)
-f (name of template)

Make sure you copy or link the released file, icodbc.so, to icodbc.so in /usr/lib. If you had specified a simple name in the Driver line above, then the path for shared objects can be used to find the icodbc driver. (LD_LIBRARY_PATH under Linux.) You can use the installic script in the examples sub-directory of the icobol release to install icodbc.so.

Just execute:

```
examples/installic icodbc
```

If you wish to turn on ODBC tracing then the following needs to be added to the odbcinst.ini file:

```
[ODBC]
Trace = Yes
Trace File = filename
```

If not specified, Trace defaults to NO and Trace File defaults to /tmp/sql.log.

odbc.ini or ~/.odbc.ini

These files describe the data-set to be used. They have the same format but refer to SystemDSN and UserDSN's respectively.

The environment variable ODBCSYSINI can be used to find the system odbc.ini file and the environment variable HOME is used to find the user .odbc.ini file. If the system file is not found then the "$HOME"/.odbc.ini file is tried. If it is not found then the unixODBC will fail on the DriverConnect. Thus you must have ODBCSYSINI set if you are not using UserDSN's.

The contents of the odbc.ini files give a section that is the data-set name, then a description, driver, and the DBQ (database) entry. Generally each driver requires different entries. The entries may be added in the same way using odbcinst, or a text editor. A sample entry to match the above driver could be:

```
[TESTDSN]
Description         = Test IC Dataset
Driver              = ICODBC
DBQ                 = /home/data/datfile89
UID                 = user-id
PWD                 = user-password
Threading           = 3
```

And this may be written to a template file, and inserted in the ini file for the current user by:

```
odbcinst -i -s -f template_file
```

The individual entries of course may vary.

The Driver line is used to match the [section] entry in the odbcinst.ini file and the Driver line in the odbcinst file and is used to find the path for the driver library, and this is loaded and the connection is then established. It's possible to replace the driver entry with a path to the driver itself. This can be used, for example if the user can't get root access to setup anything in /etc (less important now because of the movable etc path).

The DBQ line specifies the actual ICISAM database file to open. UID/PWD specify the user-id and password if any required to access the database.

The Threading line instructs unixODBC to not allow any threading. This should be the default.

Currently the icodbc.so driver has auditing in effect. An audit log "icodbc_<pid>.lg" will be created in the current directory for all connections.

The isql command that comes with unixODBC can be used to connect to a dataset and execute some simple SQL commands.

Java

One application that makes use of the unixODBC project is the JDBC-ODBC Bridge that is provided with the Java 2 Runtime environment. A java runtime can be downloaded from www.java.sun.com/products. The JDBC-ODBC Bridge enable java programs to access ODBC data when a JDBC compliant interface is not available to access the same data.

Under Java, the JDBC-ODBC Driver can be loaded with the ClassforName("sun.jdbc.odbc.JdbcodbcDriver")

A Connection to an ICOBOL Isam database can be made via:

```
   Connect con = DriverManager.getConnection( jdbc.odbc.<datasource>,
<username>, <userpassword> );
```

where <datasource> is the DataSetName for the ICOBOL database.

ERROR CONDITIONS

A. You get:

```
    Error: Connection refused (oserr=111) Connecting to localhost:7334
java.sql.SQLException:[unixODBC]
```

Icpermit is not running.

B. You get:

```
    java.sql.SQLException: No suitable driver
```

The unixODBC driver(s) cannot be found. Make sure the load path for .so files is set. (Under Linux, LD_LIBRARY_PATH=/usr/local/lib).

C. You get:

```
java.sql.SQLException: [unixODBC]
```

The icodbc.so driver cannot find "$ODBCSYSINI"/odbc.ini or "$HOME"/.odbc.ini to find the data set name.

You must have an ICODBC license available from the license manager.

G. Data Types Supported

Currently the ICODBC driver provides for the following mapping of ICODBC data types to ODBC SQL data types as shown in the following table. Examples of **ICOBOL** data types are also shown.

ICODBC Type	SQL Type	Length	Precision	Scale	**ICOBOL** Data Description
ALPHABETIC	SQL_CHAR	*n*	n/a	n/a	PIC A(*n*)
ALPHANUMERIC	SQL_CHAR	*n*	n/a	n/a	PIC X(*n*)
		varies	n/a	n/a	group item
		varies	n/a	n/a	alphanumeric edited items
		varies	n/a	n/a	numeric edited items
BYTE	SQL_BINARY	*n*	n/a	n/a	PIC X(*n*) or group used in a key/key-segment with subordinated items of non-DISPLAY usage
UNSIGNED DISPLAY	SQL_NUMERIC	*l*+*r*	*l*+*r*	*r*	PIC 9(*l*)V9(*r*) USAGE DISPLAY
DISPLAY	SQL_NUMERIC	*l*+*r*	*l*+*r*	*r*	PIC S9(*l*)V9(*r*) USAGE DISPLAY
LEADING OVERPUNCH	SQL_NUMERIC	*l*+*r*	*l*+*r*	*r*	PIC S9(*l*)V9(*r*) USAGE DISPLAY SIGN LEADING
LEADING SEPARATE	SQL_NUMERIC	*l*+*r*+1	*l*+*r*	*r*	PIC S9(*l*)V9(*r*) USAGE DISPLAY SIGN LEADING SEPARATE
TRAILING OVERPUNCH	SQL_NUMERIC	*l*+*r*	*l*+*r*	*r*	PIC S9(*l*)V9(*r*) USAGE DISPLAY SIGN TRAILING
TRAILING SEPARATE	SQL_NUMERIC	*l*+*r*+1	*l*+*r*	*r*	PIC S9(*l*)V9(*r*) USAGE DISPLAY SIGN TRAILING SEPARATE
UNSIGNED COMP	SQL_NUMERIC	varies	*l*+*r*	*r*	PIC 9(*l*)V9(*r*) USAGE COMP
		varies	*l*+*r*	*r*	PIC 9(*l*)V9(*r*) USAGE BINARY
UNSIGNED COMP-3	SQL_NUMERIC	varies	*l*+*r*	*r*	PIC 9(*l*)V9(*r*) USAGE COMP-3
		varies	*l*+*r*	*r*	PIC 9(*l*)V9(*r*) USAGE PACKED
UNSIGNED COMP-5	SQL_NUMERIC	varies	*l*+*r*	*r*	PIC 9(*l*)V9(*r*) USAGE COMP-5
COMP	SQL_NUMERIC	varies	*l*+*r*	*r*	PIC S9(*l*)V9(*r*) USAGE COMP
		varies	*l*+*r*	*r*	PIC S9(*l*)V9(*r*) USAGE BINARY
		4	9	0	USAGE INDEX
COMP-3	SQL_NUMERIC	varies	*l*+*r*	*r*	PIC S9(*l*)V9(*r*) USAGE COMP-3
		varies	*l*+*r*	*r*	PIC S9(*l*)V9(*r*) USAGE PACKED
COMP-5	SQL_NUMERIC	varies	*l*+*r*	*r*	PIC S9(*l*)V9(*r*) USAGE COMP-5
		4	10	0	USAGE POINTER
DAY	SQL_DATE	n	n	n/a	PIC 9(n) where n=5,7 ([YY]YYdd)
COMP DAY	SQL_DATE	varies	n	n/a	PIC 9(n) COMP where n=5,7 ([YY]YYddd)
DATE	SQL_DATE	n	n	n/a	PIC 9(n) where n=6,7,8 ([[Y]Y]YYMMDD)
COMP DATE	SQL_DATE	varies	n	n/a	PIC 9(n) COMP where n=6,7,8 ([[Y]Y]YYMMDD)
COMP DATE GROUP	SQL_DATE	varies	n=6 or 8	n/a	see note aa
TIME	SQL_TIME	n	n	n/a	PIC 9(n) where n=4,6,8 (hhmm[ss[ff]])
COMP TIME	SQL_TIME	varies	n	n/a	PIC 9(n) COMP where n=4,6,8 (hhmm[ss[ff]])
COMP TIME GROUP	SQL_TIME	2,3,4	n=4,6,or 8	n/a	see note bb
EPOCH TIMESTAMP	SQL_TIMESTAMP	varies	n!= 0	n/a	PIC [S]9(n)
COMP EPOCH TIMESTAMP	SQL_TIMESTAMP	varies	n!=0	n/a	PIC [S]9(n) COMP
FULLDATE	SQL_TIMESTAMP	20	n/a	n/a	PIC X(20) where n=20 (YYYYdddMMDDhhmmssffw)

TABLE 48. ICODBC Data Types to ODBC SQL Data Types

Note aa:

```
01 DATE-GROUP.
    02 YY PIC 9(2) COMP
    02 MM PIC 9(2) COMP
    02 DD PIC 9(2) COMP
        n=6 (default Picture=YYMMDD)
01 DATE-GROUP.
    02 YYYY PIC 9(4) COMP
    02 MM PIC 9(2) COMP
    02 DD PIC 9(2) COMP
```

```
            n=8 (default Picture=YYYYMMDD)
        01 DATE-GROUP.
            02 CC PIC 9(2) COMP
            02 YY PIC 9(2) COMP
            02 MM PIC 9(2) COMP
            02 DD PIC 9(2) COMP
                n=8 (CCYYMMDD)
```

Note bb:

```
        01 TIME-GROUP.
            02 HH PIC 9(2) COMP
            02 MM PIC 9(2) COMP
        01 TIME-GROUP.
            02 HH PIC 9(2) COMP
            02 MM PIC 9(2) COMP
            02 SS PIC 9(2) COMP
        01 TIME-GROUP.
            02 HH PIC 9(2) COMP
            02 MM PIC 9(2) COMP
            02 SS PIC 9(2) COMP
            02 FF PIC 9(2) COMP
```

It is intended that eventually other ODBC SQL data types will be supported through either implicit or explicit column descriptions in the table definition file. Suggestions are welcome.

H. Driver Limitations

* Entry level SQL-92 compliant, with some additional Intermediate and/or Full level functionality. See the SQL grammar section at the end of this file for the SQL grammar supported. Some modification statements (CREATE or DROP) are not supported semantically, although they are supported syntactically.

* SQLBrowseConnect, SQLTablePrivileges, SQLColumnPrivileges, SQLProcedures, and SQLProcedureColumns are not supported. These ODBC API calls are not SQL-92 compliant CLI Calls and are not commonly used.

* Character and binary values supplied for parameterized queries (SELECT * FROM EMPLOYEE WHERE NAME = ?) are limited to 255 bytes.

* Interval types are not supported.

* Qualifiers or owners are not allowed on databases, tables, etc.

* Transactions are not supported.

* Only SQL_CHAR, SQL_NUMERIC, SQL_BINARY, SQL_TIME, SQL_DATE, and SQL_TIMESTAMP are supported.

* Queries that specify columns which are components of an INDEXED key are satisfied based on the internal ordering of the key, which may not be equivalent to the external ordering.

* The following are the (maximum) limits of various implementation defined elements:

Character Literal Length	255	Binary Literal Length	255
Database Name Length	27	Column Name Length	63
Index Name Length	63	Table Name Length	63
Key Name Length	63	User Name Length	63
Password Length	63	Number of Columns in Order By	20
Number of Columns in a Key	15	Number of Columns in Index	15
Number of Foreign Keys in a Table	15		

I. SQL Grammar Supported

```
statement       ::= CREATE create | DROP drop | SELECT select orderby | INSERT insert | DELETE delete |
                    UPDATE update

create          ::= TABLE tablename ( createcols ) | INDEX indexname ON tablename ( indexcolumns )

indexcolumns    ::= indexcolumn | indexcolumn , indexcolumns

indexcolumn     ::= columnname asc

createcols      ::= createcol , createcols | createcol

createcol       ::= columnname datatype | columnname datatype ( integer ) |
                    columnname datatype ( integer , integer )

drop            ::= TABLE tablename | INDEX indexname

select          ::= selectcols FROM tablelist where groupby having

delete          ::= FROM tablename where

insert          ::= INTO tablename insertvals

update          ::= tablename SET setlist where

setlist         ::= set | setlist , set

set             ::= columnname = NULL | columnname = expression

insertvals      ::= ( columnlist ) VALUES ( valuelist ) | VALUES ( valuelist ) |
                    ( columnlist ) VALUES (SELECT select) | VALUES ( SELECT select )

columnlist      ::= columname , columnlist | columname

valuelist       ::= NULL , valuelist | expression , valuelist | expression | NULL

selectcols      ::= selectallcols * | selectallcols selectlist

selectallcols   ::= | ALL | DISTINCT

selectlist      ::= selectlistitem , selectlist | selectlistitem

selectlistitem  ::= expression | expression aliasname | expression AS aliasname | aliasname.*

where           ::= | WHERE boolean

having          ::= | HAVING boolean

boolean         ::= and | and OR boolean

and             ::= not | not AND and

not             ::= comparison | NOT comparison

comparison      ::= ( boolean ) | colref IS NULL | colref IS NOT NULL | expression LIKE pattern |
                    expression NOT LIKE pattern | expression IN ( valuelist ) | expression NOT IN ( valuelist ) |
                    expression op expression | EXISTS (SELECT select) |
                    expression op selectop ( SELECT select ) | expression IN ( SELECT select ) |
```

expression NOT IN (SELECT select) expression BETWEEN expression AND expression
expression NOT BETWEEN expression AND expression

selectop ::= | ALL | ANY

op ::= > | >= | < | <= | = | <>

pattern ::= string | ? | USER

expression ::= expression + times | expression - times | times

times ::= times * neg | times / neg | neg

neg ::= term | + term | - term

term ::= (expression) | colref | simpleterm | aggterm | scalar

scalar ::= scalarescape | scalarshorthand

scalarescape ::= --*(VENDOR(MICROSOFT),PRODUCT(ODBC) FN fn)*--

scalarshorthand ::= { FN fn }

fn ::= functionname (valuelist) | functionname () | POSITION (expression IN expression) |
EXTRACT (expression FROM expression)

aggterm ::= COUNT (*) | AVG (expression) | MAX (expression) | MIN (expression) |
SUM (expression) | COUNT (expression)

simpleterm ::= string | realnumber | ? | USER | date | time | timestamp

groupby ::= | GROUP BY groupbyterms

groupbyterms ::= colref | colref , groupbyterms

orderby ::= | ORDER BY orderbyterms

orderbyterms ::= orderbyterm | orderbyterm , orderbyterms

orderbyterm ::= colref asc | integer asc

asc ::= | ASC | DESC

colref ::= aliasname . columnname | columnname

tablelist ::= tablelistitem , tablelist | tablelistitem

tablelistitem ::= tableref | outerjoin

outerjoin ::= ojescape | ojshorthand

ojescape ::= --*(VENDOR(MICROSOFT),PRODUCT(ODBC) OJ oj)*--

ojshorthand ::= { OJ oj }

oj ::= tableref LEFT OUTER JOIN tableref ON boolean |
tableref LEFT OUTER JOIN oj ON boolean | tableref INNER JOIN tableref ON boolean |
tableref INNER JOIN oj ON boolean

tableref ::= tablename | tablename aliasname

indexname ::= identifier

functionname ::= identifier

tablename ::= identifier

datatype ::= identifier

columnname ::= identifier

aliasname ::= identifier

identifier ::= an identifier (identifiers containing spaces must be enclosed in double quotes)

string ::= a string (enclosed in single quotes)

realnumber ::= a non-negative real number (including E notation)

integer ::= a non-negative integer

date ::= dateescape | dateshorthand

dateescape ::= --*(VENDOR(MICROSOFT),PRODUCT(ODBC) d dateval)*--

dateshorthand ::= { d dateval }

dateval ::= a date in yyyy-mm-dd format in single quotes (for example, '1996-02-05')

time ::= timeescape | timeshorthand

timeescape ::= --*(VENDOR(MICROSOFT),PRODUCT(ODBC) t timeval)*--

timeshorthand ::= { t timeval }

timeval ::= a time in hh:mm:ss format in single quotes (for example, '10:19:48')

timestamp ::= timestampescape | timestampshorthand

timestampescape ::= --*(*VENDOR(MICROSOFT),PRODUCT(ODBC) ts timestampval)*--

timestampshorthand ::= { ts timestampval }

timestampval ::= a timestamp in yyyy-mm-dd hh:mm:ss[.ffffff] format in single quotes (for example, '1996-02-05 10:19:48.529')

J. Usage Notes

Jet Database Engine

The Microsoft Jet Database Engine ("Jet") is the advanced relational database engine built into Microsoft Access(R) and Visual Basic(R). Jet is intended to provide transparent access to data, regardless of the data's location and format, and therefore deals with ODBC data. Any error returned by Jet that falls in the range -7700 to -7799 is an ODBC Specification Compliance Error. The error indicates that an ODBC driver has failed to comply with the

ODBC specification and represents a bug in the driver. Please report all such errors to us with as much detail as possible.

In Microsoft Access, links to tables in an ODBC data source can be created; these links are called attached tables. Attaching ODBC tables allows you to use them transparently within Microsoft Access, but to implement this transparency, Jet must ask the ODBC driver for a great deal of information about the table and cache it locally. This process can be expensive and complex. After establishing a connection to the desired data source, Jet calls the ODBC API function SQLTables to obtain a list of tables (and other similar objects) in the ODBC data source. When you select a table, Jet calls SQLColumns, SQLStatistics, SQLSpecialColumns, and various ODBC Info functions to acquire information about the selected table.

To allow updating of attached ODBC tables, Jet creates dynasets over them. There must be a unique (primary key) value for each row in the table and it must be of type character and treatable as a null-terminated string. The unique key values of a row are also called the row's bookmark because they uniquely identify and allow direct access to the row.

Because data sources vary in their use of binary data, sometimes data loss can occur. Character data is generally considered to be in the form of a null-terminated string, so values may lose some accuracy when being transferred to Jet. If this data forms part of a table's bookmark, Jet might think the row has been deleted ("#Deleted" will appear in a Microsoft Access datasheet/form). This is because Jet asked for the row by its key values, but no exact match was found. Jet cannot distinguish this situation from that of a genuine record deletion by another user.

Microsoft Access and Visual Basic

Errors can occur while an application is running, either from the Visual Basic environment or as a stand-alone executable. In particular, Visual Basic reserves a portion of the first 1000 error numbers, and other error numbers are reserved by the Microsoft Jet Database Engine, or are available for defining custom errors. More information concerning these errors is available in online help facility in the "Trappable Errors" and "Trappable Data Access Errors" topics from Microsoft Access Help Topics.

Microsoft Access 2000

Microsoft Access 2000 appears to be stricter than earlier versions of that product regarding its adherence to the declared precision of numeric fields. The ODBC Driver responds to a request for information concerning a column of a table by Microsoft Access with, among other things, the display size, scale, and precision of the column. For the precision, the driver dutifully returns the value of <data-digits-of-precision> specified in the column definition of the table definition file. Because it is possible with Interactive COBOL COMPUTATIONAL data types to store a value whose actual precision exceeds the declared precision of a data item, strict enforcement of the declared precision has, in some cases, led to a problem whereby Microsoft Access 2000 reports that "The decimal field's precision is too small to accept the numeric you attempted to add". Further complicating this situation, the **ICOBOL** compiler sets the value of <data-digits-of-precision> as determined from the picture describing the data item in the COBOL program.

Depending on the intent of the application, there are a number of ways to correct this problem. If the value of the field is meant to be limited by the declared precision then the application should assure that no data is stored with a greater precision; and steps should be taken to eliminate the corrupt data from the data file. The application can be compiled with the "-G p" to enable size checking based on the picture; or it can perform data entry validation. If the value of the field can legitimately have more digits of precision than declared then the declared precision of the item in the COBOL source program of the application should be increased to properly reflect that fact. The value of <data-digits-of-precision> in the column definition then needs to be modified, either automatically with the compiler or manually with a text editor.

It is also possible to coerce the generated value of <data-digits-of-precision> to be the maximum for COMPUTATIONAL items through the use of the ICODBC option "-G p" command line option. Please reference page 189, 192, 226 for information regarding the storage of COMPUTATIONAL data items.

K. Debugging

Several points for starting to use the ICODBC Driver.

1. Start in read-only mode to lessen the likely hood of corrupting your data.

2. Start with just a few columns and get it working.

3. Enable SQL tracing and see what is passed into the SQL call and what the actual SQL calls return by looking at the output from the trace. This may give you a hint as to the real problem. **NOTE:** Remember to turn off tracing when it is no longer needed!!

4. Under Access start with an import and not a link.

5. Look in the readodbc.txt file for more debugging information.

L. SYWARE

The ICODBC Driver was built using the Microsoft Visual C++ 6.0 compiler with the Microsoft ODBC Software Development Kit (SDK) version 3.51 and the SYWARE Dr. DeeBee ODBC Driver Kit (Gold Edition). Portions of the product are copyrighted by SYWARE Inc. and Microsoft.

XVII. ICIDE

A. Introduction

The **ICOBOL I**ntegrated **D**evelopment **E**nvironment (ICIDE) is available with **ICOBOL** on Windows. The ICIDE provides a GUI interface to define the project, run the compiler, perform queries, tailor reports, browse sources and reports, and edit sources. To use the ICIDE, an ICOBOL Development license must be available from the license manager.

The ICIDE provides a project-based framework for editing, managing, and compiling the ICOBOL source files that make up your application. A project encompasses a set of source files plus the associated COPY files, compiler settings, and directory information necessary to organize and build your application. You determine what set of files to include in a project. You can, for example, create a project that contains all of the source files for your "Acme Accounting System." Or you can create a project that contains a subset of a larger system's source files; for an "Accounts Payable Subsystem," for example.

Editing a file is as simple as double clicking a filename. Compiling a file (or a group of files) takes just a keystroke. Compiling all of a project's files takes just a few mouse clicks. If an error occurs during compilation, the offending source line can be automatically displayed in an editing window with the click of a mouse.

Once you've defined a project and performed the initial build, the ICIDE opens up a whole new set of tools for traversing your application. As an integral part of the build process, the ICIDE creates a symbol file for each program and it integrates information from each of those symbol files into a global symbol table. This allows the ICIDE to efficiently perform operations such as a global cross-reference of a symbol. A simple right click operation on a symbol allows you to see all the places where that symbol is used in the entire application.

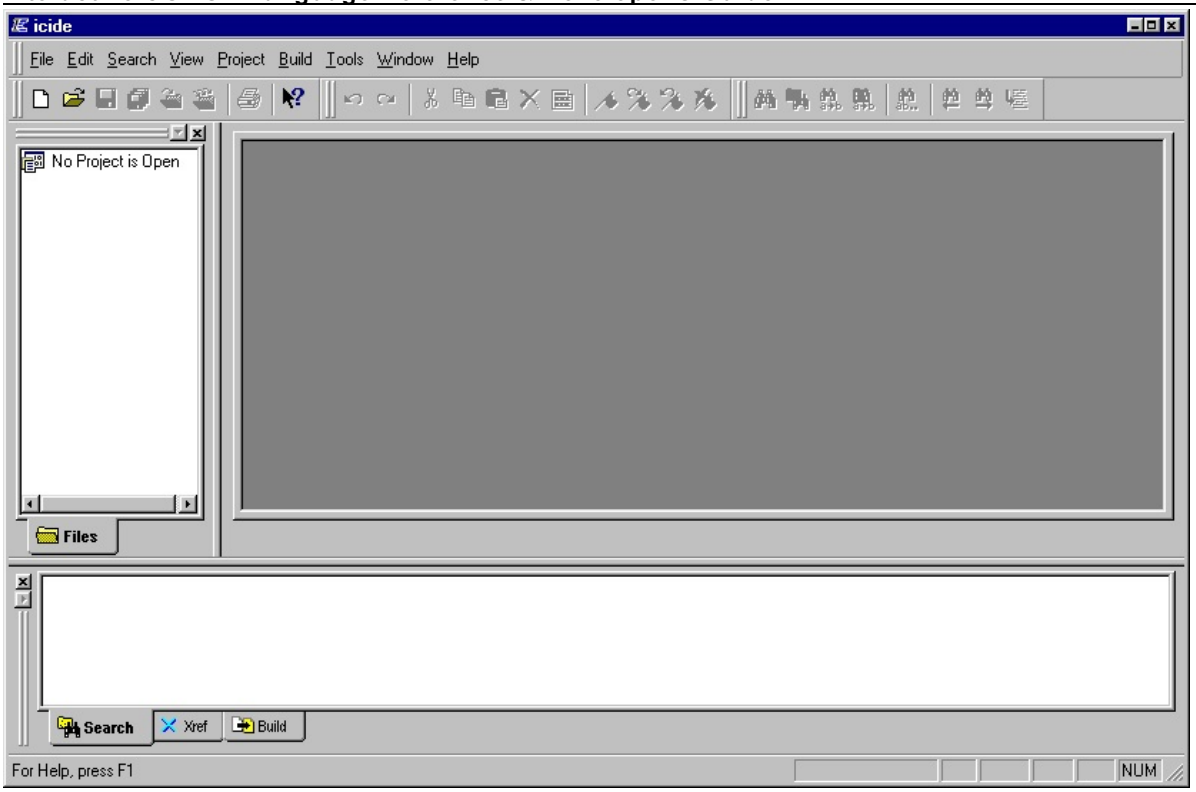
B. Use

When the ICIDE is first started a screen like below will be shown. To the left is the Project Window (where "No Project is Open" is shown. To the right (in the grey area) is the Source Window. At the bottom is the Output Window used for Search, Cross Reference, and Building.

The syntax for starting the ICIDE from the command line is:

`icide`	Start icide
`icide filename`	Start icide with filename opened
`icide /p filename`	Print filename to the default printer
`icide /pt filename printer driver port`	Print filename to the specified printer

To get started, Select **Help**, **Welcome** and then go through the **Start Here** section for a short tutorial about the ICIDE. After that you can use the **Command Reference** section to get help on different subjects as needed.



C. Changing .CO or .SR file associations

To Change the .co or .sr file association from card to free (or vice versa) when using the ICIDE.

For Windows 9x/NT

Use regedit:
- Position to HKEY_CLASSES_ROOT and open
- Position to .co
- Select (Default)
- Now select Edit and Modify
- Change the Value data from "ICIDE.CardSource" to "ICIDE.FreeSource"
- Click OK
- Exit the Registry

Now if you double-click on a *.co file it will be brought up as Free-Form format and not Card format.

Just do the opposite to change a .sr into a CardFormat.

For Windows 2000

Select My Computer
- Select Tools
- Select Folder Options
- Select File Types
- Pick .co
- Select New
- enter co extension

Click Advanced
Now go to Associated File Type and pull-down
Change to ICIDE.Free-Format
Say Yes to move association

Now if you double-click on a *.co file it will be brought up as Free-Form format and not Card format.

Just do the opposite to change a .sr into a CardFormat.

XVIII. GLOSSARY

A. Introduction

The terms in this section are defined in accordance with their meaning in COBOL, and may not have the same meaning for other languages.

These definitions are also intended as either reference or introductory material to be reviewed prior to reading the detailed language specifications that follow. For this reason, these definitions are, in most instances, brief and do not include detailed syntactical rules.

B. Definitions

Abbreviated Combined Relation Condition. The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

Access Mode. The manner in which records are to be operated upon within a file.

Actual Decimal Point. The physical representation, using the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

Alphabet-Name. A user-defined word, in the SPECIAL-NAMES paragraph of the Environment Division, that assigns a name to a specific character set and/or collating sequence.

Alphabetic Character. A letter or a space character.

Alphanumeric Character. Any character in the computer's character set.

Alternate Record Key. A key, other than the primary record key, whose contents identify a record within an indexed file.

Arithmetic Expression. An identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

Arithmetic Operation. The process caused by the execution of an arithmetic statement, or the evaluation of an arithmetic expression, that results in a mathematically correct solution to the arguments presented.

Arithmetic Operator. A single character or fixed two-character combination which belongs to the following set:

Character	Meaning
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Arithmetic Statement. A statement that causes an arithmetic operation to be executed. The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements.

Ascending Key. A key upon the values of which data is ordered starting with the lowest value of key up to the highest value of key in accordance with the rules for comparing data items.

ASCII character set. The 96-character ASCII character set is composed of the 96 characters from space (decimal 32) through DEL (decimal 127).

Assumed Decimal Point. A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning with no physical representation.

At End Condition. A condition caused during the execution of 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.

Block. A physical unit of data that is normally composed of one or more logical records. For disk files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block. The term is synonymous with physical record.

Called Program. A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit or a program which is the object of a CALL PROGRAM statement which produces a new run unit.

Calling Program. A program which executes a CALL or CALL PROGRAM to another program.

Character. The basic indivisible unit of the language.

Character Position. A character position is the amount of physical storage required to store a single standard data format character whose usage is DISPLAY.

Character-String. A sequence of contiguous characters which form a COBOL word, a literal, a PICTURE character-string, or a comment-entry.

Class Condition. The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or is wholly numeric or consists exclusively of those characters listed in the definition of a class-name.

Class-Name. A user-defined word defined in the SPECIAL-NAMES paragraph of the Environment Division that assigns a name to the proposition for which a truth value can be defined, that the content of a data item consists exclusively of those characters listed in the definition of the class-name. **NOT SUPPORTED BY *VXCOBOL***.

Clause. A clause is an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

COBOL Character Set. The complete COBOL character set consists of the characters listed below.

Character	Meaning
0, 1, ... , 9	digit
A, B, ... , Z	uppercase letter
a, b, ... , z	lowercase letter
	space
+	plus sign
-	minus Sign (hyphen)
*	asterisk
/	slant (solidus)
=	equal sign
$	currency sign (represented as # in the International Reference Version of International Standard ISO 646-1973)
,	comma (decimal point)
;	semicolon
.	period (decimal point, full stop)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon

Note: Except when used in nonnumeric literals and some PICTURE symbols, each lowercase letter is equivalent to the corresponding uppercase letter.

COBOL Word. A character-string of not more than 30 characters which forms a user-defined word, a system-name, or a reserved word.

Collating Sequence. The sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting, merging, comparing, and for processing indexed files sequentially.

Column. A character position within a print or display line. The columns are numbered from 1, by 1, starting at the left-most character position of the print line and extending to the right-most position of the line.

Combined Condition. A condition that is the result of connecting two or more conditions with the `AND' or the `OR' logical operator.

Comment-Entry. An entry in the Identification Division that may be any combination of characters from the computer's character set.

Comment Line. A source program line represented by an asterisk (*) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a slant (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line causes page ejection prior to printing the comment.

Compile Time. The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

Compiler Directing Statement. A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation. The compiler directing statement is the COPY statement.

Complex condition. A condition in which one or more logical operators act upon one or more conditions.

Computer-Name. A system-name that identifies the computer upon which the program is to be compiled or run.

Computer's character set. The computer's character set for all computers on which **ICOBOL** is currently supported is the complete 8-bit ASCII table from decimal 0-255; i.e., the entire range of 8-bit (1-byte) values.

Condition. A status of a program at execution time for which a truth value can be determined. Where the term `condition' (*condition-1*, *condition-2*, ... ) appears in these language specifications in or in reference to `condition' (*condition-1*, *condition-2*, ...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a single combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

Condition-Name. A user-defined word that assigns a name to a subset of values that a conditional variable may assume; or a user-defined word assigned to a status of a switch or device. When `condition-name' is used in the general formats, it represents a unique data item reference consisting of a syntactically correct combination of a condition-name, together with qualifiers and subscripts, as required for uniqueness of reference.

Condition-Name Condition. The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

Conditional Expression. A simple condition or a complex condition specified in an EVALUATE, IF, PERFORM or SEARCH statement.

Conditional Phrase. A conditional phrase specifies the action to be taken upon determination of the truth value of a condition resulting from the execution of a conditional statement.

Conditional statement. A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

Conditional Variable. A data item one or more values of which has a condition-name assigned to it.

Configuration Section. A section of the Environment Division that describes overall specifications of source and object programs.

Contiguous Items. Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchical relationship to each other.

Coordinated Universal Time (UTC). Time scale, based on the second (SI), as defined and recommended by the CCIR, and maintained by the Bureau International des Poids et Mesures (BIPM). For most practical purposes associated with the Radio Regulations, UTC is equivalent to mean solar time at the prime meridian (0°° longitude), formerly expressed in GMT. [NTIA] [RR] **Note 1**: The maintenance by BIPM includes cooperation among various national laboratories around the world. **Note 2**: The full definition of UTC is contained in CCIR Recommendation 460-4. (188) **Note 3** : The second was formerly defined in terms of astronomical phenomena. When this practice was abandoned in order to take advantage of atomic resonance phenomena ("atomic time") to define the second more precisely, it became necessary to make occasional adjustments in the "atomic" time scale to coordinate it with the workaday mean solar time scale, UT-1, which is based on the somewhat irregular rotation of the Earth. Rotational irregularities usually result in a net decrease in the Earth's average rotational velocity, and ensuing lags of UT-1 with respect to UTC. **Note 4**: Adjustments to the atomic, i.e., UTC, time scale consist of an occasional addition or deletion of one full second, which is called a leap second. Twice yearly, during the last minute of the day of June 30 and December 31, Universal Time, adjustments may be made to ensure that the accumulated difference between UTC and UT-1 will not exceed 0.9 s before the next scheduled adjustment. Historically, adjustments, when necessary, have usually consisted of adding an extra second to the UTC time scale in order to allow the rotation of the Earth to "catch up." Therefore, the last minute of the UTC time scale, on the day when an adjustment is made, will have 59 or 61 seconds. Synonyms World Time, Z Time, Zulu Time. (Source: www.its.bldrdoc.gov)

Counter. A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

CRC. See Cyclic Redundancy Check.

Currency Sign. The character `$' of the COBOL character set.

Currency Symbol. The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign.

Current Record. In file processing, the record which is available in the record area associated with a file.

Current Volume Pointer. A conceptual entity that points to the current volume of a sequential file.

Cyclic Redundancy Check. A sophisticated checksum, which is based on the algebra of polynomials over the integers (mod 2). It is substantially more reliable in detecting transmission errors, and is one common error-checking protocol used in modems. (Source: http://mathworld.wolfram.com/CyclicRedundancyCheck.html)

Data Clause. A clause, appearing in a data description entry in the Data, Division of a COBOL program, that provides information describing a particular attribute of a data item.

Data Description Entry. An entry, in the Data Division of a COBOL program, that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

Data Item. A unit of data (excluding literals) defined by the COBOL program.

Data-Name. A user-defined word that names a data item described in a data description entry. When used in the general formats, `data-name' represents a word which must not be reference-modified, subscripted, or qualified unless specifically permitted by the rules of the format.

Debugging Line. A debugging line is any line with a `d' or `D' in the indicator area of the line.

Declarative Sentence. A compiler directing sentence consisting of a single USE statement terminated by the separator period.

Declaratives. A set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the keyword DECLARATIVES and the last of which is followed by the keywords END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one, or more associated paragraphs.

De-Edit. The logical removal of all editing characters from a numeric edited data item in order to determine that item's unedited numeric value.

Delimited Scope Statement. Any statement which includes its explicit scope terminator.

Delimiter. A character or a sequence of contiguous characters that identify the end of a string of characters and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

Descending Key. A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

Destination. The symbolic identification of the receiver of a transmission from a queue.

Digit Position. A digit position is the amount of physical storage required to store a single digit. This amount may vary depending on the usage specified in the data description entry that defines the data item. If the data description entry specifies that usage is DISPLAY, then a digit position is synonymous with a character position.

Division. A collection of zero, one, or more sections or paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. Each division consists of the division header and the related division body. There are four divisions in a COBOL program: Identification, Environment, Data, and Procedure.

Division Header. A combination of words, followed by a separator period, that indicates the beginning of a division. The division headers in a COBOL program are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION [ USING { data-name-1 }... ] .
```

Dynamic Access. An access mode in which specific logical records can be obtained from or placed into a disk file in a non-sequential manner and obtained from a file in a sequential manner during the scope of the same OPEN statement.

Editing Character. A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
z	zero suppress
*	check protect
$	currency sign
.	(decimal point) period
/	slant (solidus)

Elementary Item. A data item that is described as not being further logically subdivided.

End of Procedure Division. The physical position of a COBOL source program after which no further procedures appear.

Entry. Any descriptive set of consecutive clauses terminated by a separator period and written in the Identification Division, Environment Division, or Data Division of a COBOL program.

Environment Clause. A clause that appears as part of an Environment Division entry.

Execution Time. The time at which an object program is executed. The term is synonymous with object time.

Explicit Scope Terminator. A reserved word which terminates the scope of a particular Procedure Division statement.

Expression. An arithmetic or conditional expression.

Extend Mode. The state of a file after execution of an OPEN statement, with the EXTEND phrase specified, for that file and before the execution of a CLOSE statement.

External Data. The data that is described in a program as external data items and external file connectors.

External Data Item. A data item which is described as part of an external record of a run unit and which itself may be referenced from any program in which it is described.

External Data Record. A logical record which is described in one or more programs of a run unit and whose constituent data items may be referenced from any program in which they are described.

External File Connector. A file connector which is accessible to one or more object programs in the run unit.

External Switch. A software device that can be specified when invoking a run unit. It is defined in the SPECIAL-NAMES paragraph and can be used to indicate that one of two alternate states exists (ON or OFF).

Field. A contiguous row of character positions on a display screen. These characters form a logical unit that can be filled with data, moved, displayed, etc.

Figurative Constant. A compiler generated value referenced through the use of certain reserved words.

File. A collection of logical records.

File Attribute Conflict Condition. An unsuccessful attempt has been made to execute an input-output operation on a file and the file attributes, as specified for that file in the program, do not match the fixed attributes for that file.

File Clause. A clause that appears as part of any of the following Data Division entries: file description entry (FD entry) and sort-merge file description entry (SD entry.)

File Connector. A storage area which contains information about a file and is used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

FILE-CONTROL. The name of an Environment Division paragraph, in which the data files are declared for a given source program.

File Control Entry. A SELECT clause and all its subordinate clauses which declare the relevant physical attributes of a file.

File Description Entry. An entry in the File Section of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

File-Name. A user-defined word that names a file connector that is described in a file description entry or a sort-merge file description entry within the File Section of the Data Division.

File Organization. The permanent logical file structure established at the time that a file is created.

File Position Indicator. A conceptual entity that contains the value of the current key within the key of reference for an indexed file, or the record number of the current record for a sequential file, or the relative record number of the current record for a relative file, or indicates that no next logical record exists, or that the number of significant digits in the relative record number is larger than the size of the relative key data item, or that an optional input file is not present, or that the at end condition already exists, or that no valid next record has been established.

File Section. The section of the Data Division that contains file description entries and sort-merge file description entries together with their associated record descriptions.

File System. An input-output control system that directs, or controls, the processing of mass storage files.

Fixed File Attributes. Information about a file which is established when a file is created and cannot subsequently be changed during the existence of the file. These attributes include the organization of the file (sequential, relative, or indexed), the primary record key, the alternate record keys, the code set, the minimum and maximum record size, the record type (fixed or variable), the collating sequence of the keys for indexed files, the blocking factor, the padding character, and the record delimiter.

Fixed Length Record. A record associated with a file whose file description or sort-merge description entry requires that all records contain the same number of character positions.

Format. A specific arrangement of a set of data.

Global Name. A name which is declared in only one program but which may be referenced from that program and from any program contained within that program. Condition-names, data-names, file-names, record-names, and some special registers may be global names. **NOT SUPPORTED BY ICOBOL.**

Group Item. A data item that is composed of subordinate data items.

High Order End. The left-most character of a string of characters.

I-O-CONTROL. The name of an Environment Division paragraph in which object program requirements for rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

I-O-CONTROL Entry. An entry in the I-O-CONTROL paragraph of the Environment Division which contains clauses which provide information required for the transmission and handling of data on named files during the execution of a program.

I-O Mode. The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement.

I-O Status. A conceptual entity which contains the two-character value indicating the resulting status of an input-output operation. This value is made available to the program through the use of the FILE STATUS clause in the file control entry for the file.

Identifier. A syntactically correct combination of a data-name, with its qualifiers, subscripts, and reference modifiers, as required for uniqueness of reference, that names a data item. The rules for `identifier' associated with the general formats may, however, specifically prohibit qualification, subscripting, or reference modification.

Imperative Statement. A statement that either begins with an imperative verb and specifies an unconditional action to be taken or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement may consist of a sequence of imperative statements.

Implicit Scope Terminator. A separator period which terminates the scope of any preceding unterminated statement, or a phrase of a statement which by its occurrence indicates the end of the scope of any statement contained within the preceding phrase.

Index. A computer storage area or register, the content of which represents the identification of a particular element in a table.

Index Data Item. A data item in which the values associated with an index-name can be stored.

Index-Name. A user-defined word that names an index associated with a specific table.

Indexed File. A file with indexed organization.

Indexed Organization. The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

Initial Program. A program that is placed into an initial state every time the program is called in a run unit.

Initial State. The state of a program when it is first called in a run unit.

Input File. A file that is opened in the input mode.

Input Mode. The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement.

Input-Output File. A file that is opened in the I-O mode.

Input-Output Section. The section of the Environment Division that names the files and the external media required by an object program and which provides information required for transmission and handling of data during execution of the object program.

Input-Output Statement. A statement that causes files to be processed by performing operations upon individual records or upon the file as a unit. The input-output statements are: ACCEPT (with the identifier phrase), CLOSE, DELETE, DISPLAY, OPEN, PURGE, READ, REWRITE, SET (with the TO ON or TO OFF phrase), START, UNDELETE, UNLOCK, and WRITE.

Input Procedure. A set of statements, to which control is given during the execution of a SORT statement, for the purpose of controlling the release of specified records to be sorted.

Integer. A numeric literal or a numeric data item that does not include any digit position to the right of the assumed decimal point. When the term `integer' appears in general formats, integer must not be a numeric data item, and must not be signed, nor zero unless explicitly allowed by the rules of that format.

Internal Data. The data that is described in a program excluding all external data items and external file connectors. Items described in the Linkage Section of a program are treated as internal data.

Internal Data Item. A data item which is described in one program in a run unit.

Internal File Connector. A file connector which is accessible to only one object program in the run unit.

Intra-Record Data Structure. The entire collection of groups and elementary data items from a logical record which is defined by a contiguous subset of the data description entries which describe that record. These data description entries include all entries whose level-number is greater than the level-number of the first data description entry describing the intra-record data structure.

Invalid Key Condition. A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

ISAM. Indexed Sequential Access Method. The term ISAM file commonly refers to a relative or indexed file.

Key. A data item which identifies the location of a record, or a set of data items which serve to identify the ordering of data.

Key of Reference. The key, either primary or alternate, currently being used to access records within an indexed file.

Key Word. A reserved word whose presence is required when the format in which the word appears is used in a source program.

Language-Name. A system-name that specifies a particular programming language.

Letter. A character belonging to one of the following two sets:

(1) uppercase letters: 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;
(2) lowercase letters: 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.

Level Indicator. Two alphabetic characters that identify a specific type of file or a position in a hierarchy. The level indicators in the Data Division are: FD and SD.

Level-Number. A user-defined word, expressed as a one or two digit number, which indicates the hierarchical position of a data item or the special properties of a data description entry. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77, and 88 identify special properties of a data description entry.

Line Number. An integer that denotes the vertical position of a report line on a page.

Line Terminator. The line terminator for data-sensitive files for a particular operating system.

Linkage Section. The section in the Data Division of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and the called program.

Literal. A character-string whose value is implied by the ordered set of characters comprising the string.

Logical Operator. One of the reserved words AND, OR, or NOT. In the formation of a condition, either AND, or OR, or both, can be used as logical connectives. NOT can be used for logical negation.

Logical Record. The most inclusive data item. The level-number for a record is 01. A record may be either an elementary item or a group item. The term is synonymous with record.

Longitudinal Redundancy Check The LRC algorithm is an extremely simple error detection method which yields any character from 00h (0) through 0FFh (255). LRC stands for Longitudinal Redundancy Check, an old method based on longitudinal parity. There are two major disadvantages of this method. The first problem is that the resulting check character may be a control character which could interfere with data communications. The second problem is that it is not highly effective, particularly with long transmissions. (Source: http://www.smartronics.com/ref/checksum.html)

Low Order End. The right-most character of a string of characters.

LRC. See Longitudinal Redundancy Check

Mass Storage. A storage medium in which data may be organized and maintained in both a sequential and non-sequential manner.

Mass Storage Control System (MSCS). An input-output control system that directs, or controls, the processing of mass storage files. Generally referred to as the file system.

Mass Storage File. A collection of records that is assigned to a mass storage medium.

Merge File. A collection of records to be merged by a MERGE statement. The file is created and can be used only by the merge function.

Mnemonic-Name. A user-defined word that is associated in the Environment Division with a specific switch name.

Native Character Set. The character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

Native Collating Sequence. The collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

Negated Combined Condition. The `NOT' logical operator immediately followed by a parenthesize combined condition.

Negated Simple Condition. The `NOT' logical operator immediately followed by a simple condition.

Next Executable Sentence. The next sentence to which control will be transferred after execution of the current statement is complete.

Next Executable Statement. The next statement to which control will be transferred after execution of the current statement is complete.

Next Record. The record which logically follows the current record of a file.

Noncontiguous item. Elementary data items, in the Working-Storage and Linkage Sections, which bear no hierarchic relationship to other data items.

Nonnumeric Item. A data item whose description permits its content to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

Nonnumeric Literal. A literal bounded by quotation marks. The string of characters may include any character in the computer's character set.

Numeric Character. A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Numeric Item. A data item whose description restricts its content to a value represented by characters chosen from the digits `0' through `9'; if signed, the item may also contain a `+', `-' or other representation of an operational sign.

Numeric Literal. A literal composed of one or more numeric characters that may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the right-most character. The algebraic sign, if present, must be the left-most character.

OBJECT-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the object program is executed, is described.

Object Computer Entry. An entry in the OBJECT-COMPUTER paragraph of the Environment Division which contains clauses which describe the computer environment in which the object program is to be executed.

Object of Entry. A set of operands and reserved words, within a Data Division entry of a COBOL program, that immediately follows the subject of the entry.

Object Program. A set or group of executable machine language instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program. Where there is no danger of ambiguity, the word `program' alone may be used in place of the phrase `object program'.

Object Time. The time at which an object program is executed. The term is synonymous with execution time.

Obsolete Element. A COBOL language element in Standard COBOL that is to be deleted from the next revision of Standard COBOL.

Open Mode. The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O, or EXTEND.

Operand. Whereas the general definition of operand is `that component which is operated upon', for the purposes of this document, any lowercase word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

Operational Sign. An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Optional File. A file which is declared as being not necessarily present each time the object program is executed. The object program causes an interrogation for the presence or absence of the file.

Optional Word. A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Ordinal Number. A number that show the order or succession in which names, objects, periods of time, or the like, are considered; as, first, second, third, fourth, and so on.

Output File. A file that is opened in either the output mode or extend mode.

Output Mode. The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before the execution of a CLOSE statement for that file.

Output Procedure. A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function reaches a point at which it can select the next record in merged order when requested.

Padding Character. An alphanumeric character used to fill the unused character positions in a physical record.

Paragraph. In the Procedure Division, a paragraph-name followed by a separator period and by zero, one, or more sentences. In the Identification and Environment Divisions, a paragraph header followed by zero, one, or more entries.

Paragraph Header. A reserved word, followed by the separator period, that indicates the beginning of a paragraph in the Identification and Environment Divisions. The permissible paragraph headers in the Identification Division are:

```
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

The permissible paragraph headers in the Environment Division are:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.
```

Paragraph-Name. A user-defined word that identifies and begins a paragraph in the Procedure Division.

Pathname. A file-name that represents the unique path through the file system to a specific file.

Phrase. A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

Physical Record. The term is synonymous with block.

Primary Record Key. A key whose contents uniquely identify a record within an indexed file.

Procedure. A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

Procedure Branching Statement. A statement that causes the explicit transfer of control to a statement other than the next executable statement in the sequence in which the statements are written in the source program. The procedure branching statements are: CALL, EXIT, EXIT PROGRAM, GO TO, MERGE (with the OUTPUT PROCEDURE phrase), PERFORM and SORT (with the INPUT PROCEDURE or OUTPUT PROCEDURE phrase).

Procedure-Name. A user-defined word which is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name (which may be qualified), or a section-name.

Program Identification Entry. An entry in the PROGRAM-ID paragraph of the Identification Division which contains clauses that specify the program-name and assign selected program attributes to the program.

Program-Name. In the Identification Division, a user-defined word that identifies a COBOL source program.

Pseudo-Text. A sequence of text words, comment lines, or the separator space in a source program or COBOL library bounded by, but not including, pseudo-text delimiters.

Pseudo-Text delimiter. Two contiguous equal sign (=) characters used to delimit pseudo-text.

Punctuation Character. A character that belongs to the following set:

Character	Meaning
,	comma
;	semicolon
:	colon
.	period (full stop)
"	quotation mark
(	left parenthesis
)	right parenthesis
	space
=	equal sign

Qualified Data-Name. An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

Qualifier.

(1) A data-name or a name associated with a level indicator which is used in a reference either together with another data-name which is the name of an item that is subordinate to the qualifier or together with a condition-name.

(2) A section-name which is used in a reference together with a paragraph-name specified in that section.

Random Access. An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from, or placed into a relative or indexed file.

Record. The most inclusive data item. The level-number for a record is 01. A record may be either an elementary item or a group item. The term is synonymous with logical record.

Record Area. A storage area allocated for the purpose of processing the record described in a record description entry in the File Section of the Data Division. In the File Section, the current number of character positions in the record area is determined by the explicit or implicit RECORD clause.

Record Description. The total set of data description entries associated with a particular record. The term is synonymous with record description entry.

Record Description Entry. The total set of data description entries associated with a particular record. The term is synonymous with record description.

Record Key. A key whose contents identify a record within an indexed file. Within an indexed file, a record key is either the primary record key or an alternate record key.

Record-Name. A user-defined word that names a record described in a record description entry ln the Data Division of a COBOL program.

Record Number. The ordinal number of a record in the file whose organization is sequential.

Reference Format. A format that provides a standard method for describing COBOL source programs.

Reference Modifier. The left-most-character-position and length used to establish and reference a data item.

Relation. The term is synonymous with relational operator.

Relation Character. A character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

Relation Condition. The proposition, for which a truth value can be determined, that the value of an arithmetic expression, data item, nonnumeric literal, or index-name has a specific relationship to the value of another arithmetic expression, data item, nonnumeric literal, or index-name.

Relational Operator. A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meanings are:

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than OR not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than OR not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to OR not equal to
IS [NOT] GREATER THAN OR EQUAL TO IS [NOT] >=	Greater than or equal to OR not greater than or equal to
IS [NOT] LESS THAN OR EQUAL TO IS [NOT] <=	Less than or equal to OR not less than or equal to

Relative File. A file with relative organization.

Relative Key. A key whose contents identify a logical record in a relative file.

Relative Organization. The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

Relative Record Number. The ordinal number of a record in a file whose organization is relative. This number is treated as a numeric literal which is an integer.

Reserved Word. A COBOL word specified in the list of words which may be used in a COBOL source program, but which must not appear in the program as user-defined words or system-names.

Resource. A facility or service, controlled by the operating system, that may be used by an executing program.

Resultant Identifier. A user-defined data item that is to contain the result of an arithmetic operation.

Routine-Name. A user-defined word that identifies a procedure written in a language other than COBOL.

Run Unit. One or more object programs which interact with one another and which function, at object time, as an entity to provide problem solutions.

Screen-name. A data name that identifies an item in the Screen Section of the Data Division.

Screen Section. The section of the Data Division where items to be used in screen ACCEPTs and DISPLAYs are defined.

Section. A set of zero, one, or more paragraphs or entries, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

Section Header. A combination of words followed by a separator period that indicates the beginning of a section in the Environment, Data, and Procedure Division. In the Environment and Data Divisions, a section header is composed of reserved words followed by a separator period. The permissible section headers in the Environment Division are:

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

The permissible section headers in the Data Division are:

```
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
SCREEN SECTION.
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a separator period.

Section-Name. A user-defined word which names a section in the Procedure Division.

Sentence. A sequence of one or more statements, the last of which is terminated by a separator period.

Separately Compiled Program. A program which, together with its contained programs, is compiled separately from all other programs.

Separator. A character or two contiguous characters used to delimit character-strings.

Sequential Access. An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

Sequential File. A file with sequential organization.

Sequential Organization. The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file

Sign Condition. The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

Significand. In floating-point representation, the fixed-point numeral that represents the significant digits of the number.

Simple Condition. Any single condition chosen from the set:

```
relation condition
class condition
condition-name condition
switch-status condition
sign condition
(simple-condition)
```

Sort File. A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

Sort-Merge File Description Entry. An entry in the File Section of the Data Division that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

Source. The symbolic identification of the originator of a transmission to a queue.

SOURCE-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

Source Computer Entry. An entry in the SOURCE-COMPUTER paragraph of the Environment Division which contains clauses which describe the computer environment in which the source program is to be compiled.

Source Item. An identifier designated by a SOURCE clause that provides the value of a printable item.

Source Program. Although it is recognized that a source program may be represented by other forms and symbols, in this document it always refers to a syntactically correct set of COBOL statements. A COBOL source program commences with the Identification Division or a COPY statement. A COBOL source program is terminated by the absence of additional source program lines.

Special Character. A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign
*	asterisk
/	slant (solidus)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point, full stop)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon

Special Character Word. A reserved word which is an arithmetic operator or a relation character.

SPECIAL-NAMES. The name of an Environment Division paragraph in which **ICOBOL**-specific names (switch name) are related to user-specified mnemonic-names.

Special Names Entry. An entry in the SPECIAL-NAMES paragraph of the Environment Division which provides means for specifying the currency sign; choosing the decimal point; specifying symbolic characters; relating switch-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating sequences; and relating class-names to sets of characters.

Special Registers. Certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features.

Standard Data Format. The concept used in describing data in a COBOL Data Division under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer or on a particular medium.

Statement. A syntactically valid combination of words, literals, and separators, beginning with a verb, written in a COBOL source program.

Sub-Queue. A logical hierarchical division of a queue.

Subject of Entry. An operand or reserved word that appears immediately following the level indicator or the level-number in a Data Division entry.

Subprogram. A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit. The term is synonymous with called program.

Subscript. An occurrence number represented by either an integer, a data-name optionally followed by an integer with the operator + or -, or an index-name optionally followed by an integer with the operator + or -, which identifies a particular element in a table.

Subscripted Data-Name. An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

Switch. A switch to a COBOL program is a character string that can be appended to the program-name when a program is started and defined in the SPECIAL-NAMES paragraph of the Environment Division.

Switch-Status Condition. The proposition, for which a truth value can be determined, that a switch, capable of being set to an `on' or `off' status, has been set to a specific status.

Symbolic-Character. A user-defined word that specifies a user-defined figurative constant.

System-Name. A COBOL word which is used to communicate with the operating environment.

Table. A set of logically consecutive items of data that are defined in the Data Division of a COBOL program by means of the OCCURS clause.

Table Element. A data item that belongs to the set of repeated items comprising a table.

Text-Name. A user-defined word which identifies library text.

Text Word. A character or a sequence of contiguous characters between margin A and margin R in a COBOL library, source program, or in pseudo-text which is:

(1) A separator, except for: space; a pseudo-text delimiter; and the opening and closing delimiters for nonnumeric literals. The right parenthesis and left parenthesis characters, regardless of context within the library, pseudo-text, are always considered text words.

(2) A literal including, in the case of nonnumeric literals, the opening quotation mark and the closing quotation mark which bound the literal.

(3) Any other sequence of contiguous COBOL characters except comment lines and the word `COPY', bounded by separators, which is neither a separator nor a literal.

Truth Value. The representation of the result of the evaluation of a condition in terms of one of two values: true, false.

Unary Operator. A plus (+) or a minus (-) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression by +1 or -1 respectively.

Unit. A discrete portion of a storage medium, the dimensions of which are determined by each the particular operating environment, that contains part of a file, all of a file, or any number of files. The term is synonymous with reel and volume.

Unsuccessful Execution. The attempted execution of a statement that does not result in the execution of all the operations specified by that statement. The unsuccessful execution of a statement does not affect any data referenced by that statement, but may affect status indicators.

User-Defined Word. A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

Variable. A data item whose value may be changed by execution of the object program. A variable used in an arithmetic-expression must be a numeric elementary item.

Variable Length Record. A record associated with a file whose file description or sort-merge description entry permits records to contain a varying number of character positions.

Variable Occurrence Data Item. A variable occurrence data item is a table element which is repeated a variable number of times. Such an item must contain an OCCURS DEPENDING ON clause in its data description entry, or be subordinate to such an item.

Variable origin (screen). The LINE phrase and COLUMN phrase in DISPLAY and ACCEPT statements allow the entire screen description referenced by screen-name to be moved to a different starting position on the user's display device than the starting position defined in the screen description. This capability is called variable origin.

Verb. A word that expresses an action to be taken by a COBOL compiler or object program.

Volume. A discrete portion of a storage medium, the dimensions of which are determined by each operating environment, that contains part of a file, all of a file, or any number of files. The term is synonymous with reel and unit.

Word. A character-string of not more than 30 characters which forms a user-defined word, a system-name, or a reserved word.

Working-Storage Section. The section of the Data Division that describes working storage data items, composed either of noncontiguous items or working storage records or of both.

77-Level-Description-Entry. A data description entry that describes a noncontiguous data item with the level-number 77.

APPENDICES

A. IMPLEMENTATION LIMITS. 817
B. ESCAPE KEY TABLE. 819
C. ANSI 74 FILE STATUS CODES. 821
D. ANSI 85 FILE STATUS CODES. 823
E. *VXCOBOL* FILE STATUS CODES. 825
F. ANSI 74 and ANSI 85 EXCEPTION STATUS CODES. 827
G. *VXCOBOL* EXCEPTION STATUS CODES. 835
H. UNIX Errno. 843
I. ASCII CODES. 844
J. EBCDIC CODES. 846
K. COBOL RESERVED WORDS. 847
L. SYSTEM CALLS. 851

APPENDIX A. IMPLEMENTATION LIMITS

Compiler and Runtime Implementation Limits

1. The maximum length of user-defined words (data-names, condition-names, file-names, index-names, etc.) kept to determine uniqueness is 30 for ***ANSI 74*** and ***ANSI 85*** and 50 for ***VXCOBOL***.

2. The maximum length of an elementary data item is 65535 characters.

3. The maximum number of subscripts for a table is 7.

4. The maximum number of occurrences for a table is 16,777,215; storage limitations will effectively cause a lower limit.

5. The maximum number of items in the USING phrase of a CALL statement or a Procedure Division header is 32.

6. The maximum length of literals is 256.

7. The maximum number of entries in a GO TO DEPENDING ON is 254.

8. The maximum number of corresponding items in a MOVE CORRESPONDING statement is 126, while the maximum number of corresponding items in an ADD or SUBTRACT CORRESPONDING statement is 63.

9. File I/O

   For a sequential data-sensitive file the maximum record size is 2048 bytes, otherwise the maximum record size is 32,768 bytes..

   ICISAM files

   For a relative file, the maximum record size is 16,384 bytes (16KB) (version 7). Versions 5 and 6 restrict this to 4096 bytes.
   For a relative file, the maximum number of keys (relative key) is 4,294,967,296 (2^{32}) (version 7). Versions 5 and 6 restrict this to 65534.

   For an indexed file, the maximum record size is 16,384 bytes (16KB) (version 7). Versions 5 and 6 restrict this to 4096 bytes.
   For an indexed file, the maximum number of alternate keys is 16 (version 7). Versions 5 and 6 restrict this to 4.
   For an indexed file, the maximum size of a key is 255 bytes (version 7). Versions 5 and 6 restrict this to 100 bytes.

   The maximum file size is 4GB for data and 4GB for index but is dependent on the operating system support on a particular platform.
   The maximum number of alternate keys is 16.

10. The maximum number of COPY files for a given multi-file compile in an IDE project is 6000.

11. The maximum number of lines per compile is 60,000.

12. The maximum code size per program is 16,777,215 bytes.

13. The maximum data size per program is 16,777,215 bytes.

14. The maximum number of STRING/UNSTRING operands is 63.

15. The maximum number of level 88 values is 100.

16. The maximum number of KEY IS clauses on an OCCURS clause is 10.

17. The maximum number of sort/merge keys is 20.

18. The maximum number of DISPLAY arguments is 63.

Runtime Implementation Limits

1. The maximum depth of a PERFORM is 31.

2. The maximum number for PERFORM x TIMES is 4,294,967,294 (2^{32} -2).

INFOS Implementation Limits (***VXCOBOL***)

The U/FOS™ product from Transoft®, is used to provide INFOS functionality.

1. The maximum number of key volumes is 16.

2. The maximum number of data volumes is 16.

3. The maximum volume size is 1053MB with large pages and 526MB with small pages.

4. The maximum key size is 255 bytes.

5. The maximum total key path including nulls between keys is 255 bytes.

6. The maximum data length is 4074 bytes for large pages and 2026 bytes for small pages.

7. The maximum number of levels is 8.

APPENDIX B. ESCAPE KEY TABLE

This is the default ESCAPE KEY table for a Data General D2xx compatible terminal which had 15 function keys (F1 - F 15) with four states (alone, SHIFT, CTRL, and CTRL-SHIFT), 4 function keys (C1 - C4) with two states, along with the arrow keys with two states and two special keys.

Key	Key alone	Key + SHIFT	Key + CTRL	Key + SHIFT+CTRL
CR	00	00	00	00
NEWLINE	00	00	00	00
ESC	01	01	01	01
F1	02	10	18	26
F2	03	11	19	27
F3	04	12	20	28
F4	05	13	21	29
F5	06	14	22	30
F6	07	15	23	31
F7	08	16	24	32
F8	09	17	25	33
F9	34	41	48	55
F10	35	42	49	56
F11	36	43	50	57
F12	37	44	51	58
F13	38	45	52	59
F14	39	46	53	60
F15	40	47	54	61
C1	62	66	62	66
C2	63	67	63	67
C3	64	68	64	68
C4	65	69	65	69
Down-arrow	*d	77	*d	77
Up-arrow	*u	70	*u	70
Right-arrow	n/a	71	n/a	71
Left-arrow	n/a	72	n/a	72
CMD-Print	73	74	73	74
Home	n/a	75	n/a	75

*d In a multi-field screen ACCEPT, goes to the next field (down-arrow), unless on the last field. On the last field (or only field) a 00 is returned. (See Next Field definition under Terminal Description).

*u In a multi-field screen ACCEPT, goes to the previous field (up-arrow), unless on the first field. On the first field (or only field) a beep is sounded. (See Previous Field definition under Terminal Description).

In all operating environments, **ICOBOL** supports terminal types other than Data General terminals with a different number of function keys. For more on those terminals supported and their supported function key values see the Installing and Configuring Interactive COBOL Manual for your operating environment.

If using pc's, either native or with a terminal emulator, the standard PC keyboard has twelve function keys (F1 - F12). With the pcwindow terminal type, four states are available with F1 - F12.

APPENDIX C. ANSI 74 FILE STATUS CODES

Code Meaning

00 Successful I/O operation.

02 Successful I/O operation but a duplicated key was detected.

04 Successful read but length of record does not conform to that specified for file.

10 AT END condition.

11 During a READ NEXT or READ PREVIOUS of an ISAM or relative file, another program added a record to the file. Use the START statement to reposition the record pointer.

21 RECORD KEY error. For an ISAM or relative file in sequential access mode, a WRITE statement used a RECORD KEY value that was not greater than the value used in the previous WRITE.

22 INVALID KEY error.
(1) An attempt has been made to write or rewrite a record that would create a duplicate primary key.
(2) An attempt has been made to UNDELETE a record that was not deleted.

23 No record exists with the specified RECORD KEY value.

24 Index structure is full. Writing a new record would necessitate creating a new index level beyond the allowable levels.

30 Hardware error or other undefined error like a print device was aborted by the user.

34 Out of disk space to write a new record.

91 OPEN error.
(1) An OPEN statement referred to a file that was nonexistent, already open, or had an illegal name.
(2) A CLOSE statement referred to a file that had not been opened.
(3) On OPEN, the filename already existed.
(4) On OPEN, a nondirectory argument was in the pathname.
(5) On OPEN, a zero-length filename was specified.
(6) On OPEN, no more files could be opened by the operating system.
(7) On OPEN, for devices the hardware is not present.
(8) On a data-sensitive READ, the line is too long for the record.

92 Access mode error. (1) File not opened.
(2) WRITE or DELETE attempted to file opened for input.
(3) READ attempted for file opened for output.
(4) OPEN attempted for file closed with lock.
(5) DELETE or REWRITE statement not preceded by a READ statement for a file in sequential access mode.
(6) OPEN attempted on a file with insufficient access rights for OPEN mode.

94 In Use Error. (1) File cannot be accessed because it is in use.
(2) Record cannot be accessed because it is locked.
(3) DELETE FILE attempted for an opened file.

96 A directory named by the program does not exist.

97 Maximum number of open files exceeded.

98 Attempt to write more than 65,535 records to a relative file.

99 Printer control file is full.

9A File description inconsistency. Record length, key length, or key positions specified in program does not agree with the data file. Invalid ICISAM version.

9B Corruption error.
(1) After a successful OPEN of an ICISAM file, the runtime system has detected possible corruption in the file. Close this file; this sets the ICISAM reliability flags and prevents further access to the file.
(2) Data (.XD) portion of an ICISAM or relative file is full. The ICISAM reliability flags are set.
(3) On an attempted OPEN of an ICISAM file, **ICOBOL** has detected that the file is possibly corrupt although the ICISAM reliability flags are clear. The appropriate reliability flag(s) are set and the file is not opened.

9C Index (.NX) portion of an ICISAM or relative file is full. The ICISAM reliability flags are not set.

9E Record lock limit has been exceeded.

9F Possible corruption of an ICISAM file has been detected on an attempted OPEN of the file; i.e., one or both of the ICISAM reliability flags had previously been set.

9T Device Timeout.

APPENDIX D. ANSI 85 FILE STATUS CODES

Code	Meaning
00	Successful I/O operation.
02	Successful I/O operation but a duplicated key was detected.
04	Successful READ operation but the length of the record does not conform to that specified for the file.
05	Successful OPEN operation but the referenced optional file was not present, it was created if an OPEN I-O or EXTEND.
10	AT END condition.
11	During a READ NEXT or READ PREVIOUS of an ISAM or relative file, another program added a record to the file. Use the START statement to reposition the record pointer.
21	RECORD KEY error. For an ISAM or relative file in sequential access mode, a WRITE statement used a RECORD KEY value that was not greater than the value used in the previous WRITE.
22	INVALID KEY error. (1) An attempt has been made to write or rewrite a record that would create a duplicate key on a key that does not support duplicates. (2) An attempt has been made to UNDELETE a record that was not deleted.
23	No record exists with the specified RECORD KEY value.
24	Index structure is full. Writing a new record would necessitate creating a new index level beyond the allowable levels for an indexed file. Attempt to write more than 65,535 records to a relative file.
30	Hardware error or other undefined error.
34	Boundary error. (1) Out of space to write a new record (Out of disk space). (2) Out of space to READ a record (record area is too small).
35	File not found. On OPEN with INPUT, I-O, or EXTEND a non optional file was not present.
37	Access error. On OPEN the specified file does not support the open mode specified.
38	On OPEN the specified file was previously closed with lock.
39	On OPEN a File description inconsistency was detected. Record length, key length, or key positions specified in program does not agree with the data file.
41	An OPEN was attempted for a file that was already open.
42	A CLOSE was attempted for a file that was not open.
43	DELETE or REWRITE statement not preceded by a READ statement for a file in sequential access mode.
44	On a WRITE or REWRITE a record that is larger or smaller than what the file allows was attempted or on a REWRITE the record is not the same size.
46	On a sequential READ no valid next record is available.
47	A READ or START was attempted for file not opened for input or I-O.
48	A WRITE was attempted on a file not open in I-O, output, or extend mode.
49	A DELETE, REWRITE, or UNDELETE was attempted for a file not opened in I-O mode.
91	OPEN error. (1) An OPEN statement referred to a file that was nonexistent or had an illegal name. (3) On OPEN, the filename already existed. (4) On OPEN, a nondirectory argument was in the pathname. (5) On OPEN, a zero-length filename was specified. (6) On OPEN, no more files could be opened from the operating system. (7) On OPEN, for devices the hardware is not present.
92	Access mode error. (1) File not opened.
94	In Use Error. (1) File cannot be accessed because it is in use. (2) Record cannot be accessed because it is locked. (3) DELETE FILE attempted for an opened file.
96	A directory named by the program does not exist.
97	Maximum number of open files exceeded.
99	Printer control file is full.

9A Invalid ICISAM file version.

9B Corruption error.

(1) After a successful OPEN of an ICISAM file, the runtime system has detected possible corruption in the file. Close this file; this sets the ICISAM reliability flags and prevents further access to the file.

(2) Data (.XD) portion of an ICISAM or relative file is full. The ICISAM reliability flags are set.

(3) On an attempted OPEN of an ICISAM file, **ICOBOL** has detected that the file is possibly corrupt although the ICISAM reliability flags are clear. The appropriate reliability flag(s) are set and the file is not opened.

9C Index (.NX) portion of an ICISAM or relative file is full. The ICISAM reliability flags are not set.

9E Record lock limit has been exceeded.

9F Possible corruption of an ICISAM file has been detected on an attempted OPEN of the file; i.e., one or both of the ICISAM reliability flags had previously been set.

9T Device Timeout.

APPENDIX E. *VXCOBOL* FILE STATUS CODES

Code Meaning

00 Successful I/O operation.

02 Successful I/O operation but a duplicated key was detected.

10 AT END condition. (end-of-file or end of subindex)

11 During a READ NEXT or READ PREVIOUS of an ICISAM indexed or relative file, another program added a record to the file. Use the START statement to reposition the record pointer.

21 RECORD KEY error. For an ICISAM indexed or relative file in sequential access mode, a WRITE statement used a RECORD KEY value that was not greater than the value used in the previous WRITE.

22 INVALID KEY error.
- (1) An attempt has been made to write or rewrite a record that would create a duplicate primary key.
- (2) An attempt has been made to UNDELETE a record that was not deleted.

23 No record exists with the specified RECORD KEY value.

24 Index structure is full. Writing a new record would necessitate creating a new index level beyond the allowable levels. (ICISAM only)

30 Hardware error or other undefined error like a print device was aborted by the user.

34 Out of disk space to write a new record.

91 OPEN error.
- (1) An OPEN statement referred to a file that was nonexistent, already open, or had an illegal name.
- (2) A CLOSE statement referred to a file that had not been opened.
- (3) On OPEN, the filename already existed.
- (4) On OPEN, a nondirectory argument was in the pathname.
- (5) On OPEN, a zero-length filename was specified.
- (6) On OPEN, no more files could be opened by the operating system.
- (7) On OPEN, for devices the hardware is not present.
- (8) On a data-sensitive READ, the line is too long for the record.

92 Access mode error.
- (1) File not opened.
- (2) WRITE or DELETE attempted to file opened for input.
- (3) READ attempted for file opened for output.
- (4) OPEN attempted for file closed with lock.
- (5) DELETE or REWRITE statement not preceded by a READ statement for a file in sequential access mode.
- (6) OPEN attempted on a file with insufficient access rights for OPEN mode.

93 Write Verification error.

94 In Use Error.
- (1) File cannot be accessed because it is in use.
- (2) Record cannot be accessed because it is locked.
- (3) DELETE FILE or EXPUNGE attempted for an opened file.

96 A directory named by the program does not exist or with INFOS the record the program is trying to access has been previously marked as logically deleted, either locally or globaly.

97 Maximum number of open files exceeded or with INFOS a REWRITE or DELETE was attempted without executing a previous READ statement for an indexed file with sequential access.

98 Attempt to write more than 65,535 records to a relative file or with INFOS while attempting to delete a primary key, the program was:
- (1) unable to access an alternate key associated with that primary key and/or
- (2) was unable to restore the file to the condition it was in before the program deleted the primary key and/or
- (3) was unable to restore the file position.pointer to its locayion prior to the delete if RETAIN POSITION was specified; the prior position may be locked.

99 A U/FOS error has occurred for which there is no corresponding File Status code. The U/FOS error code is in the INFOS Status item, if specified in the file's SELECT clause.

9A File description inconsistency. Record length, key length, or key positions specified in program does not agree with the data file. Invalid ICISAM version.

9B Corruption error.

(1) After a successful OPEN of an ICISAM file, the runtime system has detected possible corruption in the file. Close this file; this sets the ICISAM reliability flags and prevents further access to the file.

(2) Data (.XD) portion of an ICISAM indexed or relative file is full. The ICISAM reliability flags are set.

(3) On an attempted OPEN of an ICISAM file, *VXCOBOL* has detected that the file is possibly corrupt although the ICISAM reliability flags are clear. The appropriate reliability flag(s) are set and the file is not opened.

9C Index (.NX) portion of an ICISAM indexed or relative file is full. The ICISAM reliability flags are not set.

9E Record lock limit has been exceeded.

9F Possible corruption of an ICISAM file has been detected on an attempted OPEN of the file; i.e., one or both of the ICISAM reliability flags had previously been set.

9T Device Timeout.

APPENDIX F. ANSI 74 and ANSI 85 EXCEPTION STATUS CODES

Following is a list of Exception Status codes along with the File Status that will be set, if appropriate. If two File Status values are given, the first is for ***ANSI 74*** and the second is for ***ANSI 85***, if only one is given, both return the same value. If none are given, both return a file status of 30.

On Windows, errors 1 - 31 map directly to Exception Status 1 - 31, while errors 32 - 92 map to Exception Status 288 - 347, i.e., add 256 to Microsoft errors greater than 31.

On UNIX, errno maps to an Exception Status as documented in APPENDIX H.

Exception Status	74 File Status	85 File Status	Message
1	30		Invalid operation
2	91	35	File not found
3	96		Path not found
4	91		No more handles available
5	92	37	Access denied
6	92		Invalid handle
7	30		Memory control blocks bad
8	30		Insufficient memory
9	30		Invalid memory control block address
10	30		Invalid environment
11	30		Invalid format
12	30		Invalid access code
13	30		Invalid data
14			Insufficient memory to complete this operation
15	96		Invalid drive specifier
16	92		Attempt to remove current directory
17	91		Not the same device
18	91		No more files
19	30	37	Write protected disk
20	30		Unknown hardware unit
21	30		Drive is not ready
22	30		Unknown hardware command
23	30		CRC error in data
24	30		Hardware drive request is bad
25	30		Disk seek error
26	30		Unknown disk media type
27	30		Sector not found
28	30		Printer out of paper
29	30		Write fault
30	30		Read fault
31	30		General failure
32	94		The file already exists
33	94		The file is exclusively opened
34	34		The filesize is too big
35	94	41	Attempt to exclusively open an open file
36	91		The filename is not valid
37	10		End of file
38	98	24	Invalid relative key
39	34		Out of (disk) space
40	91	34	Readline argument is too long
41	91	41	Attempt to open an open file
42	91	42	Attempt to close a closed file
43	92	38	Attempt to open a locked file
44	99		Printer control file is full
45	92		Invalid operation for open mode
46	92		Handle is not open
47	94		Attempt to delete an open file
48	92	34	Record area size too small for record
49	92	44	Record size mismatch on rewrite
50	9A	39	Record too long
51	9A	39	Too many keys requested
52	9A	39	Invalid key packet length

Exception Status	74 File Status	85 File Status	Message
53	9A	39	Key is too long
54	9A	39	Invalid key definition (not in record)
55	9A	39	Record size mismatch on open
56	9A	39	Number of keys mismatch on open
57	9A	39	Key size mismatch on open
58	9A	39	Key offset mismatch on open
59	9A		.NX file version is not valid
60	9A		.XD file version is not valid
61	9E		Out of record locks
62	94		Record is locked
63	23	46	Invalid current record pointer
64	23		Record is deleted
65	22		Record is not deleted
66	21		Not rewriting the current record
67	23		Key not found
68	22		Attempt to write a duplicate key
69	24		.NX file B-tree is full (node depth or full node)
70	21		Not writing in ascending order
71	9B		The .NX file is corrupt
72	9B		The .XD file is corrupt
73	9F		Reliability flag indicates .NX file is corrupt
74	9F		Reliability flag indicates .XD file is corrupt
75	94		Attempt to rename an open file
76	9T		Device timeout
77	30		Device I/O error
78	30		Printer is offline
79	30		Printer is out of paper
80	30		I/O operation aborted by console interrupt
81	91		Device is not available or does not exist
82	9B		The file format is not valid
83	9B		The file does not have the correct revision
84	9B		Record size is zero
85	9B		Record position is too small
86	9B		Record position is not aligned
87	9B		Record position is too big
88	9B		Record position is past EOF
89	9B		Node block number is not zero
90	9B		Node block number is zero
91	9B		Node block number is too big
92	9B		Duplicates are permitted
93	9B		Duplicates are not permitted
94	9B		Key size is zero
95	9B		Node block number is past EOF
96	9B		.XD file size is too small
97	9B		.NX file size is too small
98	9B		Key entry is deleted
99	9B		Record position does not match
100	9B		File version does not match
101	9B		Node block number is inconsistent
102	9B		Node entry count is zero
103	9B		Node entry count is too big
104	9B		Node entry count is the maximum
105	9B		Node level is inconsistent
106	9B		Node key number is inconsistent
107	9B		Node leaf indicator is inconsistent
108	9B		Position is not aligned on a node boundary
109	9B		Relative key value is inconsistent
110	9B		key value is inconsistent
111	00		Reliability flag(s) have been cleared
112	9B		Internal error - invalid use of buffer manager
113	9B		Attempt to release buffer not in use
114	9B		No buffers were available
115	9B		Attempt to destroy buffer still in use
116	9B		The object definition is in use (internal error)
117	97		No more files may be OPENed

Exception Status	74 File Status	85 File Status	Message
118	97		No more OPEN resources are available
119	97		No more SEQUENTIAL files may be OPENed
120	97		No more RELATIVE files may be OPENed
121	97		No more INDEXED files may be OPENed
122	30		Data Carrier Detect (DCD) has been lost
123	30		The requested object definition is not registered (internal error)
124			The path does not specify a directory
125			I/O aborted by WATCH interrupt
126			This terminal has too few lines to watch the selected terminal
127	30		The object does not match the expected object type (internal error)
128			Console interrupts are disabled
129			Aborted by DUMP interrupt
130	97		Object handle or index entry is NULL (internal error)
131	9B		No data is available
132	9A		Named item is the wrong type to perform this operation
133	91		The parameter string is not valid for this object
134	91		-unused--
135	97		Not enough resources to complete request
136	30		Internal system error
137	30		Invalid argument to system call
138	92		A remote computer can not be specified for this operation
139	02		A duplicate key value has been written
140	02		A duplicate key value has been read
141	30	9B	File standard header is not valid
142	30	9B	File standard header checksum is bad
143	30	9B	File type does not match required type
144	30	9B	File header length, offset, or checksum is bad
145	30	9B	File has wrong byte order
146	9A	39	Key with duplicates specification does not match
147	9A	39	ICISAM file format does not match
148	9A	39	ICISAM file version does not match
149	92	39	The .NX and .XD files are not properly paired
150	9A	39	Purge deleted records mismatch on open
151	9A	39	Key null value suppression specification does not match
152	9A	39	Key uppercase conversion specification does not match
153	00	05	File was created
154	00	05	The optional file was not available
155	92	47	Invalid operation for file without input access
156	92	48	Invalid operation for file without output access
157	92	49	Invalid operation for file without I-O access
158	92	43	DELETE or REWRITE not preceded by a successful READ
159	9B		The header information from the .XD and .NX file is not consistent
160	30		A Sort or Merge operation is already active
161	92	10	Optional file was unavailable for sequential READ
162	92	23	Optional file was unavailable for random READ or START
163	30	14	The relative key value exceeds the size of the relative key on READ
164	30	24	The relative key value exceeds the size of the relative key on WRITE
165	9B		Position is not aligned on a shared page boundary
166	22		Attempt to modify an unmodifiable key
167	94		Attempt to rewrite a record which has been modified since it was read
168	94		Attempt to perform an operation which would lead to a deadlock situation
169	9B		Invalid record length value in record header
170	9A	39	Too many key occurs requested
171	9A	39	Too many key suffixes requested
172	9A	39	Too many key alsos requested
173	9A	39	Key occurs/also specification does not match
174	9A	39	Key occurs/also count does not match
175	9A	39	Key occurs span specification does not match
176	9A	39	Key suffix count specification does not match
177	9A	39	Key reverse order specification does not match
178	30		The .XL and .XD files are not properly paired
179	30		Begin/end transition is not in sequence
180	30		Invalid combination of open options
181	30		An invalid or corrupted network packet was received
182	30		Data value is not a valid data-type-value

Exception Status	74 File Status	85 File Status	Message
183	30		Data does not fit in the data area provided
184	9A	39	4GB maximum file specification does not match
185	92	44	Record size specified exceeds the maximum or is less than the minimum for the file
186			*ERROR:
187			Conversion error (index register overflow)
188			An index is out of range
189			The perform count is too large
190			The perform stack has overflowed
191			Fatal I/O error
192	04		Length of record does not conform to that specified for the file
193			The program was terminated by a console interrupt
194			**stop run**
195			Fatal Runtime System Error
196			Fatal Runtime System Error: invalid operation code
197			The system is ready. Press Newline to begin LOGON
198			The system is currently unavailable
199			The program was terminated by another console
200			The program is too big
201			–blank--
202			The program file is not valid
203			The program was not found
204			–blank--
205			–blank--
206			–blank--
207			The program is already active
208			Attempt to call too many programs
209			Parameter mismatch in call
210			–blank--
211			–blank--
212			No more programs are available
213			The program file could not be loaded
214			–blank--
215			The program had been disabled
216			--unused--
217			--unused--
218			--unused--
219			Invalid task number
220			There are no more entries in the table
221			This operation is not permitted
222			The item is currently in use
223			The item was removed
224			The requested page is not in the file
225			--unused--
226			--unused--
227			--unused--
228			The terminal is not logged on
229			The terminal is not configured into the system
230			The configuration file is not valid
231			--unused--
232			--unused--
233			--unused--
234			The abort request was sent to terminal
235			The message was sent to terminal
236			The maximum number of users are already running
237			The option is not a valid option
238			--unused--
239			Process initialization error
240			The option requires an argument
241			The argument is too long to process
242			There are no more options to process
243			Out of processes, system resources, or no data available
244			Shared memory initialization error
245			Shared area revision does not match
246			The shared area is not ready for use

Exception Status	74 File Status	85 File Status	Message
247			Semaphore initialization error
248			No more processes can be run
249			Username:
250			The Shared Resource Executive Agent (icexec) is required
251			Process termination (Quit/Logoff)
252			Program not authorized
253			Process termination (Modem Hangup)
254			The process was terminated by a global timeout
255			Process Termination (Shutdown)
256			Insufficient memory for File Table
257			Insufficient memory for File Name Table
258			–unused--
259			Insufficient memory for File Control Tables
260			Insufficient memory for Device Table
261			Insufficient memory for Device Control Tables
262			Unable to initialize standard input file
263			Unable to initialize standard output file
264			Unable to initialize standard error file
265			Invalid type for stdio file
266			Locking Open/Close semaphore
267			Insufficient memory for Handle Table
268			Insufficient memory for Buffer Control Tables
269			Insufficient memory for Buffers
270			Insufficient memory for Cache Control Tables
271			Insufficient memory for UID Tables
272			Insufficient memory for INDEXED file table
273			–unused--
274			Insufficient memory for RELATIVE file table
275			Insufficient memory for License Control tables
276			Insufficient memory for Record Lock table
277			–unused--
278			–unused--
279			Too many directories in path list
280			Insufficient memory for pathlist names
281			Insufficient memory for Library Sub-File Table
282			Too many libraries have been requested
283			–unused--
284			ICEXEC was abnormally terminated
285			Insufficient memory for ISAM File Table
286			Insufficient memory for Network Control Table
287			Insufficient memory for SEQUENTIAL File Table
288	92		Sharing violation
289	94		Lock violation
290	30		Invalid disk change
291	30		FCB unavailable
292	30		Sharing buffer overflow
293	30		-reserved-
294	30		Out of Input
295	34		Insufficient disk space
296-305	30		-reserved-
306	30		Network request not supported
307	30		Remote computer is not available
308	30		Duplicate name on network
309	30		Network path not found
310	30		Network busy
311	30		Network device no longer exists
312	30		Net BIOS command limit exceeded
313	30		Network adapter hardware error
314	32		The specified server cannot perform the requested operation
315	30		Unexpected network error
316	30		Incompatible remote adapter
317	30		Print queue full
318	30		Not enough space for print file
319	30		Print file was deleted
320	30		The specified network name is no longer available

Exception Status	74 File Status	85 File Status	Message
321	92	37	Network access denied
322	30		Network resource type incorrect
323	30		Network name not found
324	30		Network name limit exceeded
325	30		Net BIOS session limit exceeded
326	30		Temporarily paused
327	30		No more connections can be made to this remote computer at this time
328	30		Print or disk redirection is paused
329-333	30		-reserved-
334	30		Not logged in or Network name not valid
335	30		-reserved-
336	94		File exists
337	30		-reserved-
338	30		Cannot make directory entry
339	30		Fail on INT 24
340	30		Too many redirections
341	30		Duplicate redirection
342	30		Invalid password
343	30		Invalid parameter
344	30		Network data fault
345	30		The system cannot start another process at this time
346	30		Required system component not installed
347-364	30		–reserved–
365			Connection broken
366-377			–reserved--
378			The data area passed to a system call is too small.
379-415			–reserved--
416			Record Manager initialization failed
417			The Record Manager or Requester is inactive
418			The Record Manager or Requester interface is invalid
419			Record Manager does not implement the required capability
420			Record Manager returned a reserved status code
421			Record Manager returned a generic status code
422			Record Manager returned an undefined status code
423			–blank--
424			–blank--
425			–blank--
426			–blank--
427			–blank--
428			–blank--
429			–blank--
430			–blank--
431			–blank--
432			The terminal is already being WATCH'ed
433			Cannot watch a pushed terminal
434			Cannot watch a watching terminal
435			A watched terminal cannot watch another
436			Cannot interrupt the terminal to watch
437			Watched terminal has logged off
438			Watched terminal has pushed to CLI. Press Interrupt to discontinue watching.
439			Invalid operation for your own terminal
440			Watched terminal terminated itself with an error
441			Watched terminal terminated by interrupt
442			The process is defunct
443			The watched terminals program process has terminated
444			Cannot watch an SP2 or CGI server process
448			Operation would block
449			Operation now in progress
450			Operation already in progress
451			Socket operation on non-socket
452			Destination address required
453			Message too long
454			Protocol wrong type for socket
455			Protocol not available

Exception Status	74 File Status	85 File Status	Message
456			Protocol not supported
457			Socket type not supported
458			Operation not supported on socket
459			Protocol family not supported
460			Address family not supported
461			Address already in use
462			Cannot assign requested address
463			Network is down
464			Network is unreachable
465			Network dropped connection on reset
466			Software caused connection abort
467			Connection reset by peer
468			Out of stream resources
469			Socket is already connected
470			Socket not connected
471			Cannot send after socket shutdown
472			Too many connection, cannot splice
473			Connection timed out
474			Connection refused
475			Too many symbolic links in path
476			Filename too long
477			Host is down
478			No route to host
479			Host not found
480			–blank--
481			No more streams resources available
482			The user account already exists
483			The password is too short or fails some other restriction
484			This beta release expired
485			This beta release will run until
486			Reference modification position out of range
487			Reference modification length out of range
500			ExitCode 0: Processing completed successfully
501			ExitCode 1: Processing occurred, but had errors
502			ExitCode 2: Processing occurred, but was interrupted or aborted
503			ExitCode 3: Processing occurred, but was halted by a fatal internal error
504			ExitCode 4: Processing failed because of command-line errors
505			ExitCode 5: Processing failed because of an authorization failure
506			ExitCode 6: Processing failed because of program initialization errors
507			ExitCode 7: Processing did not occur because command-line help was requested instead
508			ExitCode 8: Processing did not occur because a command-line status request ran instead
509			ExitCode 9: reserved
510			Unimplemented operating system function
511			Unexpected operating system error

APPENDIX G. *VXCOBOL* EXCEPTION STATUS CODES

Following is a list of Exception Status codes along with the File Status and INFOS Status that will be set, if appropriate. If an xxx is given for an INFOS Status then the INFOS Status will be a character string with an "x" in front of the Exception Status, as "x7".

On Windows, errors 1 - 31 map directly to Exception Status 1 - 31, while errors 32 - 92 map to Exception Status 288 - 347, i.e., add 256 to Microsoft errors greater than 31.

On UNIX, errno maps to an Exception Status as documented in APPENDIX H.

Exception Status	File Status	INFOS Status	Message
0	00	0	No error
1	30	001	Invalid operation
2	91	025	File not found
3	91	023	Path not found
4	91	035	No more handles available
5	91	0102	Access denied
6	92	0147	Invalid handle
7	30	xxx	Memory control blocks bad
8	30	005	Insufficient memory
9	30	xxx	Invalid memory control block address
10	30	xxx	Invalid environment
11	30	xxx	Invalid format
12	30	xxx	Invalid access code
13	30	0221207	Invalid data
14	30	xxx	-- reserved --
15	30	xxx	Invalid drive specifier
16	91	031	Attempt to remove current directory
17	91	0221222	Not the same device
18	91	0221227	No more files
19	30	0122	Write protected disk
20	30	xxx	Unknown hardware unit
21	30	0121	Drive is not ready
22	30	xxx	Unknown hardware command for device
23	30	070	CRC error in data
24	30	xxx	Hardware drive request is bad
25	30	0155	Disk seek error
26	30	0104	Unknown disk media type
27	30	xxx	Sector not found
28	30	xxx	Printer out of paper
29	30	0121	Write fault
30	30	075	Read fault
31	30	xxx	General failure
32	91	026	The file already exists
33	94	0204	The file is exclusively opened
34	34	0146	The file size is too big
35	94	063	Attempt to exclusively open an open file
36	91	024	The filename is not valid
37	10	030	End of file
38	23	015101	Invalid relative key
39	34	021	Out of (disk) space
40	99	067	Readline argument is too long
41	92	03	Attempt to open an open file
42	92	065	Attempt to close a closed file
43	92	xxx	Attempt to open locked file
44	30	xxx	Printer control file is full
45	92	077	Invalid operation for open mode
46	92	02	Handle is not open
47	94	0356	Attempt to delete an open file
48	99	015022	Record area size too small for record
49	97	xxx	Record size mismatch on rewrite
50	9A	xxx	Record is too long
51	9A	xxx	Too many keys requested
52	9A	xxx	Invalid key packet length
53	9A	xxx	Key is too long
54	9A	xxx	Invalid key definition (not in record)
55	9A	xxx	Record size mismatched on open
56	9A	xxx	Number of keys mismatched on open
57	9A	xxx	Key size mismatch on open
58	9A	xxx	Key offset mismatch on open
59	9A	xxx	.NX file version is not valid
60	9A	xxx	.XD file version is not valid
61	9E	07034	Out of record locks
62	94	07015	Record is locked
63	23	xxx	Invalid current record pointer
64	96	015017	Record is deleted
65	23	07030	Record is not deleted
66	21	xxx	Not rewriting the current record
67	23	xxx	Key not found
68	22	07013	Attempt to write a duplicate key
69	24	xxx	.NX file B-tree is full (node depth or a full node)
70	21	xxx	Not writing in ascending order

Exception Status	File Status	INFOS Status	Message
71	9B	xxx	The .NX file is corrupt
72	9B	xxx	The .XD file is corrupt
73	9F	xxx	Reliability flag indicates .NX file may be corrupt
74	9F	xxx	Reliability flag indicates .XD file may be corrupt
75	94	xxx	Attempt to rename an open file
76	9T	076	Device timeout
77	30	070	Device I/O error
78	30	xxx	Printer is offline
79	30	xxx	Printer is out of paper
80	30	0221205	I/O operation aborted by console interrupt
81	91	0221206	Device is not available or does not exist
82	9B	xxx	The file format is not valid
83	9B	xxx	The file does not have the correct revision
84	9B	xxx	Record size is zero
85	9B	xxx	Record position is too small
86	9B	xxx	Record position is not aligned
87	9B	xxx	Record position is too big
88	9B	xxx	Record position is past EOF
89	9B	xxx	Node block number is not zero
90	9B	xxx	Node block number is zero
91	9B	xxx	Node block number is too big
92	9B	xxx	Duplicates are permitted
93	9B	xxx	Duplicates are not permitted
94	9B	xxx	Key size is zero
95	9B	xxx	Node block number is past EOF
96	9B	xxx	.XD file size is too small
97	9B	xxx	.NX file size is too small
98	9B	xxx	Key entry is deleted
99	9B	xxx	Record position does not match
100	9B	xxx	File version does not match
101	9B	xxx	Node block number is inconsistent
102	9B	xxx	Node entry count is zero
103	9B	xxx	Node entry count is too big
104	9B	xxx	Node entry count is the maximum
105	9B	xxx	Node level is inconsistent
106	9B	xxx	Node key number is inconsistent
107	9B	xxx	Node leaf indicator is inconsistent
108	9B	xxx	Position is not aligned on a node boundary
109	9B	xxx	Relative key value is inconsistent
110	9B	xxx	Key value is inconsistent
111	00	0	Reliability flags(s) have been cleared
112	9B	xxx	Attempt to destroy buffer still in use
113	9B	xxx	Attempt to release buffer not in use
114	9B	xxx	No buffers were available
115	9B	xxx	Trash or I/O bit set and should not be
116	9B	xxx	Attempt to unlock file not locked
117	91	xxx	No more files may be OPENed
118	91	xxx	No more OPEN resources are available
119	91	xxx	No more SEQUENTIAL files may be OPENed
120	91	xxx	No more RELATIVE files may be OPENed
121	91	xxx	No more INDEXED files may be OPENed
122	30	xxx	Data Carrier Detect (DCD) has been lost
123	30	xxx	Expanded Memory manager function failed
127	30	xxx	XMS Extended memory manager function failed
130	91	xxx	There are no more names available in the name table
131	9B	xxx	No data is available
132	9A	xxx	Named item is the wrong type to perform this operation
133	91	xxx	The parameter string is not valid for this object
134	91	xxx	An invalid configuration parameter was detected
135	91	xxx	Not enough resources to complete system request
136	30	xxx	Internal system error
137	30	xxx	Invalid argument to system call
138	92	xxx	File or device must be on the same node or volume
139	02	0	A duplicate key value has been written
140	02	0	A duplicate key value has been read
141	30	xxx	File standard header is not valid
142	30	xxx	File standard header checksum is bad
143	30	xxx	File type does not match required type
144	30	xxx	File header length, offset, or checksum is bad
145	30	xxx	File has wrong byte order
146	9A	xxx	Key with duplicates specification does not match
147	9A	xxx	ICISAM file format does not match
148	9A	xxx	ICISAM file version does not match
149	91	xxx	File is already opened incompatibly
150	9A	xxx	Purge-deleted-records mismatch on open
151	9A	xxx	Key null value suppression specification does not match
152	9A	xxx	Key uppercase conversion specification does not match
153	00	xxx	File was created
154	00	xxx	The optional file was not available
155	92	xxx	Invalid operation for file without input access
156	92	xxx	Invalid operation for file without output access
157	92	xxx	Invalid operation for file without I-O access
158	97	015022	DELETE or REWRITE was not preceded by a successful READ
159	9B	xxx	The header information from the .nx and .xd file
160	30	xxx	A Sort or Merge operation is already active
161	10	030	Optional file was unavailable for sequential READ
162	23	030	Optional file was unavailable for random READ or START
163	30	xxx	Relative key value exceeds size of the relative key on READ
164	30	xxx	Relative key value exceeds size of the relative key on WRITE
165	9B	xxx	Position is not aligned on a shared page boundary
166	22	xxx	Attempt to modify unmodifiable key

Exception Status	File Status	INFOS Status	Message
167	94	xxx	Attempt to rewrite a record that has been modified since it was read
168	94	xxx	Attempt to perform an operation which would lead to a deadlock situation
169	9B	061	Invalid record length value in record header
170	9A	xxx	Too many key occurs requested
171	9A	xxx	Too many key suffixes requested
172	9A	xxx	Too may key alsos requested
173	9A	xxx	Key occurs/also specification does not match
174	9A	xxx	Key occurs/also count specification does not match
175	9A	xxx	Key occurs span specification does not match
176	9A	xxx	Key suffix count speciffication does not match
177	9A	xxx	Key reverse order specification does not match
178	30	xxx	The .XL and .XD files are not properly paired
179	30	xxx	Begin/end transition is not in sequence
180	30	xxx	Invalid combination of network options
181	30	xxx	An invalid or corrupted network packet was received
182	30	xxx	The network packet argument descriptors do not match
183	30	xxx	Data does not fit in the data area provided
184	9A	xxx	The 4GB maximum file specification does not match
185	92	xxx	Record size specified exceeds the maximum or is less than the minimum for the file
186			*ERROR:
187			Conversion error (index register overflow)
188			An index is out of range
189			The perform count is too large
190			The perform stack has overflowed
191			Fatal I/O error
192	04	0	The length of the record read does not conform to that specified for the file
193			The program was terminated by a console interrupt
194			** stop run **
195			Fatal Runtime System Error
196			Fatal Runtime System Error: invalid operation code
197			The system is ready. Press Newline to begin LOGON.
198			The system is currently unavailable
199			The program was terminated by another console
200			The program is too big
201			The program is the wrong revision
202			The program file is not valid
203			The program was not found
204			The program could not be loaded
205			Error processing using
206			The program could not be accessed
207			The program is already active
208			Attempt to call too many programs
209			Parameter mismatch in call
210			Out of .VM file space
211			Out of handles during CALL
212			No more programs are available
213			The program file could not be loaded
214			Error processing the .VM file
215			The program has been disabled
216			The program identifier is not valid
217			The program is not active
218			Too many parameters in call
219			Invalid task number
220			There are no more entries in the table
221			This operation is not permitted
222			The item is currently in use
223			The item was removed
224			The requested page is not in the file
225			Do you want to shutdown the system?
226			The system is still busy. Do you want to shut down anyway?
227			Do you want to re-enable the terminals?
228			The terminal is not logged on
229			The terminal is not configured into the system
230			The configuration file is not valid
231			The configuration file is the wrong revision
232			The link file is not valid
233			The link file is the wrong revision
234			The abort request was sent to terminal
235			The message was sent to terminal
236			The maximum number of users are already running
237			The option is not a valid option
238			The option has already been selected
239			An option specification is expected
240			The option requires an argument
241			The argument is too long to process
242			There are no more options to process
243			Out of processes, system resources, or no data available
244			Shared memory initialization error
245			Shared area revision does not match
246			The shared area is not ready for use
247			Semaphore initialization error
248			No more processes can be run
249			Username:
250			Exit from hot key call
251			Process termination (Quit/Logoff)
252			Program not authorized
253			Process termination (Modem Hangup)
254			The process was terminated by a global timeout
255			Process Termination (Shutdown)

Exception Status	File Status	INFOS Status	Message
256			Insufficient memory for File Table
257			Insufficient memory for File Name Table
258			Insufficient memory for File Name Hash Table
259			Insufficient memory for File Control Tables
260			Insufficient memory for Device Table
261			Insufficient memory for Device Control Tables
262			Unable to initialize standard input file
263			Unable to initialize standard output file
264			Unable to initialize standard error file
265			Invalid type for stdio file
266			Locking Open/Close semaphore
267			Insufficient memory for Handle Table
268			Insufficient memory for Buffer Control Tables
269			Insufficient memory for Buffers
270			Insufficient memory for Cache Control Tables
271			Insufficient memory for UID Tables
272			Insufficient memory for INDEXED file table
273			Insufficient memory for INDEXED file work area
274			Insufficient memory for RELATIVE file table
275			Insufficient memory for License Control tables
276			Insufficient memory for Record Lock table
277			Insufficient memory for Device Name Table
278			Insufficient memory for Device Name Hash Table
279			Too many directories in path list
280			Insufficient memory for pathlist names
281			Insufficient memory for Library Sub-File Table
282			Too many libraries have been requested
283			Insufficient memory for Map Control Tables
284			icexec was abnormally terminated
285			Insufficient memory for ISAM file table
286			Insufficient memory for Network Control table
287			Insufficient memory for Sequential File Table
288	92		Sharing Violation
289	94		Lock Violation
290	30		Invalid disk change
291	30		FCB unavailable
292	30		Sharing Buffer overflow
293	30		-reserved-
294	30		Out of input
295	34	021	Insufficient disk space
296-305	30		-reserved-
306	30		Network request not supported
307	30		Remote computer not listed.
308	30		Duplicate name on network
309	30		Network name not found
310	30		Network busy
311	30		Network device no longer exists
312	30		Net BIOS command limit exceeded
313	30		Network adapter hardware error
314	32		Incorrect response from network
315	30		Unexpected network error
316	30		Incompatible remote adapter
317	30		Print queue full
318	30		Not enough space for print file
319	30		Print file was deleted
320	30		Network name deleted
321	91	102	Network access denied
322	30		Network device type incorrect
323	30		Network name not found
324	30		Network name limit exceeded
325	30		Net BIOS session limit exceeded
326	30		Temporarily paused
327	30		Network request not accepted
328	30		Print or disk redirection is paused
329-333	30		-reserved-
334	30		Not logged in or Network name not valid
335	30		-reserved-
336	94		File exists
337	30		-reserved-
338	30		Cannot make directory entry
339	30		Fail on INT 24
340	30		Too many redirections
341	30		Duplicate redirection
342	30		Invalid password
343	30		Invalid parameter
344	30		Network data fault
345	30		The system cannot start another process at this time
346	30		Required system component not installed
347-377	30		-reserved-
378			The data area passed to a system call is too small.
379-415			-reserved--
416			BTRIEVE initialization failed
417			BTRIEVE Record Manager or Requester is inactive
418			BTRIEVE Record Manager or Requester interface is invalid
419			BTRIEVE does not implement the required capability
420			BTRIEVE returned a reserved status code
421			BTRIEVE returned a generic status code
422			BTRIEVE returned an undefined status code
423			C-ISAM initialization failed
424			C-ISAM is not available
425			C-ISAM interface is invalid
426			C-ISAM does not implement the required capability
427			C-ISAM returned a reserved status code

Exception Status	File Status	INFOS Status	Message
428			C-ISAM returned a generic status code
429			C-ISAM returned an undefined status code
430			Winsock initialization failed
431			Winsock interface is invalid
432			The terminal is already being WATCH'ed
433			Cannot watch a pushed terminal
434			Cannot watch a watching terminal
435			A watched terminal cannot watch another
436			Cannot interrupt the terminal to watch
437			Watched terminal has logged off
438			Watched terminal has pushed to CLI. Press Interrupt to discontinue watching.
439			Invalid operation for your own terminal
440			Watched terminal terminated itself with an error
441			Watched terminal terminated by interrupt
442			The process is defunct
443			The watched terminals program process has terminated
448			Operation would block
449			Operation now in progress
450			Operation already in progress
451			Socket operation on non-socket
452			Destination address required
453			Message too long
454			Protocol wrong type for socket
455			Protocol not available
456			Protocol not supported
457			Socket type not supported
458			Operation not supported on socket
459			Protocol family not supported
460			Address family not supported
461			Address already in use
462			Cannot assign requested address
463			Network is down
464			Network is unreachable
465			Network dropped connection on reset
466			Software caused connection abort
467			Connection reset by peer
468			Out of stream resources
469			Socket is already connected
470			Socket not connected
471			Cannot send after socket shutdown
472			Too many connection, cannot splice
473			Connection timed out
474			Connection refused
475			Too many symbolic links in path
476			Filename too long
477			Host is down
478			No route to host
480			Connection broken
481			No more streams resources available
482			The user account already exists
483			The password is too short or fails some other restriction
484			This beta release expired
485			This beta release will run until
486			Reference modification position out of range
487			Reference modification length out of range
510			Unimplemented operating system function
511			Unexpected operating system error

The following only occur with U/FOS files.

Exception Status	File Status	INFOS Status	Message
2560	99	07006	Positioned above main index
2561	99	xxx	'ufos verify' found errors
2562	99	07005	Invalid current position
2563	99	xxx	Indicates alternate index
2564	99	07777	Function not implemented
2565	99	xxx	Software has expired
2566	99	xxx	User interrupt
2567	99	xxx	Start of fatal messages (request is aborted)
2568	91	07141	Not a U/FOS database or INFOS volume, index, or database
2569	91	07211	Unable to access the key volume(s)
2570	91	07212	Unable to access the data volume(s)
2571	91	xxx	Unable to access the logging file
2572	91	xxx	Unable to access the shadow volume
2573	91	07211	Unable to access the database directory
2574	34	xxx	Exceeded maximum volume size
2575	94	07222	Database is exclusively open, cannot open
2576	94	07217	Database is open, cannot exclusively open
2577	91	07055	Database is invalid, must be fixed first
2578	91	07231	Database left open, run 'ufos_verify' or 'IVERIFY'
2579	99	07001	Index full, cannot add a new level
2580	91	07241	Access to index volume denied
2581	91	07242	Access to data volume denied
2582	91	xxx	Access to logging file denied
2583	34	xxx	U/FOS data volumes are full
2584	91	07061	Illegal channel number
2585	99	xxx	Do not have a U/FOS or INFOS database open
2586	99	07061	No database is open on this channel
2587	34	xxx	No space for main key volume
2588	34	xxx	No space for main data volume
2589	92	xxx	Database is read-only
2590	94	07076	Already opened for read/write; cannot open for sequential
2591	91	xxx	No valid primary index file
2592	91	xxx	Cannot find the primary index file
2593	94	07101	Already opened for sequential, cannot open for write
2594	92	07437	On-line backup is already in progress
2595	99	xxx	Problem opening/creating shadow volume
2596	91	xxx	Cannot use 'big-endian' (not byteswapped) database here
2597	91	xxx	Cannot use 'little-endian' (byteswapped) database here
2598	92	xxx	Database is not in on-line backup mode
2599	99	xxx	Unknown processing packet function
2600	99	07702	Invalid packet type
2601	99	xxx	Invalid keyed motion word
2602	99	xxx	Invalid keyed position word
2603	99	xxx	Invalid key address
2604	99	xxx	Invalid data record (_DREC) value
2605	99	07201	Invalid number of volumes
2606	99	xxx	Invalid split point
2607	99	07026	Invalid number of subindex levels
2608	99	07207	Inavlid maximum volume size
2609	99	xxx	Invalid volume element size
2610	99	07175	Invalid database page size
2611	99	xxx	Invalid number of concurrent locks
2612	99	xxx	__GEN or __APP not allowed on __WRITE
2613	99	xxx	Occurence number is zero for __DUP read
2614	99	07173	Filename address pointer is invalid
2615	99	07105	A key address is invalid
2616	99	07064	Record length less than 1
2617	99	07064	Record length too large for page
2618	99	07007	Subindices are not allowed in this subindex
2619	99	07020	Exceeded maximum number of subindexes
2620	99	xxx	Data record 2 (partial record) not allowed in this subindex
2621	23	07030	Key not found
2622	99	xxx	No key path has been defined
2623	22	07013	Key already exists
2624	99	xxx	Data record 1 must exist
2625	99	xxx	Data record 2 must exist
2626	23	07014	No data record 1
2627	23	xxx	No data record 2
2628	22	07023	Data record 1 already exists
2629	22	xxx	Data record 2 already exists
2630	99	xxx	Tree too deep for unlink in group
2631	99	07065	Invalid feedback value
2632	99	07006	Positioned above main index
2633	99	07005	Invalid current position
2634	23	07016	Subindex already defined
2635	23	07030	Subindex key does not exist
2636	99	07010	Subindex is not defined
2637	10	07011	End of subindex
2638	99	xxx	Key does not have subindex
2639	99	xxx	Current position has been deleted
2640	99	07034	Maximum number of locks exceeded
2641	99	07021	Cannot erase key, subindex defined
2642	99	xxx	Data record is not locked by user
2643	94	07015	Data record (data record 1) is locked by another user
2644	94	07025	Partial record (data record 2) is locked by another user
2645	99	07104	Key too long for this subindex
2646	99	07104	Zero length key
2647	22	07036	Duplicate keys not allowed in this subindex
2648	99	07043	Subindex not empty; Cannot remove subindex--indices defined
2649	99	xxx	No transaction group defined
2650	99	xxx	Invalid transaction group count

Exception Status	File Status	INFOS Status	Message
2651	23	xxx	Total key path length too long
2652	23	xxx	A single null byte '\0' is not a valid key
2653	99	xxx	No reads allowed below this index
2654	99	xxx	No writes allowed below this index
2655	99	xxx	No modifications allowed below this index
2656	99	xxx	No deletes allowed below this index
2657	92	xxx	User not privileged to access this file
2658	92	07042	User not privileged to modify this file
2659	91	xxx	Cannot find primary index for alternate index
2660	91	xxx	Alternate index points to another alternate index
2661	91	xxx	Alternate key index pointer out of bounds
2662	91	0247	Cannot find the server
2663	99	xxx	Cannot create/open pipe to server
2664	99	xxx	No setup acknowledgement from server
2665	99	xxx	Error in writing to the server
2666	99	xxx	Error in reading from the server
2667	99	07051	Allowed number of system users exceeded
2668	99	xxx	Allowed number of system databases exceeded
2669	91	07142	Allowed number of databases for this user exceeded
2670	99	xxx	Log file truncated
2671	99	xxx	Incorrect log file revision
2672	99	xxx	Attempt to access negative page number
2673	99	xxx	Page is after end of volume
2674	99	xxx	Page types do not match
2675	99	xxx	Page number mismatch
2676	99	xxx	'KEY_NXT' value is 0
2677	99	xxx	Invalid page type
2678	99	xxx	Data record address is zero
2679	99	xxx	Data record item number is invalid
2680	99	xxx	Data item pointer is out of range
2681	99	xxx	Data item is marked as deleted
2682	99	xxx	Indirect address is zero
2683	99	xxx	Page descriptor is zero
2684	99	xxx	Insert invalid first key
2685	99	xxx	Invalid subindex xref item
2686	99	xxx	Invalid key volume number
2687	99	xxx	Invalid data volume number
2688	99	xxx	Attempt to delete subindex null key
2689	99	xxx	Expanded record is not the correct length
2690	99	xxx	Key page overflows allowed size
2691	99	xxx	Data page is on the wrong space chain
2692	99	xxx	File mode in header is wrong
2693	99	xxx	Header page descriptor invalid
2694	99	xxx	Error from database lseek request
2695	99	xxx	Error from database read request
2696	99	xxx	Error from database write request
2697	99	xxx	Invalid page type request
2698	99	xxx	Bad volume number in page request
2699	99	xxx	Bad page number in page request
2700	99	xxx	Bad page number in page request
2701	99	xxx	Attempt to read part of header
2702	99	xxx	Reserved -0705
2703	99	xxx	Reserved -0706
2704	99	xxx	Attempt to write page with invalid address
2705	99	xxx	Attempt to flush and page address mismatch
2706	99	xxx	Unable to find allocated VM page for user/db
2707	99	xxx	Error when attempting to read shadow file
2708	99	xxx	Exceeded maximum number of user/database opens
2709	99	xxx	Unexpected error from U/FOS
2710	99	xxx	Invalid AOS INFOS subindex definition packet
2711	96	015017	Data record is marked as logically deleted
2712	96	015017	Partial record is marked as logically deleted
2713	99	015022	Data record read exceeds specified maximum length
2714	99	015022	Partial record read exceeds specified maximum length
2715	99	07046	Invalid partial record length
2716	92	07102	Request requires read-only access
2717	91	07120	Invalid index node size
2718	91	07213	Index filename already exists
2719	99	07240	Invalid merit factor
2746	99	xxx	Transaction group already in progress
2747	99	xxx	No transaction group in progress
2748	99	xxx	Transaction group not initiated by this user
2749	99	xxx	Invalid key length
2750	99	xxx	Cannot find banner file name
2751	99	xxx	No banner file present for this product
2752	99	xxx	Banner server not found
2753	99	xxx	Cannot attach to the banner server
2754	99	xxx	AIM tools not allowed
2755	99	xxx	Cannot get maximum number of users
2756	99	xxx	Cannot allocate maximum number of users
2757	99	xxx	Data has been corrupted in message queues
2758	99	xxx	No message available on interprocess queue
2759	99	xxx	Could not generate a valid session
2760	99	xxx	Bad session ID
2761	99	xxx	System access to server failed
2762	99	xxx	Request denied by banner server
2763	99	xxx	Corrupt or inaccessible banner name
2764	99	xxx	Banner file name entry missing from configuration file
2765	99	xxx	Error in target database
2766	99	xxx	Error reading target database
2767	99	xxx	Cannot link alternate index to another alternate index database

Exception Status	File Status	INFOS Status	Message
2768	99	xxx	No more linked indices allowed for this database
2769	99	xxx	Error positioning target database
2770	99	xxx	Unable to rewrite database header
2771	99	xxx	Unable to create alternate index symbolic link
2772	99	xxx	Unable to create alternate index
2773	99	xxx	Unable to write to alternate index
2778	99	xxx	Unable to place message on message queue
2779	99	xxx	Message send timed out
2780	99	xxx	Message receive timed out
2781	99	xxx	No space in shared memory table of kernel processes
2782	99	xxx	No space in shared memory table of databases opened
2783	99	xxx	Database request queue full
2784	99	xxx	Timed out waiting on request queue
2785	99	xxx	Failed to start monitor process
2786	99	xxx	Shared memory tables invalid
2787	99	xxx	Failed to write to log file
2788	99	xxx	Error attaching to shared memory
2789	99	xxx	Error accessing shared memory semaphore
2790	99	xxx	Error accessing file of process locks
2791	99	xxx	Error getting file status
2792	99	xxx	Error removing the semaphore set
2793	99	xxx	Error removing the shared memory set
2794	99	xxx	Error accessing database semaphore set
2795	99	xxx	Error reading page from multi-request diff file
2796	99	xxx	Error from database volume unlink request
2797	99	xxx	Error from database volume truncate request
2798	99	xxx	Invalid subindex root node size
2799	99	xxx	Database in checkpointing mode, cannot start transaction
2800	99	xxx	Cannot checkpoint, as not in checkpointing mode
2801	99	xxx	Can only perform a checkpoint with database closed
2802	99	xxx	Cannot checkpoint, as not in explicit checkpointing mode
2803	91	xxx	Cannot open database lock table volume
2804	99	xxx	Failed to write to database lock table volume
2805	99	xxx	Failed to read from database lock table volume
2806	99	xxx	Failed to start the 'ufos_connect' process
2807	99	xxx	Failed to open the network path (.unp) file
2808	99	xxx	U/FOS kernel not yet implemented on this platform
2809	99	xxx	Error in the network path (.unp) file
2810	99	xxx	Unable to access the ipc type used by the kernel
2811	99	xxx	Invalid host name specified
2812	99	xxx	Error reading the kernel configuration file
2813	99	xxx	Exceeded number of user database combinations
2814	99	xxx	Error setting effective user ID
2815	99	xxx	Error setting effective group ID
2816	99	xxx	Error setting supplementary group access list
2817	99	xxx	Log in use by kernel not supporting 'ufos_new_log'
2818	99	xxx	Failed to rename old log file
2819	99	xxx	Failed to create a new log file
2820	99	xxx	Subindex root-node page item table corrupt
2821	99	xxx	Subindex root-node item number invalid
2822	99	xxx	Database volumes contain an incomplete checkpoint
2823	99	xxx	Error processing the multiple request differential file
2824	99	xxx	Error processing subindex root node
2825	99	xxx	Error expanding subindex root node
2826	99	xxx	Error processing opening checkpoint

APPENDIX H. UNIX Errno

The following is a mapping of UNIX errors (errno) to Exception Status codes.

E2BIG	241	EHOSTUNREACH	478	ENONET	307	ESPIPE	25
EACCES	5	EIDRM	223	ENOPKG	346	ESRCH	219
EADDRINUSE	461	EINPROGRESS	315	ENOPROTOOPT	455	ESRMNT	315
EADDRNOTAVAIL	462	EINTR	80	ENOSPC	39	ETIME	76
EADV	315	EINVAL	137	ENOSR	481	ETIMEDOUT	473
EAFNOSUPPORT	460	EIO	77	ENOSTR	5	ETOOMANYREFS	472
EAGAIN	243	EISCONN	469	ENOSYS	511	ETXTBSY	5
EALREADY	315	EISDIR	5	ENOTBLK	11	EWOULDBLOCK	448
EBADF	6	ELBIN	315	ENOTCONN	470	EXDEV	17
EBADFD	6	ELOOP	475	ENOTDIR	3		
EBADMSG	344	EMFILE	4	ENOTSOCK	451		
EBUSY	33	EMLINK	340	ENOTTY	5		
ECHILD	1	EMSGSIZE	453	ENOTUNIQ	308		
ECHRNG	6	EMULTIHOP	340	ENXIO	81		
ECOMM	313	ENAMETOOLONE	476	EOPNOTSUPP	458		
ECONNABORTED	466	ENETDOWN	463	EOVERFLOW	34		
ECONNREFUSED	474	ENETRESET	465	EPERM	5		
ECONNRESET	467	ENETUNREACH	464	EPFNOSUPPORT	459		
EDEADLK	168	ENFILE	18	EPIPE	365		
EDEADLOCK	168	ENOBUFS	468	EPROTO	314		
EDESTADDRREQ	452	ENODATA	131	EPROTONOSUPPORT	456		
EDOM	137	ENODEV	81	EPROTOTYPE	454		
EDOTDOT	315	ENOENT	2	ERANGE	136		
EDQUOT	295	ENOEXEC	137	EREMCHG	320		
EEXIST	32	ENOLCK	61	EREMOTE	138		
EFAULT	9	ENOLINK	311	EROFS	29		
EFBIG	34	ENOMEM	8	ESHUTDOWN	471		
EHOSTDOWN	477	ENOMSG	131	ESOCKTNOSUPPORT	457		

any errno not listed generates a 511

APPENDIX I. ASCII CODES

Dec	Oct	Hex	DG Function	Ctrl-code	PC Function/Character
0	000	00	Null	Ctrl @	NUL space
1	001	01	Print Screen Form	Ctrl A	SOH ☺
2	002	02	Reverse off	Ctrl B	STX ☻
3	003	03		Ctrl C	ETX ♥
4	004	04		Ctrl D	EOT ♦
5	005	05	Read cursor address	Ctrl E	ENQ ♣
6	006	06		Ctrl F	Ack ♠
7	007	07	Bell	Ctrl G	Bell •
8	010	08	Cursor Home	Ctrl H	Backspace ◘
9	011	09		Ctrl I	HTab ○
10	012	0A	Newline	Ctrl J	Linefeed ◙
11	013	0B	Erase EOL	Ctrl K	VTab ♂
12	014	0C	Erase Screen	Ctrl L	Form-feed ♀
13	015	0D	Carriage Return	Ctrl M	Carriage Return ♪
14	016	0E	Blink ON	Ctrl N	SO ♫
15	017	0F	Blink off	Ctrl O	SI ☼
16	020	10	Write cursor addr(c,r)	Ctrl P	DLE ►
17	021	11	Print Screen	Ctrl Q	DC1 (XON) ◄
18	022	12	Roll Enable	Ctrl R	DC2 ↕
19	023	13	Roll Disable	Ctrl S	DC3 (XOFF) ‼
20	024	14	Underscore ON	Ctrl T	DC4 ¶
21	025	15	Underscore OFF	Ctrl U	NAK §
22	026	16	Reverse On	Ctrl V	SYN ▬
23	027	17	Cursor Up	Ctrl W	ETB ↨
24	030	18	Cursor Right	Ctrl X	CAN ↑
25	031	19	Cursor Left	Ctrl Y	EM ↓
26	032	1A	Cursor Down	Ctrl Z	SUB →
27	033	1B	Escape	Ctrl [	ESC ←
28	034	1C	Dim ON	Ctrl \	FS ∟
29	035	1D	Dim OFF	Ctrl]	GS ↔
30	036	1E	Command Header	Ctrl ^	RS ▲
31	037	1F		Ctrl _	US ▼

Dec	Oct	Hex	DG	PC
32	040	20	space	space
33	041	21	!	!
34	042	22	"	"
35	043	23	#	#
36	044	24	$	$
37	045	25	%	%
38	046	26	&	&
39	047	27	'	'
40	050	28	(	(
41	051	29	)	)
42	052	2A	*	*
43	053	2B	+	+
44	054	2C	, (comma),	
45	055	2D	-	-
46	056	2E	.	.
47	057	2F	/	/
48	060	30	0	0
49	061	31	1	1
50	062	32	2	2
51	063	33	3	3
52	064	34	4	4
53	065	35	5	5
54	066	36	6	6
55	067	37	7	7
56	070	38	8	8
57	071	39	9	9
58	072	3A	:	:
59	073	3B	;	;
60	074	3C	<	<
61	075	3D	=	=
62	076	3E	>	>
63	077	3F	?	?

Dec	Oct	Hex	DG	PC
64	100	40	@	@
65	101	41	A	A
66	102	42	B	B
67	103	43	C	C
68	104	44	D	D
69	105	45	E	E
70	106	46	F	F
71	107	47	G	G
72	110	48	H	H
73	111	49	I	I
74	112	4A	J	J
75	113	4B	K	K
76	114	4C	L	L
77	115	4D	M	M
78	116	4E	N	N
79	117	4F	O	O
80	120	50	P	P
81	121	51	Q	Q
82	122	52	R	R
83	123	53	S	S
84	124	54	T	T
85	125	55	U	U
86	126	56	V	V
87	127	57	W	W
88	130	58	X	X
89	131	59	Y	Y
90	132	5A	Z	Z
91	133	5B	[	[
92	134	5C	\	\
93	135	5D	]	]
94	136	5E	^	^
95	137	5F	_	_

Dec	Oct	Hex	DG	PC
96	140	60	`	`
97	141	61	a	a
98	142	62	b	b
99	143	63	c	c
100	144	64	d	d
101	145	65	e	e
102	146	66	f	f
103	147	67	g	g
104	150	68	h	h
105	151	69	i	i
106	152	6A	j	j
107	153	6B	k	k
108	154	6C	l	l
109	155	6D	m	m
110	156	6E	n	n
111	157	6F	o	o
112	160	70	p	p
113	161	71	q	q
114	162	72	r	r
115	163	73	s	s
116	164	74	t	t
117	165	75	u	u
118	166	76	v	v
119	167	77	w	w
120	170	78	x	x
121	171	79	y	y
122	172	7A	z	z
123	173	7B	{	{
124	174	7C	|	|
125	175	7D	}	}
126	176	7E	~	~
127	177	7F	DEL	⌂

Dec	Oct	Hex	DGI	PC
128	200	80		Ç
129	201	81		ü
130	202	82		é
131	203	83		â
132	204	84		ä
133	205	85		à
134	206	86		å
135	207	87		ç
136	210	88		ê
137	211	89		ë
138	212	8A		è
139	213	8B		ï
140	214	8C		î
141	215	8D		ì
142	216	8E		Ä
143	217	8F		Å
144	220	90		É
145	221	91		æ
146	222	92		Æ
147	223	93		ô
148	224	94		ö
149	225	95		ò
150	226	96		û
151	227	97		ù
152	230	98		ÿ
153	231	99		Ö
154	232	9A		Ü
155	233	9B		¢
156	234	9C		£
157	235	9D		¥
158	236	9E		₧
159	237	9F		ƒ
160	240	A0	space	á
161	241	A1	‾	í
162	242	A2	½	ó
163	243	A3	µ	ú
164	244	A4	²	ñ
165	245	A5	³	Ñ
166	246	A6	¤	ª
167	247	A7	¢	º
168	250	A8	£	¿
169	251	A9	ª	⌐
170	252	AA	º	¬
171	253	AB	¡	½
172	254	AC	¿	¼
173	255	AD	©	¡
174	256	AE	®	«
175	257	AF	‡	»
176	260	B0	»	░
177	261	B1	«	▒
178	262	B2	¶	▓
179	263	B3	™	│
180	264	B4	ƒ	┤
181	265	B5	¥	╡
182	266	B6	±	╢
183	267	B7	≤	╖
184	270	B8	≥	╕
185	271	B9	·	╣
186	272	BA	` (grave)	║
187	273	BB	$	╗
188	274	BC	° (degree)	╝
189	275	BD	¨ (umlaut)	╜
190	276	BE	´ (acute)	╛
191	277	BF	↑	┐
192	300	C0	Á	└
193	301	C1	À	┴
194	302	C2	Â	┬
195	303	C3	Ä	├
196	304	C4	Ã	─
197	305	C5	Å	┼
198	306	C6	Æ	╞
199	307	C7	Ç	╟
200	310	C8	É	╚
201	311	C9	È	╔
202	312	CA	Ê	╩
203	313	CB	Ë	╦
204	314	CC	Í	╠
205	315	CD	Ì	═
206	316	CE	Î	╬
207	317	CF	Ï	╧
208	320	D0	Ñ	╨
209	321	D1	Ó	╤
210	322	D2	Ò	╥
211	323	D3	Ô	╙
212	324	D4	Ö	╘
213	325	D5	Õ	╒
214	326	D6	Ø	╓
215	327	D7	Œ	╫
216	330	D8	Ú	╪
217	331	D9	Ù	┘
218	332	DA	Û	┌
219	333	DB	Ü	▉
220	334	DC	space	▄
221	335	DD	Ÿ	▌
222	336	DE	space	▐
223	337	DF	space	▀
224	340	E0	á	α
225	341	E1	à	β
226	342	E2	â	Γ
227	343	E3	ä	π
228	344	E4	ã	Σ
229	345	E5	å	σ
230	346	E6	æ	µ
231	347	E7	ç	τ
232	350	E8	é	Φ
233	351	E9	è	Θ
234	352	EA	ê	Ω
235	353	EB	ë	δ
236	354	EC	í	∞
237	355	ED	ì	φ
238	356	EE	î	ε
239	357	EF	ï	∩
240	360	F0	ñ	≡
241	361	F1	ó	±
242	362	F2	ò	≥
243	363	F3	ô	≤
244	364	F4	ö	⌠
245	365	F5	õ	⌡
246	366	F6	ø	÷
247	367	F7	œ	≈
248	370	F8	ú	°
249	371	F9	ù	·
250	372	FA	û	·
251	373	FB	ü	√
252	374	FC	β	ⁿ
253	375	FD	ÿ	²
254	376	FE	space	■
255	377	FF	space	space

▸ **Notes:**

1. Decimal codes 128 - 159 for DGI are the same as their 7-bit counterparts by default.
2. DGI is as defined by a D216E+/D217/D413/D463 terminal.

APPENDIX J. EBCDIC CODES

Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char
0	000	00	NUL	32	040	20	DS	64	100	40	space	96	140	60	-
1	001	01	SOH	33	041	21	SOS	65	101	41		97	141	61	/
2	002	02	STX	34	042	22	FS	66	102	42		98	142	62	
3	003	03	ETX	35	043	23		67	103	43		99	143	63	
4	004	04	PF	36	044	24	BYP	68	104	44		100	144	64	
5	005	05	HT	37	045	25	LF	69	105	45		101	145	65	
6	006	06	LC	38	046	26	ETB	70	106	46		102	146	66	
7	007	07	DEL	39	047	27	ESC	71	107	47		103	147	67	
8	010	08		40	050	28		72	110	48		104	150	68	
9	011	09		41	051	29		73	111	49		105	151	69	
10	012	0A	SMM	42	052	2A	SM	74	112	4A		106	152	6A	|
11	013	0B	VT	43	053	2B	CU2	75	113	4B	.	107	153	6B	,
12	014	0C	FF	44	054	2C	DC4	76	114	4C	<	108	154	6C	%
13	015	0D	CR	45	055	2D	ENQ	77	115	4D	(	109	155	6D	_
14	016	0E	SO	46	056	2E	ACK	78	116	4E	+	110	156	6E	>
15	017	0F	SI	47	057	2F	BEL	79	117	4F	|	111	157	6F	?
16	020	10	DLE	48	060	30		80	120	50	&	112	160	70	
17	021	11	DC1 (XON)	49	061	31		81	121	51		113	161	71	
18	022	12	DC2	50	062	32	SYN	82	122	52		114	162	72	
19	023	13	DC3(XOFF)	51	063	33		83	123	53		115	163	73	
20	024	14	RES	52	064	34	PN	84	124	54		116	164	74	
21	025	15	NL	53	065	35	RS	85	125	55		117	165	75	
22	026	16	BS	54	066	36	UC	86	126	56		118	166	76	
23	027	17	IL	55	067	37	EOT	87	127	57		119	167	77	
24	030	18	CAN	56	070	38		88	130	58		120	170	78	
25	031	19	EM	57	071	39		89	131	59		121	171	79	`
26	032	1A	CC	58	072	3A		90	132	5A	!	122	172	7A	:
27	033	1B	CU1	59	073	3B	CU3	91	133	5B	$	123	173	7B	#
28	034	1C	FS	60	074	3C		92	134	5C	*	124	174	7C	@
29	035	1D	GS	61	075	3D	NAK	93	135	5D	)	125	175	7D	'
30	036	1E	RS	62	076	3E		94	136	5E	;	126	176	7E	=
31	037	1F	US	63	077	3F	SUB	95	137	5F	~	127	177	7F	"

Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char
128	200	80		160	240	A0		192	300	C0	{	224	340	E0	\
129	201	81	a	161	241	A1	~	193	301	C1	A	225	341	E1	
130	202	82	b	162	242	A2	s	194	302	C2	B	226	342	E2	S
131	203	83	c	163	243	A3	t	195	303	C3	C	227	343	E3	T
132	204	84	d	164	244	A4	u	196	304	C4	D	228	344	E4	U
133	205	85	e	165	245	A5	v	197	305	C5	E	229	345	E5	V
134	206	86	f	166	246	A6	w	198	306	C6	F	230	346	E6	W
135	207	87	g	167	247	A7	x	199	307	C7	G	231	347	E7	X
136	210	88	h	168	250	A8	y	200	310	C8	H	232	350	E8	Y
137	211	89	i	169	251	A9	z	201	311	C9	I	233	351	E9	Z
138	212	8A		170	252	AA		202	312	CA		234	352	EA	
139	213	8B		171	253	AB		203	313	CB		235	353	EB	
140	214	8C		172	254	AC		204	314	CC		236	354	EC	
141	215	8D		173	255	AD	[	205	315	CD		237	355	ED	
142	216	8E		174	256	AE		206	316	CE		238	356	EE	
143	217	8F		175	257	AF		207	317	CF		239	357	EF	
144	220	90		176	260	B0		208	320	D0	}	240	360	F0	0
145	221	91	j	177	261	B1		209	321	D1	J	241	361	F1	1
146	222	92	k	178	262	B2		210	322	D2	K	242	362	F2	2
147	223	93	l	179	263	B3		211	323	D3	L	243	363	F3	3
148	224	94	m	180	264	B4		212	324	D4	M	244	364	F4	4
149	225	95	n	181	265	B5		213	325	D5	N	245	365	F5	5
150	226	96	o	182	266	B6		214	326	D6	O	246	366	F6	6
151	227	97	p	183	267	B7		215	327	D7	P	247	367	F7	7
152	230	98	q	184	270	B8		216	330	D8	Q	248	370	F8	8
153	231	99	r	185	271	B9		217	331	D9	R	249	371	F9	9
154	232	9A	^	186	272	BA		218	332	DA		250	372	FA	
155	233	9B		187	273	BB		219	333	DB		251	373	FB	
156	234	9C		188	274	BC		220	334	DC		252	374	FC	
157	235	9D		189	275	BD	]	221	335	DD		253	375	FD	
158	236	9E		190	276	BE		222	336	DE		254	376	FE	
159	237	9F		191	277	BF		223	337	DF		255	377	FF	

▸ **Note:**

APPENDIX K. COBOL RESERVED WORDS

BOLD words are ANSI 85 reserved words. Trailing letter(s) after the word indicate the following:

- i indicates an additional non-ANSI reserved word for **ANSI 74** and **ANSI 85**.
- q indicates an additional non-ANSI reserved word for the ***ISQL*** feature-set.
- v- indicates an ANSI reserved word that is NOT reserved for ***VXCOBOL***.
- v indicates an additional non-ANSI reserved word for ***VXCOBOL***.

Words without a trailing "i", "v-", "v", or "q" are reserved words for all **ICOBOL** dialects.

ABSOLUTE q
ACCEPT
ACCESS
ACCESSIBILITY v
ACTION q
ADD
ADDRESS i
ADVANCING
AFTER
ALL
ALLOCATE q
ALLOW v
ALLOWS v
ALPHABET v-
ALPHABETIC
ALPHABETIC-LOWER v-
ALPHABETIC-UPPER v-
ALPHANUMERIC v-
ALPHANUMERIC-EDITED v-
ALSO
ALTER
ALTERNATE
AND
ANY v-
APPROXIMATE v
ARE
AREA
AREAS
AS q
ASC
ASCENDING
ASCII i, v
ASSIGN
AT
AUTHOR
AUTO i, v
AUTOMATIC v
AVG q

BACKGROUND i
BACKGROUND-COLOR i
BACKWARD i, v
BECOMES v
BEEP i
BEFORE
BELL i, v
BETWEEN q
BINARY v-
BIT v, q
BLACK i
BLANK
BLINK
BLOCK
BLUE i
BOLD i
BOTTOM
BRIGHT i
BROWN i
BY

CALL
CANCEL
CD
CF
CH
CHANNEL v
CHAR q
CHARACTER
CHARACTERS
CHECK v, q
CHECKPOINT v
CLASS v-
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COL
COLLATING
COLS q
COLUMN
COLUMNS q
COMMA
COMMAND q
COMMIT v, q
COMMON v-
COMMUNICATION
COMP
COMP-1 v
COMP-2 v
COMP-3 i, v
COMP-5 i
COMPRESSION v
COMPUTATIONAL
COMPUTATIONAL-1 v
COMPUTATIONAL-2 v
COMPUTATIONAL-3 i, v
COMPUTATIONAL-5 i
COMPUTE
CONCURRENT v
CONFIGURATION
CONNECT v, q
CONNECTED v
CONNECTION q
CONTAINS
CONTENT
CONTIGUOUS v
CONTINUE
CONTROL
CONTROLS
CONVERT i, q
CONVERTING v-
COPY
CORR
CORRESPONDING
COUNT
CR v
CREATE v, q
CURRENCY
CURRENT v, q
CURRENT-DATE q
CURRENT-TIME q
CURRENT-TIMESTAMP q
CURSOR i, v
CYAN i

DATA
DATA-SENSITIVE i, v
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
DBMS v
DB-EXCEPTION v
DE
DEALLOCATE q
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DEFAULT i
DEFINE v
DELETE
DELIMITED
DELIMITER
DEPENDING
DESC
DESCENDING
DESTINATION
DETAIL
DIAGNOSTICS q
DICTIONARY v
DIM i
DISABLE
DISCONNECT v, q
DISK i, v
DISPLAY
DISTINCT q
DIVIDE
DIVISION
DOUBLE q
DOWN
DROP q
DUPLICATE v
DUPLICATES
DYNAMIC

EBCDIC i, v
ECHO i
EGI
ELSE
EMI
EMPTY v
ENABLE
END
END-ACCEPT i, v
END-ADD
END-CALL
END-CHECKPOINT v
END-COMMIT q
END-COMPUTE
END-CONNECT q
END-CREATE q
END-DEALLOCATE q
END-DEFINE v
END-DELETE
END-DISCONNECT q
END-DISPLAY i
END-DIVIDE
END-DROP q
END-EVALUATE v-
END-EXEC q
END-EXECUTE q
END-EXPUNGE v
END-FETCH q
END-FINISH-REQUEST-GROUP v
END-GET q
END-IF
END-INSERT q
END-LINK v
END-LOCK-RESOURCE v
END-MODIFY-REQUEST-GROUP v
END-MULTIPLY
END-OF-PAGE
END-PERFORM
END-PREPARE q
END-READ
END-RECEIVE v-
END-RETRIEVE v
END-RETURN
END-REWRITE
END-ROLLBACK q
END-SEARCH
END-SELECT q
END-SET q
END-START
END-START-REQUEST-GROUP v
END-STRING
END-SUBTRACT
END-UNDELETE i, v
END-UPDATE q
END-UNSTRING
END-WRITE
ENTER
ENVIRONMENT
EOL i
EOP
EOS i
EQUAL
ERASE i, v
ERROR
ESCAPE i, v
ESI
EVALUATE v-
EVEN v
EVERY
EXCEPTION
EXCLUDE v
EXCLUSIVE i, v
EXEC q
EXECUTE q
EXISTS q
EXIT
EXPIRATION v
EXPUNGE v
EXTEND
EXTERNAL

FALSE v-
FD
FEEDBACK v
FETCH q
FIELD i, v
FIELDS i, v
FILE
FILE-CONTROL
FILES v
FILESET v
FILLER
FINAL
FIND v
FINISH v
FIRST
FIX v
FIXED i, v
FLOAT q
FOOTING
FOR
FOREGROUND i
FOREGROUND-COLOR i
FORWARD i, v
FOUND q
FROM

FULL i, v
FUNCTION i, v

GENERATE
GENERATION v
GENERIC v
GET q
GIVING
GLOBAL
GO
GOBACK i, v
GREATER
GREEN i
GROUP

HAVING q
HEADER i, v
HEADING
HIERARCHICAL v
HIGH i, v
HIGH-VALUE
HIGH-VALUES
HIGHLIGHT i
HOUR q

I-O
I-O-CONTROL
ID v
IDENTIFICATION
IF
IGNORE i
IMMEDIATE i, v
IN
INDEX
INDEXED
INDICATE
INDICATOR q
INFOS v
INITIAL
INITIALIZATION v
INITIALIZE v-
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INT q
INTEGER q
INTERVAL q
INTO
INVALID
INVALIDATE v
INVERTED v
IS

JUST
JUSTIFIED

KEY
KEYBOARD i, v
KEYS v

LABEL
LABELS v
LAST
LEADING
LEFT
LENGTH
LESS
LEVELS v
LIKE q
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINK v
LINKAGE
LOCAL v
LOCK
LOCK-RESOURCE v
LOGICAL i, v
LOW i
LOW-VALUE
LOW-VALUES
LOWLIGHT i
LRU v

MAGENTA i
MANAGEMENT v
MANDATORY v
MAX q
MAXIMUM v
MEMBER v
MEMORY
MERGE
MERIT v
MESSAGE
MIN q
MINUS i
MINUTE q
MODE
MODIFY v
MODULES
MONTH q
MOVE
MULTIPLE
MULTIPLY

NAME i, v
NATIONAL q
NATIVE
NEGATIVE
NEXT
NO
NODE v
NONE v,q
NOT
NULL i,v
NULLS i
NUMBER
NUMERIC
NUMERIC-EDITED v-

OBJECT-COMPUTER
OBTAIN v
OCCURRENCE v
OCCURS
ODD v
OF
OFF
OFFSET v
OMITTED
ON
ONLY v, q
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OUT v
OUTPUT
OVERFLOW
OWNER v

PACKED-DECIMAL v-
PAD v, q
PADDING
PAGE
PAGE-COUNTER
PARITY v
PARTIAL v, q
PERFORM
PF
PH
PHYSICAL i, v
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRECISION q
PREPARE q
PREVIOUS i
PRINTER i, v
PRINTER-1 i, v
PRINTING
PRIOR v, q
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PROMPT i
PURGE v-

QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
READY v
REAL q
RECEIVE
RECONNECT v
RECORD
RECORDING i, v
RECORDS
RED i
REDEFINES
REEL
REFERENCE
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACE v-
REPLACING
REPORT
REPORTING
REPORTS
REQUEST-GROUP v
REQUIRED i, v
RERUN
RESERVE
RESERVE-KEY v
RESET
RETAIN v
RETRIEVAL v
RETRIEVE v
RETURN
REVERSE i
REVERSE-VIDEO i
REVERSED
REWIND
REWRITE
RF
RH
RIGHT

ROLLBACK v, q
ROOT v
ROUNDED
ROW q
RUN

SAME
SAVE v
SCREEN i, v
SD
SEARCH
SECOND q
SECONDS v
SECTION
SECURE i, v
SECURITY
SEEK v
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
SQL q
SQLCA q
SQLCODE q
SQLERROR q
SQLSTATE q
SQLWARNING q
STANDARD
STANDARD-1
STANDARD-2
STANDARD-3 v
START
STATIC v
STATUS
STOP
STORE v
STRING
SUB-INDEX v
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBSCHEMA v
SUBTRACT
SUM
SUPPRESS
SWITCH i, v
SYMBOLIC
SYNC
SYNCHRONIZED
SYSTEM v

TAB i
TABLE
TALLYING
TAPE
TEMPORARY v, q
TERMINAL
TERMINATE
TEST v-
TEXT
THAN
THEN
THROUGH
THRU
TIME
TIME-OUT i, v
TIMES
TIMESTAMP q
TO
TOP
TRAILER v
TRAILING
TRANSACTION
TRUE v-
TRUNCATE v
TYPE

UNDEFINED v
UNDELETE i, q
UNDERLINE i
UNDERLINED i
UNION q
UNIT
UNLOCK
UNSTRING
UNTIL
UP
UPDATE i, q
UPON
USAGE
USE
USER i, v, q
USING

VALID q
VALUE
VALUES
VARCHAR q

VARIABLE i, v
VARYING
VERIFY v
VIRTUAL v
VIRTUAL-STORAGE v
VOLUME v

WAIT v
WHEN
WHENEVER q
WHERE q
WHITE i
WITH
WITHIN v
WORDS
WORK q
WORKING-STORAGE
WRITE

YEAR q
YYYYDDD i, v
YYYYMMDD i, v

ZERO
ZEROES
ZEROS
ZONE q

+
-
*
/
**
<
<>
<=
=
>
>=

The following words are not currently reserved words in **ICOBOL** but may be used in the future or are reserved words in another manufacturer's COBOL product.

```
ABSENT
ACTIVE-CLASS
AUTOTERMINATE

B-AND
B-NOT
B-OR
B-XOR
BASED
BINARY-CHAR
BINARY-DOUBLE
BINARY-LONG
BINARY-SHORT
BINARY-SEQUENTIAL
BLINKING
BOOLEAN

C01
C02
C03
C04
C05
C06
C07
C08
C09
C10
C11
C12
CARD-PUNCH
CARD-READER
CASSETTE
CLASS-ID
COMP-4
COMP-6
COMPUTATIONAL-4
COMPUTATIONAL-6
CONSOLE
CONSTANT
CONVERSION
CRT

DISC

EXCEPTION-OBJECT

FACTORY
FILE-ID
FLOAT-EXTENDED
FLOAT-LONG
FLOAT-SHORT
FREE
FUNCTION-ID

INHERITS
INTERFACE-ID
INVOKE

LISTING
MANUAL
METHOD
METHOD-ID

NATIONAL-EDITED
NESTED

OBJECT
OPTIONS
OVERRIDE

PRINT
PRINTER-2
PRINTER-3
PROGRAM-POINTER
PROPERTY
PROTOTYPE

RAISE
RAISING
REMARKS
REPOSITORY
RESUME
RETRY
RETURN-CODE
RETURNING

SHARING
SORT-WORK
SOURCES
SWITCH-1
SWITCH-2
SWITCH-3
SWITCH-4
SWITCH-5
SWITCH-6
SWITCH-7
SWITCH-8
SYSIN
SYSOUT
SYSTEM-DEFAULT

TYPEDEF

UNIVERSAL
UPSI-0
UPSI-1
UPSI-2
UPSI-3
UPSI-4
UPSI-5
UPSI-6
UPSI-7
USER-DEFAULT

VALIDATE
```

APPENDIX L. SYSTEM CALLS

1. Overview

ICOBOL uses the CALL PROGRAM statement to access a set of system-defined routines. These are called system calls and are listed in the table below. Following the table is a description of each of the calls. If the system call provides user interaction through a menu, that interface is documented in the appropriate chapter in the Utilities Manual.

***** NOTE: New applications should avoid using these system calls and instead use the preferred IC_xx builtins described in this document. Each system call (except for #D) has a corresponding builtin, as indicated in the table.**

System Call	Function	Comparable Builtin Function
#A	Abort a program	IC_ABORT_TERM
#D	Treated as a normal CALL PROGRAM	n.a.
#H	Log off a terminal	IC_HANGUP
#L	Chain to LOGON	IC_LOGON
#M	Send a message	IC_SEND_MSG
#N	Rename a file	IC_RENAME
#O	Detach a COBOL job	IC_DETATCH_PROGRAM
#P	View and change the current status of the print spooling system, including the files in the system	IC_PRINT_STAT
#S	Shutdown the runtime system	IC_SHUTDOWN
#T	Terminal Status	IC_TERM_STATUS
#W	Pause for a period of time	IC_DELAY
##C	Compute a check block	IC_CHECK_DATA
##D	Get total and free disk space	IC_GET_DISK_SPACE
##E	Get an environment variable	IC_GET_ENV
##F	Resolve a filename or do a directory listing	IC_DIR_LIST
##G	Return a system message	IC_MSG_TEXT
##I	Show internal status information	IC_SYS_INFO
##M	Move file data	IC_MOVE_FILE_DATA
##P	Like #P but returns to program	IC_PRINT_STAT
##S	Return a serial number	IC_SERIAL_NUMBER
##T	Sets timeout value for ACCEPTs	IC_SET_TIMEOUT
##U	Shutdown the runtime system	IC SHUTDOWN

For ***VXCOBOL***, #D, #H, #L, #S, and #W are supported. #A, #M, #P, and #T will chain to logon. All other system calls will return exception 203, “Program not found”.

2. #A Abort a Program

The #A system call allows active terminals to be aborted either to facilitate a system shutdown or for other reasons. Upon invocation, a terminal status window of all logged-on terminals will be displayed. You are then prompted as to which terminal you wish to abort. Once that terminal is aborted you will see the confirmation in the status window. Aborting a terminal will not remove it from the terminal status window but will mark the terminal as `Stopped' in the terminal status window.

The #A system call is enabled with the Abort terminal privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 203 "Program not found.".

The syntax is:

```
CALL PROGRAM "#A"
```

On exit from #A, a CALL PROGRAM "#L" is performed.

On UNIX, Abort communicates with ICEXEC to interrupt the appropriate process. ICEXEC uses the UNIX Signal mechanism with SIGUSR1 to shutdown the named process. An abort is just like a *kill -16* from the shell to that process.

For more on #A see the Abort utility in the Utilities Manual.

3. #H Hang Up the Terminal

The syntax is:

```
CALL PROGRAM "#H"
```

#H terminates **ICOBOL** like #S does.

4. #L Call LOGON

The #L system call runs the standard LOGON program and makes the terminal line Inactive in the terminal status window. #L does not remove the terminal from the Terminal Status window. No ICISAM files should be open in the LOGON program since the system can and will abort users executing LOGON when entered via #L or after the initial sign-on.

The syntax is:

```
CALL PROGRAM "#L"
```

NOTE: A CALL PROGRAM "LOGON" is not the same as #L, since it will not mark the terminal as being Inactive.

5. #M Message Broadcast

The #M system call allows the user to send a message to one, several, or all logged-on **ICOBOL** users, either active or inactive on the same machine. The message will not appear on the user's console until the next opcode is executed by that process. Thus if a user is waiting in an ACCEPT the message will not appear until that ACCEPT has been terminated.

The #M system call is enabled with the Message sending privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 203 "Program not found.".

Two message modes are available.

Mode 1 (Interactive Mode)

For mode 1, the syntax is:

```
CALL PROGRAM "#M"
```

Upon invocation, a terminal status window of all logged on terminals is displayed. You are then prompted for the message that you wish to send. You are then prompted for the terminal number to send the message to. If none, the message is sent to all logged-on users.

On exit from a Mode 1 #M, a CALL PROGRAM "#L" is performed.

For more on #M in Mode 1 see the Message Broadcast utility in the Utilities Manual.

Mode 2 (Program Mode)

For mode 2, the syntax is:

```
CALL PROGRAM "#M n message"   or
CALL PROGRAM "#M * message"
```

Where

n
: Specifies the terminal number to send the message to.

*
: Implies all logged on terminals.

message
: Is the message to be sent.

If *n* is an invalid terminal number or is not currently active, an Exception Status 228 "The terminal is not logged on" is returned. If *n* is a terminal which is not enabled, Exception Status 229 "The terminal is not configured into the system" is returned.

When a Mode 2 #M is finished, execution continues with the next statement in the COBOL program.

6. #N Rename a File

The #N system call allows a file to be renamed.

The syntax is:

```
CALL PROGRAM "#N old-filename  new-filename"
```

Where

old-filename
: Is the old filename to be renamed.

new-filename
: Is the new filename.

Pathnames can be used. Separate the filenames by at least one (1) space. To rename an ICISAM file you must rename each individual portion explicitly supplying the .XD and .NX extensions with two system calls.

Execution continues at the next statement.

The #N filenames do not go through the ICLINK link file facility.

Old-filename and new-filename are processed as an External Filename as described on page 751, except a full pathname is not made if a simple name is given.

7. #O Detach a COBOL program

The #O system call enables a user to start a COBOL program on another logical console (called a detached program).

The #O system call is enabled with the Detach/Host programs privilege in the Program Environment of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 203 "Program not found.".

The syntax is:

```
CALL PROGRAM "#O program-name [ filename ]" [ USING term-id ].
```

Where

program-name
: Is a valid COBOL program name including program switches, although no spaces can separate the switches from the program.

filename
: Specifies a disk filename for the output file.

term-id
: Is at least a PIC 9(5) DISPLAY field. If set to 65535, it instructs **ICOBOL** to start the detached program on the next available detachable console and return that console number in term-id. If set to anything other than 65535, **ICOBOL** tries to detach the COBOL program attached to that specific detached console if it is available, otherwise an error is given. If no term-id is given, then the next available detachable console is used.

An available detachable console is defined to be a logical console that is:

1) enabled,
2) whose device is set to NUL (on Windows) or null (on UNIX), and
3) is currently not running a detached program.

If a detached program is started with no optional output file, then all output from the program will go to the null device (bit-bucket). That is, the program output will be discarded.

All detached programs will generate an end-of-file (EOF) error on any ACCEPT or READ from the console, as the input device will always be set to the null device.

A detached program can only execute non-screen DISPLAY statements. A screen DISPLAY will generate an error and the program will terminate.

Possible errors for #O include:

1	Invalid operation
36	Filename is not valid (for an invalid program name)
209	Parameter mismatch (for no program name specified or if term-id is invalid, i.e., greater than 65535 or not a number)
212	No more programs are available (if no available consoles can be found to detach this program to)
219	Invalid task number (if the console specified by term-id is not available or is in use.

The detached program will inherit the starting program's username. Its privileges are those specified for the console on which it is running. Detached programs cannot execute any system calls or builtins that perform screen I/O.

If a detached program terminates abnormally, any error will be written to the standard output file or to the starting program's standard error file on UNIX.

On UNIX, the program performing the #O must have icrun in the working directory or in a directory on its PATH variable.

NOTE: A standard CALL PROGRAM error like Program Not Found, Program Too Big, etc. is not returned by a #O because it occurs after the "detached program" has been detached from the current program.

8. #P and ##P Printer Control Utility

The #P and ##P system calls enable the user to view and change the current status of the print spooling system including the files in the system, the files currently queued to a print queue or printing, and the files that have been printed.

The #P and ##P system calls are enabled with the Printer control privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 203 "Program not found.".

The syntax is:

```
      CALL PROGRAM "#P"
or
      CALL PROGRAM "##P"
```

For more on #P, ##P, and the printer spooling system, see the Printer Control utility in the Utilities Manual.

On exit from #P, a CALL PROGRAM "#L" is performed.

On exit from ##P control is returned to the program that called it.

9. #S Stop Runtime System Execution

The #S and ##U system calls allow the program to terminate the runtime system.

The #S and ##U system calls are enabled with the System Shutdown privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 203 "Program not found.".

The syntax is:

```
   CALL PROGRAM "#S"
```

The #S system call terminates the currently running **ICOBOL** process and returns control to the process that invoked **ICOBOL**. If that process was login, then the terminal is logged off the system. #H and ##U behave exactly the same as #S. When **ICOBOL** has been started in Program mode (i.e., *icrun program*) then a STOP RUN or Fatal Error will behave the same as #S.

10. #T Terminal Status

The #T (Terminal Status) system call allows the user to view the status of all **ICOBOL** users on the machine as well as current system information.

The #T system call is enabled with the Terminal status privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 203 "Program not found.".

The syntax is:

```
CALL PROGRAM "#T"
```

For more on #T see the Terminal Status utility in the Utilities Manual.

11. #W Wait for a Specified Time

The #W system call suspends program execution for a time expressed in tenths of a second. The default integer is 30, which is a three-second pause. The maximum integer is 65,535, producing a pause of 109 minutes and 13.5 seconds. No CPU time is used during the pause.

The syntax is:

```
CALL PROGRAM "#W[ { space }... ]integer".
```

NOTE: If any value but spaces or digits are found after the #W, a conversion error is detected and the default value of 30 is used.

12. ##C Compute Check Block

The ##C extended system call is supported to allow programmers to easily calculate a CRC, LRC (XOR), or checksum on a block of data.

The syntax is:

```
CALL PROGRAM "##C" USING option, polynomial, length, buffer, result
```

Where

option

Is a 1-byte binary, PIC 99 COMP, that holds the calculation option. Current options are:

0 for a normal CRC using the supplied polynomial,
1 for a reverse CRC using the supplied polynomial,
2 for a LRC (XOR) 8-bit calculation, and
3 for a checksum calculation.

Adding 64 to one of the above calculation options says to use the passed in *result* as the base to start the calculation, otherwise zero is used.

polynomial

Is a 2-byte binary, PIC 9(4) COMP, that holds the binary value for the CRC generator polynomial.

length

Is a 2-byte, PIC 9(4) DISPLAY, which holds the length of data in the buffer on which to perform the calculation. This cannot be larger than buffer.

buffer

Is a PIC X(n) that holds the data on which the check is to be calculated.

result

Is a 2-byte binary, PIC 9(4) COMP, that holds the calculated value as a binary number.

Some common crc polynomials are:

CRC-CCITT	1021h or 4129 (base 10)
CRC-16	8005h or 32773
reverse CRC-CCITT	8408h or 33800
reverse CRC-16	A001h or 40961

The CRC-CCITT polynomial is used for XMODEM-CRC protocol.

For example, calculation option 64 would be used to calculate a CRC on a block (or file) that is larger than the buffer by making repeated calls.

13. ##D Get Disk Space Information

The ##D extended system call is supported to allow system developers to access the total disk space allowed and the current amount free in bytes for a disk drive (on Windows) or for a mounted file system (on UNIX).

The syntax is:

```
CALL PROGRAM "##D" USING location, space
```

Where

location

An alphanumeric item of at least the length to hold the two-character drive name (on Windows), the mounted filesystem name (on UNIX) (/, /usr, etc.), or spaces for the current drive name or filesystem you wish to check.

space

A structure composed of two PIC 9(10) items into which **ICOBOL** returns the total amount of storage (in bytes) for the given filesystem and the number of bytes currently free. The structure could look like:

```
01 DISK-SPACE.
   02 TOTAL-BYTES     PIC 9(10).
   02 FREE-BYTES      PIC 9(10).
```

If you wish to determine the default filesystem, the ##F call should be used.

Location is processed as an External Filename as described on page 751.

14. ##E Get Environment Variable

The ##E extended system call is supported to allow programmers to read the value of an environment parameter that was set before **ICOBOL** started.

The syntax is:

```
CALL PROGRAM "##E" USING name-argument, return-argument
```

Where

name-argument

Is a PIC X(n) that holds the name of the environment variable to be read

return-argument

Is a PIC X(n) to which **ICOBOL** moves the value of the environment variable according to the rules for MOVE.

If the environment variable named by name-argument cannot be found either in the environment or as configured in the environment strings section of the configuration file (.cfi), an error is generated and the ON EXCEPTION clause, if present, is executed.

If the return-argument for the environment variable is too small to hold the full value, the value is moved and truncated and an error is generated and the ON EXCEPTION clause, if present, is executed.

15. ##F Filename Resolution or Directory Lookup

The ##F extended system call is supported to allow system developers to:

1) resolve a simple or relative filename into a full pathname and check to see if the filename exists or
2) do a directory lookup on a given template to get the number of matches along with an optional list in a sequential file for each file entry along with its file information (filename, filesize, attributes, last-modified).

Option 1 Filename Resolution

For option 1, the syntax is:

```
CALL PROGRAM "##F" USING argument-1
```

Where

argument-1

Is an alphanumeric item of at least 64 characters. Argument-1 should be set to the filename or template you wish to resolve or check for existence before making the ##F call. If the filename does not exist, or the file is a symbolic link for which the resolution does not exist, or no files match the template the Exception Status will be set to 2 (File not found) and the ON EXCEPTION clause, if given, is executed.

Argument-1 is returned as a fully resolved filename or template including all directory specifiers that can be up to 64 characters.

To get the default directory, argument-1 should be set to spaces; and the fully resolved pathname is returned.

On Windows, to get the default directory for any disk drive, set argument-1 to just the disk drive itself; i.e., `C:', `D:', etc.; and the fully resolved pathname will be returned. When running in network mode, network files will be resolved to their fully qualified network name. For example if drive D is redirected to the server \\386MAINSERVER's E drive, the file `D:\PROGRAMS\FILE' would resolve to the fully qualified name of `\\386MAINSERVER\E\PROGRAMS\FILE'.

When no template is given, argument-1 is processed as an External Filename as described on page 751.

Option 2 Directory Lookup

For option 2, the syntax is:

```
CALL PROGRAM "##F" USING argument-1, argument-2 [, argument-3 ]
```

Where

argument-1

Is the same as for option 1 but normally will contain a template.

argument-2

Is a numeric item of at least PIC 9(5) that returns the number of file entries found by the template specified in argument-2. If this number is zero, no files were found that matched the template provided by argument-1. In addition, the Exception Status is set and the ON EXCEPTION clause, if given, is executed.

argument-3

Is optional and specifies the filename of the file to which each of the file entries that match the template given by argument-1 are written. Argument-3 should be an alphanumeric item at least large enough to hold the filename to be given.

Argument-1, when not a template, and argument-3 are processed as an External Filename as described on page 751.

The file entries are written to the file as printer-records, with each entry terminated with a line-terminator. If the count returned is greater than zero, the filename given by argument-3 is created if there is no file by that name already, and is deleted and re-created if it already exists. If the count returned in argument-2 is zero, the file given in argument-3 is not touched.

To read the file created above you should have an ASSIGN TO KEYBOARD in your COBOL program and read with at least 64-byte records to get each file entry. Each filename entry will be a single line of characters terminated by a line-terminator. Each entry can be defined as such:

```
01 FILE-ENTRY.
   02 MODIFIED-INFO.
      03 DATE-MODIFIED        PIC 9(6).
      03 TIME-MODIFIED        PIC 9(8).
   02 ACCESSED-INFO.
      03 DATE-ACCESSED        PIC 9(6).
      03 TIME-ACCESSED        PIC 9(8).
   02 FILESIZE-BYTES          PIC 9(10).
   02 F-ATTRIBUTES            PIC X(8).
   02 F-ATTRIBUTE-RED  REDEFINES F-ATTRIBUTES.
      03 READABLE-ON          PIC X(1).
      03 WRITABLE-ON          PIC X(1).
      03 PROTECTABLE-ON       PIC X(1).
      03 ARCHIVE-IT           PIC X(1).
      03 DIRECTORY-TYPE       PIC X(1).
      03 SYSTEM-TYPE          PIC X(1).
      03 EXECUTABLE-TYPE      PIC X(1).
      03 FILLER               PIC X(1).
   02 FILENAME                PIC X(64).
```

The date and time fields are exactly like ACCEPT FROM DATE (YYMMDD) and TIME (HHMMSShh) respectively.

On Windows

The Accessed Info (ACCESSED-INFO) field is not available and will always be set to zeros(0).

The file attributes (F-ATTRIBUTES) will have an `R' in position 1 if the file can be read, a `W' in position 2 if the file can be written to, a `P' in position 3 if it cannot be deleted, and an `A' in position 4 if it has been modified since it was archived. Position 5 will be set to `D' if the file entry is a directory. Position 6 will be set to `S' if the file is a system file. Otherwise the positions will be set to space.

The Read-only attribute will force position 2 (WRITABLE-ON) to be set to space and position 3 (PROTECTABLE-ON) to be set to P. The archive, directory, and system attributes match the os attributes. The executable-type (X) will be set for directories, .EXE, .COM, and .BAT files. The filename argument will never be more than 12 characters.

On UNIX

Positions 1, 2, 3, and 7 of the file attributes are based on how the current user within his group can access each particular file. For files that are symbolic links, the file attributes returned are for the resolution file, not the symbolic link itself.

The file attributes (F-ATTRIBUTES) will have an `R' in position 1 if the file can be read by the current user, a `W' in position 2 if the file can be written by the current user, and a `P' in position 3 if the file cannot be deleted by the current user (i.e., the current user does not have Write access to the directory). Position 4 will always return a space. Position 5 will be set to `D' if the file entry is a directory. Position 6 will be set to `S' if the file is a special character device, block device, or other special files. Position 7 will be set to `X' if the file is executable. Otherwise the positions will be set to space.

If a simple filename is ever to exceed 64 characters then this structure should be increased to match the largest size a simple name could be. **ICOBOL** will only write as much as needed to the temporary file.

16. ##G Return a System Message

The ##G extended system call is supported to allow system developers to pick up any of the system messages from the message file (system.ms). These messages map one-to-one to the Exception Status received in ACCEPT FROM EXCEPTION STATUS.

The syntax is:

```
CALL PROGRAM "##G" USING exc-code, return-argument
```

Where

exc-code
Specifies the numeric code (PIC 9(3)) of the message to be given between 0 and the maximum Exception Status value.

return-argument
Specifies a character string of at least 60 characters. The system call will return with *return-argument* holding the corresponding message for the numeric code in *exc-code*.

You can actually get back any message from the message file, not just those that map to Exception Status value, provided you know the correct value.

17. ##I Internal Status Information

The ##I extended system call is supported in **ICOBOL** to allow internal status information for the entire **ICOBOL** system to be viewed.

The ##I extended system call is enabled with the System Information privilege in the Program Environment configuration of the configuration file (.cfi). If not enabled, the call will fail with an Exception Status 203 "Program not found.".

The syntax is:

```
CALL PROGRAM "##I"
```

For Internal Status, **ICOBOL** provides a screen of statistical information about various **ICOBOL** parameters. For the named resource, three numbers are displayed. These are:

```
In Use          is the number currently in use
MaxUsed         is the most this has ever been, for this invocation
Max             is the maximum number configured
```

The MaxUsed values can be used to either raise or lower individual System Parameters in the configuration file (.cfi), the CONFIG.SYS file (on Windows), or in the UNIX Kernel (on UNIX) to provide a better tuned system.

On exit from ##I, control is returned to the program that called it.

18. ##M Move File data

The ##M extended system call is supported in **ICOBOL** to allow files to be copied from one place to another with various options.

The syntax is:

```
CALL PROGRAM "##M" USING option, source-name, destination-name [, count
   [, start-src-pos [, start-dst-pos ]]]
```

Where

option

Specifies a 1-byte binary, PIC 99 COMP, composed of the following bits.

Option-bit	Meaning
1	Don't erase destination if it exists
2	Write at eof (ignore start-dst-pos)
4	The destination file must exist
8	The destination file must NOT exist

Below are the useful combinations of the above option-bits.

Option	Destination file		Destination Position
	Does NOT exist	Exists	
0	create	erase	as specified
1	create	don't erase	as specified
3	create	don't erase	at eof
4	ERROR	erase	as specified
5	ERROR	don't erase	as specified
7	ERROR	don't erase	at eof
8	create	ERROR	as specified

source-name

Specifies a PIC X(n) and holds the source filename to be copied.

destination-name

Specifies a PIC X(n) and holds the destination filename to be copied. It cannot be a directory.

count

Specifies a PIC 9(n) DISPLAY, and holds an optional count for how many bytes to copy from source or until EOF. If given, the number of bytes actually copied is returned.

start-src-pos

Specifies a PIC 9(n) DISPLAY, and holds an optional byte offset in the source file from which to start copying. I.E., a start-src-pos of 0 is the beginning of file.

start-dst-pos

Specifies a PIC 9(n) DISPLAY, and holds an optional byte offset in the destination file to which copying should start. If not given, the beginning of file is used.

The source file must exist and be able to be opened for binary input.

Unless you use the appropriate option, this call will allow a file to be copied upon itself with possible unintended results.

Source-name and destination-name are processed as an External Filename as described on page 751.

19. ##P Print Spooling

The ##P extended call is the same as the #P system call with the exception that upon exiting from the Printer Control Utility it returns to the calling program, just like a CALL. See #P for more information.

20. ##S Return the System Serial Number

The ##S extended system call is supported to allow system developers to check for a unique serial number. This call will return the unique runtime license serial number as provided by the license manager (ICPERMIT) from the license description file.

The syntax is:

```
CALL PROGRAM "##S" USING argument
```

Where

argument

Should be declared as PIC 9(10).

21. ##T Set Timeout Value

The ##T extended system call allows the system developer to enable and disable timeouts for ACCEPT statements and STOP literal statements.

The syntax is:

```
CALL PROGRAM "##T" [ USING timeout ]
```

Where

timeout

Specifies a numeric value (PIC 9(5)). The values 0 through 63000 set a timeout in tenths of seconds, a 65535 is interpreted to wait forever, a 65534 says to default to the value specified as the global timeout (ICTIMEOUT), while a number between 63000 and 65534 will set the value to 63000. This value represents the time allowed between keystrokes before the system will timeout and terminate the operation. Setting a 0 essentially only reads the input buffer.

If no argument is specified, wait forever is set. The timeout value remains in effect whenever this program is active. I.E., if a CALL statement is made, while in the new program the timeout is reset to that specified by the global timeout (ICTIMEOUT) for the new program. Upon returning to the calling program, the timeout is restored to be the value that was set before the CALL.

When an ACCEPT statement times out, ESCAPE KEY is set to 99 and no data is moved to the particular item (just as when an ESC key is pressed).

If an invalid argument string is given, an Exception Status 209 is returned and the timeout value is not changed.

Where possible, the TIME-OUT AFTER clause on the ACCEPT statement should be used rather than the ##T system call.

22. ##U Unconditional Shutdown

The #S and ##U system calls are enabled with the System Shutdown privilege in the Program Environment configuration of the configuration file (.cfi).

The syntax is:

```
CALL PROGRAM "##U".
```

The ##U extended system call acts just like #S and terminates the **ICOBOL** process.

23. System Call Errors

For those system calls that are "NOT SUPPORTED" under **ICOBOL** for whatever reason, the runtime system will give an Exception Status 203 "Program not found", and the ON EXCEPTION clause, if any, is executed. This is also true if the console issuing the system call is not privileged to do so.

INDEX

. PICTURE. 177, 179
.BAT. 497, 702, 859
.CF. 695
.CFI. 293, 403, 498, 509, 515, 528, 536, 544, 548, 556, 561-565, 669, 695, 747-750, 852, 854-857, 860, 862
.CL. 695
.CX. . 512, 514, 695, 699, 700, 702, 706, 707, 741, 743, 744
.FA. 695
.LG. 693, 695, 699, 741, 743, 783
.LGB. 693, 695
.LK. 695
.NX. 263, 267, 270, 271, 551, 695, 761, 762, 821, 824, 826, 828, 829, 835, 836, 853
.PQ. 544, 695, 747
.profile. 702
.PT. 695
.PTI. 690, 691, 695
.SY. 695, 701, 706, 709, 721
.TD. 695
.TDI. 294, 522, 690, 691, 695, 768
.XD. 263, 267, 270, 551, 695, 761, 762, 821, 824, 826, 828, 829, 835-837, 853
.XDB. . . . 695, 701, 706, 709, 717, 718, 771-773, 775, 780
.XDT. 695, 701, 706, 709, 717-719, 771-775
.XL. 695, 829, 837
.XLG. 695
, PICTURE. 177, 178
?CBADDR. 30, 493, 494
?CBBADDR. 30, 493, 495
{ }. 33, 34, 692
+ PICTURE. 177
- PICTURE. 177
* PICTURE. 177, 179
/ PICTURE. 177, 178
/dev. 750
<000>. 419, 426
<cr>. 487, 488, 554
<ff>. 487, 488
<lf>. 487, 488
<nl>. 282, 305, 487, 488, 554
0 PICTURE. 177, 178
01 level. . . . 63, 120, 148, 168, 173, 183, 185, 200, 208, 209, 219, 220, 273, 277, 494, 703, 714
4GB. 519, 521, 817, 830, 837
66 level. 168, 185, 701, 709
77 level. . . . 63, 118, 165, 173, 174, 188, 199, 277, 494, 699, 703, 805, 813
88 level. . . . 63, 118, 168, 173, 174, 189, 192, 199, 712, 805, 817
9 PICTURE. 177, 178
A PICTURE. 176, 177
Abort Terminal. 498, 500, 528, 564, 852
Abort terminal privilege. 498, 500, 528, 564, 852
ACCEPT statement. . 36, 209, 210, 216, 223, 224, 251, 279, 282-289, 291, 293, 338, 339, 559, 749, 813, 859, 860, 862
ACCESS MODE clause. 90, 91, 258, 458, 459
ADD statement. 247, 297
AIX. 6, 292
Alignment rules. 119, 195
ALPHABETIC. . 40-42, 78, 79, 85, 86, 88, 93, 94, 110, 111, 118, 119, 172, 176, 177, 179, 181, 196, 236, 237, 282, 285, 377, 381, 393-395, 475, 476, 579, 581, 583, 616, 618, 626, 635-637, 644, 661, 695, 701, 717, 719, 775-778, 785, 797, 798, 805, 847
ALTERNATE RECORD KEY clause. . 93, 94, 96, 257, 418, 460, 490
ANSI COBOL 74. . . . . 699, 712-714
ANSI COBOL 85. . . . . 2, 69, 709, 712-714
ANSI switch. 401, 763
AOS/VS. . . 6, 26, 30, 35, 103, 105, 108, 175, 271, 496, 669, 703, 755, 756, 777
APPEND. 259, 693, 699, 741, 743, 754-756
Area A. 52, 59-63, 66, 67, 69, 799
Area B. 59-62, 66, 799
ASCENDING phrase. 94, 111, 390, 391, 452, 453
ASCII. . . 44, 45, 79, 108, 109, 145, 146, 152, 190, 193, 194, 391, 454, 488, 507, 516, 592, 731, 771, 797, 799, 844, 847
ASSIGN clause. 88, 98, 153, 706
ASSIGN TO DISK. 282, 341, 458
ASSIGN TO DISPLAY. 146, 341, 487
ASSIGN TO PRINTER. . 146, 341, 403, 484, 487, 757, 758
AT END. . . 250-252, 255, 261, 264, 266-268, 271-273, 391, 400, 401, 414, 415, 417, 420-423, 431, 439-441, 454, 479, 700, 706, 798, 803, 821, 823, 825
audit file. 28, 693, 727
Audit switch. 693
AUTO clause. 210, 712
AUTOEXEC.BAT. 702
B PICTURE. 176, 177
BACKGROUND. 28, 29, 208, 209, 211, 286, 339, 847
BACKGROUND-COLOR. 28, 208, 209, 211, 339, 847
backslash. 752
Bad code switch. 714
BELL clause. 212
big file. 521
big-endian. 190, 721
BINARY SEQUENTIAL. 106, 109
BLANK LINE. . . . 61, 62, 65, 208, 213, 219, 339, 704, 707, 710
BLANK SCREEN. 206-208, 213, 220, 339

BLANK WHEN ZERO clause. 118, 169
BLOCK CONTAINS clause. 144, 147
BOLD. . . . 202, 205, 207, 209, 214, 218, 228, 288, 340, 772, 847
BRIGHT. 214, 228, 288, 340, 672, 676-679, 847
BTRIEVE. 760, 765, 766, 838
buffers. . . 113, 310, 435, 485, 563, 683, 757, 828, 831, 836, 838
builtins. . . 25, 27-30, 55, 289, 291, 294, 295, 300, 463, 493, 494, 498-500, 502-507, 509-522, 525-534, 536, 541, 542, 545, 549-552, 554-562, 564, 565, 567-570, 572, 573, 575, 577, 578, 681, 729, 747, 749, 751, 753-756, 768, 769, 851, 854
C-ISAM. 760, 767
CALL PROGRAM statement. 133, 199, 251, 277, 303, 304, 306, 753, 754, 798, 851
CALL statement. . 55, 56, 189, 199, 251, 254, 277, 299-301, 306, 307, 357, 371, 493, 559, 753, 754, 798, 812, 817, 862
CANCEL statement. 307, 357, 371, 753, 754
Card Format. 59, 61, 695, 704, 794, 795
Case switch. 702
CGI. 832
cgiCOBOL. 293, 690, 691
character set. . 39, 44, 46, 52, 61, 67, 69, 71, 73, 74, 76-79, 88, 117, 145, 146, 176-178, 259, 476, 579, 630, 634, 661, 669, 674, 797-800, 806, 811
character-string. . . 33, 39, 40, 43-45, 47, 52, 61, 64, 78, 80, 166, 176-182, 186, 187, 189, 195, 203, 205, 223, 225, 381, 465, 475, 798, 799, 805, 808, 810, 813
checksum. 502, 800, 805, 829, 836, 856
class. . 40, 41, 48-52, 71, 73-75, 78, 118, 129-134, 171, 195, 232-237, 250, 290, 291, 311, 316, 319, 334, 347, 348, 352, 353, 363, 364, 381, 413, 419, 426, 437, 449, 488, 579, 584-586, 588-590, 593, 603, 604, 606, 608, 611, 612, 614-616, 618, 620, 622, 624, 626, 632, 635-637, 639, 642-650, 657, 661, 662, 713, 734, 798, 811, 847, 849
CLI. 30, 493, 497, 730, 786, 832, 839
CLOSE statement. . . 262, 266, 307, 309, 310, 391, 392, 401, 419, 425, 455, 463, 717, 750, 751, 764, 802-804, 807, 821, 825
CMD.EXE. 302
COBOL character set. . . . 39, 52, 78, 79, 117, 176, 798, 800
CODE-SET clause. 77, 145, 146, 186
collating sequence. . 47, 71, 73, 74, 77-81, 88, 90, 234, 256, 259, 389-391, 415, 422, 451, 453, 454, 459, 461, 489, 581, 592, 635, 797, 799, 803, 806, 811
color. 28, 208, 209, 211, 281, 286, 336, 339, 847
color-name. 211
COLUMN clause. . . 213, 217, 219-222, 228, 289, 341, 717
COLUMN phrase. 220, 283, 286, 338, 340, 813
comment line. 39, 40, 61, 62, 65, 67, 73, 115, 148, 704, 799, 808, 812
comment-entry. . . . 33, 39, 40, 52, 64, 67, 69, 798, 799
compatibility mode. 714
compress mode. 28, 341
COMPUTATIONAL. . . . 189-191, 193, 194, 322, 337, 430, 494-496, 498, 500, 502, 508, 509, 511, 512, 514, 517, 519, 521-523, 527, 528, 531, 532, 536-539, 541, 543-546, 553, 556, 559, 562-565, 577, 590, 607, 613, 617, 670, 672-680, 684-686, 700, 701, 704, 707, 713, 714, 717, 719, 763, 776, 777, 785, 786, 790, 847, 849, 856, 861
COMPUTATIONAL-3. . . 189-191, 337, 719, 776, 785, 847
COMPUTATIONAL-5. . . 189-191, 517, 527, 532, 719, 776, 785, 847
COMPUTE statement. 313, 713
computer's character set. . 44, 46, 52, 61, 67, 69, 78, 79, 177, 178, 476, 661, 797, 799, 806
COMSPEC. 497
condition. . . . 41, 71, 75, 77, 78, 88, 118, 122, 123, 125, 126, 130-132, 136, 153, 154, 160, 167, 168, 173, 183, 189, 191, 194-196, 199, 232, 233, 235-246, 250, 251, 260-262, 264-273, 276, 277, 301, 304, 326, 327, 348, 349, 362, 373, 374, 388, 391, 400-402, 404-406, 408-411, 415-418, 422-424, 430, 431, 435, 436, 439, 440, 443, 445, 453, 454, 459, 461, 469, 472, 476, 477, 479, 480, 485-487, 489, 490, 580, 666, 710, 717, 757, 797-800, 802-806, 808-812, 817, 821, 823, 825
condition-name. 41, 71, 75, 77, 78, 118, 122, 123, 125, 126, 136, 167, 168, 173, 183, 195, 196, 199, 233, 238-240, 242-245, 277, 404-406, 408-411, 439, 440, 443, 445, 717, 799, 800, 803, 808, 811
conditional expression. . . 232, 240, 349, 406, 439, 799, 802
conditional statement. . . . 246, 250-252, 254, 271, 272, 299, 300, 304, 311, 316, 318, 319, 334, 352, 353, 364, 366, 367, 373, 413, 417, 423, 431, 437, 449, 466, 477, 799, 803
config. 563, 781, 860
CONFIG.SYS. 563, 860
configuration file. . . 293, 403, 498, 509, 515, 528, 536, 544, 548, 556, 561-565, 669, 695, 747-750, 830, 837, 841, 842, 852, 854-857, 860, 862
configure. 25, 768, 780
console interrupt. . . . 500, 501, 513, 515, 564, 725, 727, 736, 748, 758, 764, 828-830, 836, 837
Console interrupt privilege. 500, 501, 515
continuation line. 61, 66
CONTINUE statement. 318
Control Panel. 779
CONVERT. 29, 67, 280, 287, 289, 292, 295, 335, 340, 341, 692, 754-756, 760, 847
COPY file. . . 29, 149, 531, 702-705, 716, 723, 793, 817

COPY Path switch. 703
COPY statement. . . . 57, 62, 64-66, 252, 253, 705, 799, 811
CORRESPONDING phrase. . . 246, 247, 298, 393, 416, 423, 432, 468
CR PICTURE. 176, 177
CRC. 502, 800, 827, 835, 856, 857
cross reference. 700, 716, 717, 793
Ctrl-\. 764
Ctrl-C. 725, 727
Ctrl-Del. 764
Ctrl-R. 768
CURRENCY. 78, 79, 179, 800
Currency PICTURE. 177, 179
CX file. 699, 706, 707, 743
Data Division. . 35, 41, 53, 57-59, 62, 63, 88, 103, 105, 107, 117, 136-138, 165, 174, 196, 198-200, 235, 389, 390, 406, 431, 434, 444, 452, 484, 716, 800-803, 805, 806, 808-813
DATA RECORDS clause. 148
data-name. . 41, 52, 53, 84-87, 90, 91, 95-97, 103, 105, 110-112, 117, 120, 123, 125-127, 139-143, 148, 153-155, 158, 165-168, 170, 171, 174, 175, 183-185, 199, 230, 246, 277, 300, 304, 389-391, 421, 424, 426, 439, 440, 451-453, 459, 460, 471, 487, 717, 800, 801, 803, 808, 812
DATAFILE. 749, 772, 778
data-sensitive. 106, 108, 163
DB PICTURE. 176, 177
DCD. 750, 829, 836
Debug switch. 706, 709
debugging. 62, 65, 66, 73, 338, 500, 565, 701, 704, 709, 721-723, 743, 791, 801, 847
debugging line. 62, 65, 66, 73, 704, 801
DEBUGGING MODE. 73
decimal. . . 45, 46, 52, 71, 73-75, 78, 80, 103, 105, 117, 119, 177-182, 189-191, 193, 194, 227, 245, 249, 271, 284, 289, 340, 344, 394, 397, 439, 521, 525, 534, 579, 630, 632, 634, 704, 731, 735, 744, 777, 790, 797-799, 801, 804, 806, 811, 845, 847, 848
Declaratives. . . 53, 57, 58, 63, 103, 105, 149, 229, 230, 260, 261, 272, 362, 369, 388, 389, 406, 430, 452, 479, 481, 700, 704, 722, 801, 847
DELETE FILE statement. 331, 717, 755, 756, 764
DELETE statement. 102, 259, 274, 325-329
delete-is-physical. 102, 327, 759-761
descending. . . . 77-79, 94, 95, 111, 175, 258, 259, 390, 391, 452, 453, 761, 801, 847
DESCENDING phrase. . . . 94, 111, 390, 391, 452, 453
Detach/Host program privilege. 500
detached program. 293, 509, 510, 854, 855
DG terminal. 341, 487, 684
DG/UX. 6
DIM. 209, 214, 218, 228, 288, 340, 341, 556, 674, 844, 847
DISPLAY statement. . 28, 208, 210, 212, 213, 217-219, 223, 228, 283, 284, 289, 335-341, 487, 509, 854
DIVIDE statement. 343, 344, 713
drawlines. 669
DUPLICATES phrase. 95-97, 257, 453, 490, 704
ELSE phrase. 373
environment. . 25, 28, 29, 41, 42, 53, 57-59, 62, 71-73, 80, 83, 135, 136, 159, 236, 238, 282, 290-293, 315, 455, 484, 497, 498, 500, 509, 520, 521, 528, 536, 539, 544, 554-556, 558, 561, 562, 564, 565, 575, 669, 670, 689, 691, 694, 696, 697, 699, 702, 712, 716, 722, 741, 743, 747, 749, 751, 771, 782, 783, 790, 793, 797, 798, 800-804, 806, 807, 810-813, 819, 827, 835, 847, 851, 852, 854-858, 860, 862
Environment Division. . . 41, 53, 57-59, 62, 71-73, 80, 83, 136, 236, 238, 282, 484, 716, 797, 798, 800-804, 806, 807, 810-812
environment variable. . . . 315, 497, 520, 521, 539, 544, 554, 555, 558, 670, 689, 691, 694, 696, 697, 702, 741, 743, 749, 782, 851, 857, 858
ERASE EOL. 208, 217, 287, 339, 340, 844
ERASE EOS. 208, 217, 287, 288, 339, 340
ERASE LINE. 208, 217, 287, 339, 340, 704
Error File switch. 703
ERRORLEVEL. 696
ESC. 282, 284-286, 288, 294, 559, 573, 678-680, 730, 733-735, 738, 768, 819, 844, 846, 862
ESCAPE KEY. 210, 284-286, 288-291, 293, 294, 523, 559, 679, 734, 819, 862
Exception Status. 29, 56, 103, 105, 149, 171, 260, 269, 271, 273, 290, 291, 294, 300-302, 304, 419, 426, 481, 493, 497-500, 505-507, 509, 511, 516, 517, 522, 524, 527-529, 532, 533, 535, 536, 544, 548-550, 553, 556, 560-562, 564, 565, 569, 570, 573, 577, 671, 675, 678, 679, 684-686, 724, 725, 727, 730, 733, 734, 740, 747, 748, 750, 757, 759, 766, 767, 827, 835, 843, 852-856, 858, 860, 862
exclusive. . . 3, 4, 78, 179-182, 399, 403, 418, 424, 691, 703, 705, 749, 750, 847
exit code. 302, 463, 497, 524, 561, 693, 696, 702
EXIT PROGRAM statement. . . 55, 254, 300, 306, 307, 357, 411
EXIT statement. 355, 407
exponentiation. 231, 232, 246, 797
export. 702
extended device open. 750, 751, 757, 758
extended disk open. 403, 757, 759
extended indexed open. 760
extended open options. . . 289, 419, 463, 488, 755-757, 765
extended PCQ open. 757, 758
extended relative open. 759
extension to ANSI COBOL. . . . 93, 102, 104, 110, 162, 163, 200, 260, 262, 265, 273, 279, 290, 303,

331, 335, 359, 403, 414, 433, 469, 471, 473, 483
external filename. . . . 99, 100, 299, 303, 331, 531, 551, 718, 751, 854, 857, 858, 861
Fatal. 103, 105, 260, 261, 294, 486, 696, 700, 705, 709-711, 725, 748, 768, 830, 833, 837, 840, 855
FAX. 765, 877, 879, 881, 883
feature-set. . . 35-37, 133, 192, 226, 253, 290, 291, 704, 847
FILE
OPTIONAL. 402
file associations. 794
file attribute file. 695
File Status. . . . 35, 84-87, 103, 105, 133, 149, 260, 261, 270-273, 294, 328, 329, 362, 388, 403, 419, 426, 430, 703, 717, 725, 734, 747, 750, 757, 759, 764, 766, 803, 821, 823, 825, 827, 835, 842
FILE STATUS clause. 103, 260, 717, 803
file transfer. 758
file-name. . . . 41, 53-55, 72, 83-88, 90, 93, 96, 110, 111, 114, 115, 125, 138-143, 145, 154, 256, 266, 272, 309, 310, 321, 322, 325-328, 331, 359, 361, 362, 387-392, 402, 414-418, 420-424, 426, 427, 429-432, 434-436, 451-455, 457-461, 469, 471-473, 479, 480, 485, 486, 489-491, 509, 537, 541, 543, 545-548, 717, 766, 767, 772, 773, 802, 803, 807, 811
FILLER clause. 143, 165, 168, 170, 184, 376
filter. 536-539, 541, 544-548
filtering. 536, 539, 544, 547, 548
FIRST. 40, 42, 53-56, 58, 60-62, 65-67, 76, 88, 91, 94, 100, 111, 112, 120, 124, 126, 128, 136, 149, 154, 156, 160, 170, 173, 174, 180-182, 185, 190, 219, 220, 229, 231-234, 240, 254, 256, 261, 268, 271, 274, 275, 282-286, 300, 304, 305, 322, 328, 338, 340, 350, 351, 361-363, 365, 366, 373, 376, 380-385, 387, 388, 390-393, 400-403, 407, 411, 415, 416, 418, 422, 424-426, 430, 436, 439-441, 445, 452-455, 458-461, 472, 477, 480, 486, 487, 489-491, 493, 541, 542, 545, 546, 560, 566, 567, 572, 641, 655, 657, 659, 674, 678, 679, 689, 693, 695, 701, 714, 717, 718, 724-726, 729, 731, 732, 734, 736, 737, 744, 748, 752-755, 760-762, 764, 768, 781, 790, 793, 801, 804, 805, 807, 810, 819, 827, 840, 841, 847
fixed insertion. 179, 180
fixed length record. 256
fixed length records. . 54, 158, 159, 162, 257, 258, 389-392, 452, 453, 455, 803
floating insertion. 179-182
Flow Control
Hardware Output. 758
Software Input. 758
Software Output. 758
For ANSI 74. . . . . 383, 399, 400, 402
For ANSI 74 and ANSI 85. . 88-90, 100, 166, 208, 209, 247, 256, 257, 281, 282, 284, 331, 336, 337, 359, 369, 390, 403, 410, 434, 452, 497, 517, 527, 530, 532, 568, 817, 847
For ANSI 85. . . . . 88, 399, 402, 417, 827
For ANSI 85 and VXCOBOL. 88, 383, 704
For VXCOBOL. 88, 100, 106, 152, 158, 171, 174, 256, 258-260, 271, 272, 282, 284, 295, 331, 390, 399, 402, 406, 434, 436, 439, 452, 453, 458, 486, 497, 704, 851
FOREGROUND. 28, 29, 208, 209, 211, 286, 339, 847
FOREGROUND-COLOR. . . . 208, 209, 211, 339, 847
FormPrint. 25, 691
forwardslash. 752
forward-only-cursor. 363
Free-form format. 59, 60, 62, 794, 795
FROM clause. 208, 209, 218, 223, 282, 712, 717
FULL. . . . 203, 205-207, 210, 262, 263, 265, 267, 269, 270, 285, 291, 493, 496, 499, 518, 543, 545, 551, 552, 670, 703, 726, 738, 766, 777, 786, 798, 800, 808, 811, 821, 823-828, 831, 835, 838, 840, 842, 848, 854, 858
function keys. . 282, 284-286, 294, 679, 680, 684, 768, 819
General switch. 27, 44, 700, 701, 704
generic. . 26, 33, 58, 112, 275, 692, 726, 749, 761, 766, 832, 838, 839, 848
global timeout. 289, 463, 523, 559, 831, 837, 862
GMT. 697, 698, 800
GO TO statement. 57, 369, 441, 479, 712, 714
GOBACK statement. 28, 371
Greenwich mean time. 697, 698
GUI. 25, 570, 572, 575, 577, 691, 793
hard links. 403, 693
help directory. 690, 696
Help switch. 693, 694
HIGH-VALUE. 47, 78, 79, 714, 848
HIGHLIGHT. 214, 228, 288, 340, 848
HP-UX. 26
HTML. 800, 805
hyphen. 40, 50, 61, 66, 67, 69, 692, 701, 707, 718, 751, 798
I-O Status. 103, 161, 260, 261, 264, 267-269, 271-274, 309, 326, 327, 331, 359, 402, 415, 417-419, 421, 423-426, 434-436, 459, 469, 471, 473, 485, 486, 489, 490, 803
IC_ABORT_TERM. 493, 498, 528, 851
IC_CHANGE_DIR. 493, 499
IC_CHANGE_PRIV. 493, 500
IC_CHECK_DATA. 493, 502, 851
IC_COMPRESS_OFF. 28, 493, 503
IC_COMPRESS_ON. 28, 493, 504
IC_CREATE_DIR. 493, 505
IC_CURRENT_DIR. 493, 506
IC_DECODE_URL. 493, 507, 516
IC_DELAY. 493, 508, 851

IC_DETACH_PROGRAM. . . 493, 509, 510, 749, 753, 754, 756
IC_DIR_LIST. 493, 511, 751, 755, 756, 851
IC_DISABLE_HOTKEY. 493, 512, 514, 768
IC_DISABLE_INTS. 493, 513, 515
IC_ENABLE_HOTKEY. 493, 512, 514, 768
IC_ENABLE_INTS. 493, 513, 515
IC_EXTRACT_STRING. 493, 517
IC_FULL_DATE. 291, 493, 518
IC_GET_DISK_SPACE. 493, 519, 751, 755, 756, 851
IC_GET_ENV. 493, 520, 851
IC_GET_FILE_IND. 28, 493, 521
IC_GET_KEY. 493, 522, 523, 681
IC_HANGUP. 30, 493, 524, 748, 851
IC_HEX_TO_NUM. 30, 493, 525
IC_INFOS_STATUS_TEXT. 30, 493, 526
IC_INSERT_STRING. 493, 527
IC_KILL_TERM. 493, 528
IC_LOGON. 30, 493, 529, 748, 851
IC_LOWER. 493, 530
IC_MOVE_FILE_DATA. 493, 531, 751, 755, 756, 851
IC_MOVE_STRING. 493, 532
IC_MSG_TEXT. 294, 493, 533, 851
IC_NUM_TO_HEX. 30, 493, 534
IC_PID_EXISTS. 30, 493, 535
IC_PRINT_STAT. 493, 536, 541, 545, 747, 851
IC_QUEUE_STATUS. 30, 493, 549
IC_REMOVE_DIR. 493, 550
IC_RENAME. 493, 551, 751, 755, 756, 851
IC_RESOLVE_FILE. 493, 552
IC_SEND_MAIL. 27, 493, 554, 555
IC_SEND_MSG. 28, 493, 556, 851
IC_SERIAL_NUMBER. 493, 557, 851
IC_SET_ENV. 28, 493, 558
IC_SET_TIMEOUT. 289, 463, 493, 559, 685, 851
IC_SET_USERNAME. 295, 493, 560
IC_SHUTDOWN. 30, 493, 561, 748, 851
IC_SYS_INFO. 28, 493, 562, 851
IC_TERM_CTRL. 493, 564
IC_TERM_STAT. 293, 493, 565
IC_TRIM. 28, 493, 567
IC_UPPER. 493, 568
IC_VERSION. 493, 569
IC_WINDOW_TITLE. 27, 493, 570
IC_WINDOWS_MSG_BOX. 27, 493, 572, 573
IC_WINDOWS_SETFONT. 27, 28, 493, 575
IC_WINDOWS_SHELLEXECUTE. 29, 493, 576
IC_WINDOWS_SHOW_CONSOLE. . . . 27, 493, 576-578
ICCHECK utility. 695-697, 762
ICCODEPATH. 499, 552, 753, 754, 764
ICCONFIG utility. 522
ICCREATE utility. 761
ICDATAPATH. 499, 521, 552, 755, 766, 767
ICEDCFW utility. 522
ICEXEC service. . . . 293, 498, 528, 691, 727, 747, 748, 771, 831, 838, 852
ICFONT. 575
ICFONTSIZE. 575
ICIDE. 28, 29, 691, 695, 793-795, 817
ICINFO utility. 692
ICISAM file. . 88, 93-95, 102, 111, 263, 267, 270, 316, 326, 529, 551, 689-691, 695, 704, 755, 761, 762, 780, 817, 821, 824, 826, 829, 836, 852, 853
ICISAM reliability. . 263, 267, 270, 310, 762, 821, 824, 826
ICLIB utility. 697
ICLINK utility. 509, 511, 531, 537, 539, 551, 552, 764, 768, 853
ICLOGS. 691
ICMAKEMS utility. 526
ICNETD service. 521, 691, 766, 767, 773
ICOBOL compiler. . . . 34, 59, 130, 285, 691, 697, 699, 702, 704, 706, 709, 711-713, 715, 721, 751, 772, 790
ICOBOL ODBC Driver. . 316, 412, 691, 695, 700, 701, 706-709, 717-719, 771, 772, 780-785, 790, 791
ICPACK utility. 695
ICPCQFILTER. 539
ICPERMIT service. 557, 691, 699, 765, 783, 861
ICQPRW. 25, 691
ICREORG utility. 762, 763, 765-767
ICREV utility. 292, 293, 697, 741
ICREVSET utility. 25, 293, 741
ICROOT. 689, 691, 696, 697, 702, 741, 743
ICRUN. . . 261, 463, 512, 526, 560, 563, 691, 695, 697, 699, 721-723, 747, 765, 767, 855
ICRUNRC client. 570, 572, 575, 577
ICRUNRS surrogate. 493, 566
ICRUNW. 29, 570, 572, 575
ICSCROPT. 722, 738
ICSDMODE. 669-671
ICSORT utility. 697
ICSP2. . . . . 25, 691
ICSTAT utility. 689
ICTERM. 282
ICTHINS surrogate. 566
ICTIMEOUT. 289, 463, 523, 559, 862
ICTMPDIR. 455, 697, 702, 741
Identification Division. . 52, 53, 55, 57-59, 67, 69, 716, 799, 801, 802, 807, 808, 811
IF statement. 373, 374, 440
IMMEDIATE. . . 27, 181, 251, 253, 293, 353, 354, 366, 433, 435, 483-485, 491, 586, 729, 749, 762, 848
index-name. 41, 123, 126, 127, 166, 174, 175, 199, 233, 235, 253, 277, 406, 408, 411, 439-441, 443, 444, 447, 448, 713, 717, 804, 809, 812
INDEXED BY phrase. . . 123, 126, 174, 439, 441, 444, 447, 713

indexed file. 28, 85, 86, 88, 90, 91, 93, 95, 96, 100, 101, 106, 110, 111, 141, 147, 150, 151, 163, 256, 257, 259-262, 264-266, 268-270, 326, 327, 331, 359, 390, 399, 402, 414-416, 418, 420-422, 424, 433-435, 452, 459, 469, 471, 472, 483, 484, 489, 490, 521, 704, 717, 718, 760-762, 765, 767, 771-773, 775, 797, 799, 803-805, 808, 809, 817, 823, 825, 829, 831, 836, 838
Information switch. 705
infostat.ms. 526
Infostat.txt. 526
inline comment. 28, 61, 62
INSPECT statement. 379, 381, 383, 384, 714
Install. 25, 690, 691, 699, 780, 782
installic. 782
Intel. 6, 191, 292
Interactive COBOL. . . . 1, 5, 25-29, 35, 176, 223, 463, 763, 767, 768, 771, 773, 779, 780, 782, 790, 819, 883
intercept spooling. 750
internal filename. 100, 751
Intr key. 736, 748, 764
Intrinsic Functions. 25, 27, 29, 129, 130, 579-581, 583, 584
ISAM file. 263, 767, 804, 831, 838
ISAM reliability. 263
ISQL. 27, 36, 37, 40-44, 48-51, 118-120, 128, 133-135, 167, 168, 192, 194-197, 204, 205, 226, 227, 232, 235, 238, 246, 251-253, 290-292, 311, 313, 315, 316, 319, 333, 348, 349, 351, 353, 363, 365, 366, 375, 377, 381, 393, 394, 412, 437, 443, 444, 449, 700, 704, 707, 729, 730, 734, 780, 783, 847
ISQL COMMIT statement. 311
ISQL CONNECT statement. 315, 316
ISQL DEALLOCATE statement. 319, 333, 334
ISQL EXECUTE IMMEDIATE statement. . . . 27, 251, 253, 353, 354, 366, 729
ISQL EXECUTE statement. 351, 352, 363, 412
ISQL FETCH statement. 351, 363, 364, 707
ISQL GET DIAGNOSTICS
COMMAND FUNCTION phrase. 365, 366
DYNAMIC FUNCTION phrase. 365, 366
MESSAGE LENGTH phrase. 365
MESSAGE TEXT phrase. 365
NUMBER phrase. 365
ISQL GET DIAGNOSTICS statement. . . 135, 251, 252, 365-367
ISQL PREPARE statement. 319, 351, 412, 413
ISQL ROLLBACK statement. 437
ISQL SET CONNECTION statement. . . 251, 315, 449, 450
ISQL SQLERROR. . . . 27, 43, 127, 128, 133, 134, 311, 316, 319, 333, 334, 351-354, 363-367, 412, 413, 437, 449, 450, 730, 734, 848
ISQL SQLSTATE. . . 251, 252, 311, 315, 319, 333, 351, 353, 363, 412, 437, 449, 848
item-name. 63
JAVA. 771, 783, 784
JUSTIFIED clause. . . 47, 119, 131, 172, 196, 465, 712
KEY IS phrase. 126, 174, 175, 415, 439, 440, 458
kill. 493, 528, 564, 727, 852
Kill Terminal. 528
LAST. . . 34, 40, 42, 59, 61, 63, 69, 174, 175, 179, 180, 185, 190, 210, 240, 254, 256, 266, 272, 274, 282-286, 293, 300, 305, 306, 326, 327, 337, 339, 357, 369, 371, 382, 384, 391, 392, 402, 407, 410, 417, 424, 430, 434-436, 440, 453-455, 460, 463, 476, 477, 486, 491, 541, 542, 545, 546, 669, 683, 701, 724, 727, 729, 732, 734, 737, 766, 800, 801, 810, 819, 848, 858
LEADING. . . . 45, 46, 50, 51, 119, 178-181, 186, 187, 225, 289, 337, 340, 380, 381, 383-385, 630, 634, 657, 719, 752, 772, 773, 776, 781, 785, 848
LENGTH OF. 27, 43-47, 88, 93, 95, 96, 108-112, 128, 132, 146, 163, 164, 175, 193, 197, 220, 234, 257, 258, 261, 264, 269, 275, 323, 366, 381, 393, 419, 425, 430, 440, 458, 460, 466, 502, 507, 516, 517, 527, 532, 543, 544, 548, 567, 581, 582, 590, 612, 719, 760, 776, 777, 817, 821, 823, 830, 837, 856
level-number. 33, 41, 63, 117, 118, 120, 143, 148, 165, 166, 168, 170, 173, 174, 183, 185, 189, 192, 199-201, 203, 206, 208, 247, 800, 804, 805, 808, 812, 813
library file. 695, 741
license. 3, 4, 27, 134, 501, 557, 691, 699, 760, 765, 767, 772, 784, 793, 831, 838, 861
license description file. 861
LINAGE. 29, 43, 52, 125, 127, 128, 133, 139, 140, 143, 149, 153-155, 400, 484, 486, 487, 848
LINE clause. 213, 217, 219-222, 228, 717
LINE NUMBER. . . . 153, 154, 286, 290, 291, 295, 340, 705, 710, 716, 717, 721, 723, 724, 729, 732, 734, 736, 740, 805
LINE phrase. 283, 286, 338, 340, 813
LINE SEQUENTIAL. 106, 108, 419
link file. 551, 695, 749, 754-756, 837, 853
Link Kit. 55, 301, 754
Linux. 26, 292, 781-783
LISTFILE. 749
Listing file switch. 705
little-endian. 190, 721
LOCK phrase. 276, 309, 401, 418, 424
logical operator. 239, 241, 799, 805, 806
Logon mode. 748
Lowest console. 293
LOWLIGHT. 214, 288, 340, 848
lp. 543, 747, 750, 764
LRC. 502, 805, 856
Master Console. 293
MERGE file. . . . 55, 84, 87, 88, 99, 115, 138, 142, 258, 331, 359, 389, 399, 427, 431, 432, 452, 454, 802, 803, 805, 811

MERGE Statement. . . 57, 138, 145, 254, 389-392, 401, 431, 805, 807
Message
error. 271, 282, 285, 294, 554, 686, 691, 693, 711, 734
message file. 526, 695, 860
Message Sending. 293, 500, 556, 564, 852
Message sending privilege. 500, 556, 564, 852
mnemonic-name. . . . 41, 71, 73-78, 279, 281, 282, 335, 336, 443, 445, 484, 486, 806, 811
modem. 750, 758, 831, 837
modem control. 750, 758
MOVE statement. . . 160, 178, 191, 194, 218, 247, 282, 285, 291, 293, 339, 375-377, 381, 393, 395, 416, 418, 423, 424, 427, 432, 434, 476, 477, 485, 489, 713, 735
MS-DOS. 6, 755, 756, 766
MULTIPLY statement. 397
native character set. 76-79, 88, 145, 146, 259, 806
negated combined condition. 806
negated simple condition. 806
network mode. 858
NO ADVANCING phrase. 337
No switch. 701, 706
nonnumeric item. 806
nonnumeric literal. . . 40, 44, 45, 47, 61, 75-79, 88, 196, 208, 234, 281, 299, 303, 307, 315, 319, 333, 336, 351, 353, 363, 380, 412, 465, 476, 806, 809
nonnumeric operand. 234
NOT ANSI. 400, 458, 460
numeric edited item. 289, 297, 313, 344, 394, 397, 467, 714
numeric item. . . 119, 130, 189, 190, 194, 195, 236, 282, 285, 289, 297, 313, 344, 381, 394, 397, 467, 713, 806, 858
numeric literal. 40, 43, 45-47, 163, 195, 230, 231, 281, 297, 344, 394, 397, 406, 415, 447, 467, 797, 804, 806, 807, 809
numeric operand. 234
NX file. . . 263, 267, 270, 271, 551, 695, 761, 762, 821, 824, 826, 828, 829, 835, 836, 853
obsolete. . . 47, 52, 67, 69, 74, 114, 148, 152, 162, 258, 293, 463, 703, 806
OCCURS. . 28, 29, 40, 45, 55, 65, 66, 93-95, 120-124, 126, 127, 154, 166, 174, 175, 183, 185, 197, 203, 206, 218, 222, 229, 245-247, 253, 254, 258, 271-274, 288, 311, 316, 319, 333, 344, 352, 353, 363, 366, 369, 381, 382, 390, 392, 393, 395, 402, 407, 413, 417, 418, 423, 426, 431, 435, 437, 439-441, 444, 447-449, 452, 454, 455, 459-461, 476, 479, 480, 487, 496, 510, 521, 541, 589, 590, 592, 596, 605, 610, 612, 613, 674, 689, 691, 716, 717, 719, 727, 757, 761, 793, 812, 813, 818, 829, 837, 848, 855
OCCURS clause. . . . 120-123, 126, 127, 174, 175, 183, 185, 197, 218, 222, 258, 390, 392, 393, 395, 439-441, 444, 447, 452, 454, 455, 590, 612, 719, 761, 812, 818
ODBC. 27, 134, 135, 316, 717, 719, 771, 779-786, 788-791
ODBC Administrator. 316, 779, 781
ON SIZE ERROR phrase. 246, 251, 298, 313, 344, 345, 397, 468
On UNIX. . 25, 109, 271, 293, 331, 359, 403, 487, 488, 493, 497, 498, 509, 510, 528, 544, 546-548, 550, 560, 563, 575, 689-693, 695, 696, 699, 701, 702, 725, 727, 741, 747, 749, 750, 752, 754-756, 760, 764, 767, 771, 780, 781, 827, 835, 852, 854, 855, 857, 859, 860
On UNIX only. 767
On Windows. 25, 29, 109, 331, 359, 487, 488, 493, 497, 498, 509, 528, 546, 547, 560, 563, 573, 575, 576, 578, 689, 690, 692, 693, 696, 697, 699, 701, 702, 725, 727, 747, 750, 752, 760, 765, 771, 779, 793, 827, 835, 854, 857-860
On Windows only. 493, 765
OPEN statement. . . . 109, 145, 154, 259, 260, 262, 264-266, 269, 272, 391, 392, 399-403, 454, 455, 459, 486, 717, 747, 755, 764, 801-804, 807, 821, 823, 825
operational sign. . . . 119, 177, 178, 186, 187, 225, 237, 394, 579, 580, 806, 807
optional. . . 28, 33, 36, 42, 43, 48-51, 67, 71, 73, 83-85, 88, 106, 117, 119, 123, 135, 165, 178, 186, 187, 199, 200, 225, 227, 229, 252, 253, 260-262, 264, 265, 268, 272, 273, 307, 309, 315, 322, 326, 327, 362, 388, 391, 392, 400-402, 417, 430, 435, 439, 455, 459, 463, 469, 472, 509, 519, 524, 531, 549, 554, 561-564, 582, 677, 686, 692, 726, 734, 772, 773, 775-777, 798, 803, 807, 810, 823, 829, 836, 848, 854, 858, 861
Ordinal number. 77-79, 131, 636, 637, 807, 809
ORGANIZATION clause. 106
Output file switch. 707
P PICTURE. 176, 177
PACKED-DECIMAL. 189-191, 848
paragraph-name. . . . 41, 42, 62, 70, 125, 229, 230, 407, 717, 807, 808
parallel. 750, 758
PASS. . . . 282, 301, 306, 341, 407, 411, 496, 544, 554, 706, 727, 730, 749, 768
PATH . . 93, 94, 96, 110, 111, 268, 269, 275, 322, 328, 362, 388, 403, 425, 426, 430, 436, 472, 490, 491, 499, 552, 575, 576, 701, 703, 741, 743, 761, 762, 764, 781-783, 807, 818, 827, 829, 831, 833, 835, 838-842, 855
PCQ. 107, 403, 536-539, 541, 542, 544-548, 562, 563, 749, 757, 758
PERFORM statement. . 55, 57, 123, 127, 253, 254, 357, 371, 392, 404, 406-411, 444, 447, 454, 455, 479, 721, 729
period. 4, 34, 39, 40, 62-64, 78, 80, 138, 178-180, 229, 252, 253, 255, 373, 440, 522, 582, 586,

639, 700, 751, 797, 798, 801-803, 807, 808, 810, 811, 851
permissions. 331, 359, 560
PICTURE. . . . 52, 78-80, 119, 165, 168, 176, 179, 180, 195, 196, 199, 208, 209, 223, 250, 282, 285, 712, 715
. PICTURE. 176, 223
, PICTURE. 33, 122, 168, 189, 192, 226
+ PICTURE. 179
- PICTURE. 179
A PICTURE. . . 30, 39, 40, 52, 176-182, 186, 187, 189, 199, 208, 209, 225, 493, 496, 712, 798
B PICTURE. 177
CR PICTURE. 176, 179
DB PICTURE. 176, 179
P PICTURE. 176
PICTURE character-string. 118
S PICTURE. 176, 223
V PICTURE. 176
PLUS phrase. 460
Print Pass Through. 282, 341
Print Screen. 844
Printer Control
directory. 403
file. . . 263, 267, 331, 359, 403, 695, 747, 757-759, 821, 823, 827, 835
privilege. 500, 501, 536, 544, 548, 855
queues. . . . 107, 541, 545, 549, 747, 749, 750, 759
utility. 403, 536, 539, 541, 545, 549, 560, 747, 750, 855, 861
Printer control management privilege. 500, 501
PRN. 562, 749
Procedure Division. . . 36, 42, 52, 53, 56-59, 62, 63, 80, 133, 154, 189, 190, 193, 200, 229, 230, 250, 253-255, 277, 300, 304, 306, 337, 357, 369, 371, 389, 406, 452, 479, 716, 801, 802, 807, 808, 810, 817
procedure-name. 229, 355, 369, 404-408, 479, 808
processes. 283, 285, 299, 301, 303, 331, 339, 562, 753, 755, 830, 831, 837, 842
Program debugging privilege. 500
program lines. 57, 58, 61, 750, 758, 799, 811
program mode. 498, 556, 748, 853, 855
program switches. 303, 305, 752-754, 854
program-name. . . . 41, 67, 69, 292, 293, 299, 303, 481, 717, 808, 812, 854
purge. 521, 804, 829, 836, 848
QPR. 691
qualification. 43, 123-125, 133, 136, 273, 803
QUEUE IS. 29, 84, 107, 549, 747, 759
Quiet switch. 693, 694
Quit key. 725, 764
radix. 117, 119, 190, 193, 246
READ statement. 88, 145, 250, 251, 261-268, 270, 273, 275, 326, 327, 391, 414-426, 434, 454, 460, 717, 736, 798, 821, 823, 825
readme. 293, 690
reason code. 724-726, 731, 736, 737
RECORD clause. 36, 152, 157-160, 175, 257, 258, 323, 416, 432, 808
RECORD KEY clause. . . 92-94, 96, 110-112, 257, 418, 426, 460, 461, 490
record-name. 41, 63, 125, 138, 171, 416, 423, 427, 432-436, 483-487, 489, 490, 803, 809
RECORDING MODE clause. . 106, 108, 158-160, 162, 163, 712
REDEFINES. . 122, 124, 126, 166, 168, 171, 183, 193, 196, 247, 249, 277, 292, 376, 511, 521, 522, 541, 545, 552, 553, 701, 709, 717, 718, 848, 859
REDEFINES clause. 124, 126, 168, 171, 183, 196, 277, 376, 717, 718
reference modification. 27, 29, 124, 130, 131, 249, 432, 517, 527, 532, 567, 803, 833, 839
reference modifier. 393, 809
REGISTRY. 794
relation condition. . . . 41, 132, 136, 189, 233, 240, 241, 391, 453, 580, 797, 809, 811
relational operator. . 233, 234, 240, 241, 459, 461, 797, 809
relative file. . . 85, 90, 91, 141, 160, 163, 256, 262, 263, 265, 267, 268, 310, 326, 327, 390, 392, 402, 414, 415, 417, 418, 420, 422, 424, 434, 435, 454, 455, 458, 459, 469, 483, 488, 489, 759, 761, 762, 803, 804, 809, 817, 821, 823-826, 829, 831, 836, 838
relative key. . 90, 91, 259, 265, 268, 273, 326, 327, 392, 415-418, 422-424, 435, 454, 455, 458, 459, 469, 489, 762, 798, 803, 809, 817, 827-829, 835, 836
RELATIVE KEY phrase. . 90, 416, 418, 422, 424, 458, 459, 489
RELEASE statement. 427, 454
RENAMES. . . . 118, 167, 173, 185, 247, 277, 375, 376, 701, 709, 718, 848
RENAMES clause. . 118, 173, 185, 277, 375, 376, 718
REQUIRED. . . . 2, 3, 33, 34, 39, 40, 42, 43, 54, 67, 88, 117, 119, 120, 122-124, 126, 129, 130, 134, 143, 159, 165, 173, 174, 183, 190, 191, 196, 199, 200, 203, 205-208, 210, 218, 228, 285, 288, 352, 363, 383, 439, 493, 515, 517, 527, 532, 554, 555, 560, 565, 579, 651, 653, 691, 692, 694, 700, 706, 713, 717, 718, 726, 732, 755, 761, 766, 771-773, 775, 776, 783, 798-805, 811, 829, 831, 832, 836, 838, 839, 848
REQUIRED clause. 210
reserved words. 29, 33, 35, 36, 40, 42, 43, 46, 802, 805, 806, 809, 810, 847, 849
RETURN statement. 160, 250, 392, 431, 432, 454, 455
Revision switch. 707
REWRITE statement. . . . 145, 259, 262, 263, 265, 266, 268, 272, 418, 424, 425, 433-436, 717, 821, 823, 825
ROUNDED. . . 190, 191, 245, 246, 289, 297, 298, 313, 343-345, 397, 419, 467, 468, 608, 647, 848

Run Program. 749
run unit. 52-55, 69, 71, 95, 97, 112, 133, 145, 146, 149, 155, 171, 299-301, 303, 306, 307, 311, 315, 316, 333, 401, 411, 412, 437, 449, 463, 486, 641, 724, 727, 768, 798, 802, 804, 810, 812
runtime. 25, 27-29, 35, 56, 95, 107, 129-131, 134, 147, 149, 193, 194, 199, 263, 267, 270, 285, 286, 289, 292, 293, 295, 301, 304, 305, 338, 340, 347, 351, 363, 366, 412, 419, 463, 487, 493, 498, 500, 501, 513, 524, 528, 553, 557, 561, 569, 572, 575, 577-580, 690-692, 699, 701, 704, 707, 709, 712, 721, 724, 726, 727, 733, 734, 747, 749, 753, 756, 760, 764-769, 771, 780, 783, 817, 818, 821, 824, 826, 830, 837, 851, 855, 861, 862
S PICTURE. 176-178
SAME clause. 115, 273, 452
SAME RECORD AREA. . 54, 115, 273, 390, 416, 422, 427, 434, 485
SCO. 292
SCREEN DEMON. 669-671, 685
SCREEN HANDLER. 669-671, 682, 685, 768, 769
SCREEN OPTIMIZER. 722
screen-data. . . . 200, 203, 205, 208-210, 213, 214, 217-219, 223, 224, 281, 283, 284, 339
screen-group. 200, 206-212, 214, 224, 283
screen-literal. . . 200-202, 208, 209, 213, 214, 217, 219, 339
screen-name. . . 200, 208, 279, 281, 283, 284, 335, 336, 338, 339, 717, 810, 813
SEARCH ALL statement. 704
SEARCH statement. 123, 126, 235, 439-442, 799
section-name. 41, 42, 125, 229, 230, 407, 717, 808, 810
SECURE clause. 224, 288
SEPARATE CHARACTER. . . 166, 178, 186, 187, 204-206, 225, 394, 476
separator. . . . 34, 39, 40, 44, 62-65, 179, 180, 229, 252, 253, 255, 373, 440, 632, 752, 801-803, 807, 808, 810, 812
sequential file. . . 29, 84, 88, 90, 98, 100, 106-108, 139, 140, 160, 162, 256, 258, 260, 262, 265, 309, 331, 359, 399, 402, 403, 414, 415, 419, 420, 422, 426, 433, 435, 457, 458, 460, 483, 486, 704, 712, 757, 759, 764, 800, 803, 810, 829, 831, 836, 838, 858
SER. 563, 749
services. 769
SET statement. 41, 71, 75, 77, 123, 124, 127, 131, 132, 136, 189, 193, 233, 375-377, 443-445, 447, 448, 713
shared data. 57
shared memory. 771, 830, 837, 842
shared objects. 781, 782
SHELL. . . 301, 302, 497, 565, 689, 692, 702, 730, 741, 748, 764, 852
SIGN. . . . 45, 50, 51, 71, 73-75, 78, 79, 119, 131, 145, 166, 176-181, 186, 187, 189, 190, 192-194, 204-206, 223, 225-227, 233, 234, 236-238, 284, 289, 337, 340, 344, 381, 394, 460, 476, 529, 570, 573, 579, 580, 582, 583, 611, 630, 632, 645, 657, 659, 714, 715, 718, 719, 731, 751, 752, 785, 798, 800, 801, 806-808, 810-812, 848, 852
SIGN clause. 78, 79, 119, 178, 179, 186, 187, 225, 800
sign condition. 238, 810, 811
SIZE ERROR. . 130, 232, 246, 251, 252, 297, 298, 313, 343-345, 397, 467, 468, 580, 714, 763
Solaris. 6
SORT file. 138, 451, 811
SORT Statement. 57, 392, 401, 427, 451-455, 804, 807, 811
SORT-MERGE. 55, 87, 88, 99, 115, 138, 142, 258, 389, 390, 427, 431, 432, 452, 802, 803, 811, 813, 848
SP2. . . . . 25, 565, 566, 578, 691, 832
SPECIAL-NAMES. . . 41, 47, 71-75, 78-80, 136, 179, 236-238, 282, 336, 390, 453, 484, 797, 798, 800, 802, 807, 811, 812, 848
Spooler
    UNIX. 747, 750, 764
spooling. 747, 750, 851, 855, 861
SQL. . . 27, 36, 40, 41, 48, 49, 120, 134, 135, 193, 194, 226, 227, 292, 311, 315, 333, 351-354, 365, 366, 412, 437, 449, 704, 729, 730, 734, 771, 773, 780, 782-786, 791, 848
Standard COBOL. 40, 47, 52, 67, 69, 74, 114, 148, 152, 291, 336, 463, 806
START statement. 90, 93, 110, 258-260, 266, 415, 422, 457, 459-461, 821, 823, 825
STOP statement. 289, 463
STRING statement. 465, 466
stty. 764
subscript. . . 93, 120, 123, 124, 126-128, 344, 393, 432, 713, 731, 812
SUBTRACT statement. . . 246, 247, 249, 251, 467, 797
suffixed. 94, 718, 761
SunOS. 6
suppress. 85, 93-95, 100, 275, 276, 287, 426, 433, 436, 484, 491, 542, 694, 709, 719, 774, 777, 801, 848
SUPPRESS WHEN clause. 719
switch. . . . 27, 28, 35, 41, 44, 59, 62, 71, 73, 75, 77, 78, 80, 100, 120, 136, 188, 233, 237, 238, 293, 295, 304, 305, 369, 401, 439, 443, 445, 479, 494, 692-694, 700-707, 709, 711, 712, 714, 717, 718, 721, 741, 744, 745, 749, 753, 760, 762, 763, 765, 767, 772, 799, 802, 806, 811, 812, 848, 850
switch-name. 73, 238, 811
symbol file. . . . 701, 709, 721-723, 729, 732, 733, 735-737, 740, 793

symbolic. . . 41, 47, 71, 73-78, 331, 359, 403, 496, 801, 811, 812, 833, 839, 842, 848, 858, 859
symbolic links. 331, 359, 403, 833, 839, 859
SYNCHRONIZED clause. 120, 188
system calls. . . 303, 304, 306, 496, 510, 544, 729, 730, 754, 756, 768, 829, 832, 836, 838, 851-858, 860-862
System Information. 293, 500, 562, 565, 856, 860
System Information privilege. 500, 562, 860
System Parameters. 563, 860
System Shutdown privilege. 500, 561, 855, 862
system.pq. 747
system-name. 40, 42, 799, 805, 809, 812, 813
SYSTEM-CODE. 293
tab. 29, 60, 62, 281, 287, 288, 341, 523, 549, 702, 768, 848
TCP. 691
TCP/IP. 691
terminal description file. 522, 695, 768
Terminal number switch. 749
Terminal Status. 293, 498, 500, 528, 529, 556, 564, 565, 851-853, 856
Terminal status privilege. 500, 564, 565, 856
text-name. 41, 64, 812
ThinClient. 493, 570, 573, 575, 577, 690, 691
THRU phrase. 196
time zone. 697, 698
TIME-OUT. . . 279, 281, 289, 414, 419, 522, 670, 680, 685, 848, 862
timeout. . . 135, 289, 310, 419, 426, 463, 488, 493, 522, 523, 559, 670, 671, 678-681, 685, 750, 751, 757, 758, 821, 824, 826, 828, 831, 836, 837, 851, 862
TO clause. 208, 209, 218, 223, 712, 717
TRAILING. . . . 186, 187, 225, 554, 556, 558, 567, 570, 572, 630, 634, 657, 719, 752, 773, 776, 785, 847, 848
TSR. 765
ttyname. 749
TZ. 697, 698
UNDELETE. . . 102, 251, 252, 255, 259, 260, 262, 265, 266, 269, 271, 272, 274, 326, 328, 401, 469, 471, 472, 730, 766, 767, 804, 821, 823, 825, 847, 848
UNDELETE statement. 272
UNDERLINE. 214, 287, 288, 340, 341, 669, 673, 848
UNDERLINED. . . 33, 34, 43, 214, 288, 341, 672, 676-679, 848
uniqueness of reference. . . 41, 124, 125, 136, 799, 803
UNIX . . . 6, 25, 26, 109, 271, 292-294, 301, 302, 316, 331, 359, 403, 487, 488, 493, 497, 498, 509, 510, 519, 528, 537, 541-548, 550, 560, 563, 575, 689-693, 695, 696, 699, 701, 702, 725, 727, 741, 747, 749, 750, 752, 754-756, 760, 764, 767, 771, 777, 780, 781, 827, 835, 843, 852, 854, 855, 857, 859, 860
unixODBC. 316, 771, 780-783
Unixware. 292
UNLOCK statement. 276, 473
UNSTRING statement. 47, 251, 475-478
UNTIL phrase. 253, 408, 410
url. 493, 507, 516, 576, 773, 780
USAGE clause. 119, 189, 190, 192, 193, 199, 226, 237
USE Statement. 230, 252, 260, 271, 272, 417, 423, 479, 480, 801
User Library. 691
USER NAME. 290, 291, 295, 315, 537, 542, 560, 722, 786
user-defined subroutine. 55
user-defined word. . . . 40, 41, 69, 73, 74, 256, 277, 714, 797-800, 802, 804-810, 812, 813
user-id. 541-543, 545, 547, 560, 782, 783
USING clause. 134, 208, 209, 218, 222, 223, 352, 712, 717
USING phrase. . 56, 189, 199, 277, 299-301, 303, 304, 306, 357, 371, 452, 454, 817
V PICTURE. 176-178
VALUE clause. . . . 47, 55, 69, 122, 131, 165, 171, 183, 189, 195-197, 199, 208, 220, 228, 376, 377, 445, 718
variable length record. 256
variable length records. . 54, 94, 96, 111, 158, 162, 257, 258, 389, 390, 392, 452, 453, 455, 813
variable origin. 283, 284, 338, 339, 813
VARYING phrase. . . 123, 127, 175, 193, 197, 258, 411, 441
virtual memory. 696
Watch Facility. 293
Watch other terminals privilege. 500, 564
WHEN phrase. 94, 349, 350, 440, 441
Windows. . . . 25-29, 109, 271, 292, 294, 301, 302, 316, 331, 359, 487, 488, 493, 497, 498, 509, 519, 528, 537, 541-543, 545-547, 560, 563, 570, 572, 573, 575-578, 689-693, 696, 697, 699, 701, 702, 715, 723, 725, 727, 737, 738, 744, 747, 750, 752, 760, 765, 771, 779, 793, 794, 827, 835, 854, 857-860
Windows print spooler. 747
Windows printer. 750
Windows Server. 26
Windows Vista. 26
Windows XP. 26
WITH DEBUGGING MODE clause. 73
Working-Storage. . . . 54, 111, 122, 137, 165, 168, 170, 171, 173, 195, 196, 198-200, 222, 223, 249, 279, 285, 290, 299, 303, 322, 716, 806, 810, 813, 849
WRITE statement. 78, 90, 112, 154, 160, 161, 175, 250, 251, 256, 259-261, 264, 266, 268, 271, 272, 274, 275, 341, 392, 418, 424, 425, 455, 483-490, 717, 821, 823, 825
X PICTURE. 176-178
XD file. . . 263, 267, 270, 551, 695, 761, 762, 821, 824, 826, 828, 829, 835-837, 853

Z PICTURE. 176-178
ZERO. . . . 34, 42, 44-47, 62-64, 81, 106, 107, 118-120, 153, 154, 158, 160, 166, 168, 169, 174, 176-181, 189, 192-194, 196, 203, 205, 208, 209, 219, 226, 227, 229, 230, 232, 234, 236-238, 246, 249, 262, 266, 269, 271, 286, 288, 291, 294, 323, 328, 338, 340, 341, 361, 366, 381, 383, 384, 393, 394, 406, 407, 419, 425, 430, 436, 458, 460, 461, 466, 476, 479, 484, 486, 491, 494, 496, 497, 499, 502, 506, 507, 516, 522, 524, 529, 536-539, 543, 544, 561, 563, 565, 575, 579, 584-589, 592, 593, 596, 597, 600, 605, 611, 614, 615, 619, 620, 622, 625, 626, 628, 631, 632, 634, 641, 643, 645-647, 655, 657, 659, 675, 678, 679, 693, 702, 711, 713, 714, 730, 759, 760, 773, 800, 801, 804, 805, 807, 809, 810, 821, 823, 825, 828, 836, 840, 841, 849, 856, 858, 859
zero suppression. 176, 179, 181
[ ]. 33, 34, 206, 692, 726, 752

SUPPORT

Questions, comments, bugs, etc. can be addressed to your supplier or distributor or to the Developers at:

Envyr Corporation
Suite 160
4904 Waters Edge Drive
Raleigh, N.C. 27606
U.S.A.

FAX	(919) 851-4609
WWW	www.icobol.com
FTP	ftp.icobol.com
E-Mail:	support@icobol.com

Support Information Request (SIR)

Support FAX: (919) 851-4609 Date: __/__/__

Support E-mail: support@icobol.com

FROM Originator ID No:_____

Contact: _____ Phone: _____

Company: _____ Fax: _____

Address: _____ Timezone: _____

_____

City: _____ State/Province: _____

Country: _____ Zip/Mail Code: _____

ICOBOL revision: _____ Purchased From: _____

Kind of problem:_____ (Enhancement/Suggestion, Question, Documentation error, Software error)

Frequency: _____ (Frequent, Occasional, Erratic, Reproducible)

Significance: _____ (Low, Medium, High, Urgent)

Host Machine Configuration

Vendor: _____ Model: _____ CPU type: ____

Amount of Memory: _____

Peripherals:

OS Name and Version: _____ _____

Other software in Use, with versions:

Rebooted from scratch? Y N

(turned the power off and back on)

Problem/Suggestion: (Describe as fully as possible. If a COBOL problem, a sample of code that generates the error would be appreciated.)

Attachments: _____ (None, tape, floppy, listings, etc.)
(Please label attachments with company, contents, format, and "# of #" (e.g. 1 of 2).

From: _____

_____

_____

Stamp

Envyr Corporation
Support
#160
4904 Waters Edge Drive
Raleigh, N.C. 27606
U.S.A.

(Fold Here)

Enhancement/Suggestion Request

FAX (919) 851-4609

Date: ___/___/___

OEM: _____ Timezone:

contact: _____ Phone: _____

ICOBOL revision: _____

Suggestion: (Describe as fully as possible. If possible give a brief example of COBOL code to use this feature.)

From: _____

_____

_____

Stamp

Envyr Corporation
Suggestions
#160
4904 Waters Edge Drive
Raleigh, N.C. 27606
U.S.A.

(Fold Here)

CUSTOMER DOCUMENTATION COMMENT FORM

FAX (919) 851-4609

Name: _____

Title: _____

Company: _____

Street: _____

City: _____

State: _____ Zip: _____

Country: _____

Manual: Interactive COBOL Language Reference & Developer's Guide
No: 011-00100-13

About the manual:

Is it easy to read?	Yes	No
Is it easy to understand?	Yes	No
Are the topics logically arranged?	Yes	No
Is the information correct?	Yes	No
Can you easily find what you want?	Yes	No
Does it tell you what you need to know?	Yes	No

Comments:

From: _____

_____

_____

Stamp

Envyr Corporation
Documentation
#160
4904 Waters Edge Drive
Raleigh, N.C. 27606
U.S.A.

(Fold Here)