

GNU COBOL 2.0

(Formerly OpenCOBOL)

[11FEB2012 Version]

Programmer's Guide

2nd Edition, 21 November 2013

Gary Cutler

CutlerGL@gmail.com

OpenCOBOL Copyright © 2001-2009 Keisuke Nishida

OpenCOBOL Copyright © 2006-2012 Roger While

Under the terms of the GNU General Public License

Document Copyright © 2009-2013 Gary Cutler

Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License [FDL], Version 1.3
or any later version published by the Free Software Foundation;

with Invariant Section "What is GNU COBOL?", no Front-Cover Texts, and no Back-Cover Texts

A copy of the FDL is included in the section entitled "GNU Free Documentation License"

GNU COBOL is an evolving tool.

While all reasonable attempts will be made to maintain the currency of the information in this document, neither the author of this document nor the authors of the GNU COBOL software, extend any warranties of any kind for this document or for the information contained therein.

Summary of Changes

Edition	Date	Change Description
2 nd	17 July 2012	<p>Updated for version 11FEB2012 of GNU COBOL 2.0</p> <ul style="list-style-type: none"> ▶ The use of a slash character (“/”) in column 7 was documented – this feature has existed since at least the 06FEB2009 version of OpenCOBOL 1.1, but was undocumented (section 1.6) ▶ Added documentation on the DEBUG-ITEM special register (section 6.1.8). ▶ Updated DECLARATIVES documentation to better explain how to use it. See section 6.1.4. ▶ A new section was added to the documentation to discuss the ramifications, rules and capabilities of sub-programming (section 7). ▶ Documentation was added on the COB_SET_DEBUG environment variable (section 8.1.4). ▶ The listings of all sample programs in chapter 9 are now presented as listings generated by the GNU COBOL Interactive Compiler utility (itself included as a sample program in section 9.4). This not only shows full source listings of the sample programs but complete cross-reference listings as well. ▶ A new sample program – DAY-FROM-DATE – was introduced to illustrate how to write a user-defined function (section 9.3) <ul style="list-style-type: none"> ▶ User-defined functions are now supported (sections 3, 7.1, 7.4.2 and 9.3) ▶ A new built-in subroutine – CSPRINTABLE – was introduced (section 8.3.1.11) (the COBDUMP sample program (section 10.2) now uses it!
	7 July 2011	<p>Updated for pre-release version 29APR2011 of OpenCOBOL 2.0</p> <ul style="list-style-type: none"> ▶ Corrected a problem with bogus footnote references in Figure 6-23. ▶ A reference to a new figure documenting error codes was added to the EXCEPTION-STATUS function (section 6.1.7.21). ▶ Documentation was added to the CLOSE statement (section 6.4.7) to explicitly document how the last record written to a LINE SEQUENTIAL or LINE ADVANCING file may have a terminating delimiter sequence written at the time the file is closed. ▶ Documentation was added to the WRITE statement (section 6.4.50) to explicitly document how the ADVANCING options are handled with LINE SEQUENTIAL and the new LINE ADVANCING files. ▶ Additional documentation on the cobcrun command (section 8.2.2) was added. ▶ LINE ADVANCING files are now supported (section 1.3.3.5). ▶ Floating-point literals of the form <code>[+-.]nn.nnE[+-.]nn</code> are now supported (section 1.8) ▶ Z"xxxx" null-delimited alphanumeric literals are now supported (section 1.8) ▶ The COPY statement now supports the COBOL2002 standard LEADING and TRAILING options as well as the “IN/OF library-name” and SUPPRESS PRINTING options (section 2.1.1) ▶ The REPLACE Compiler-Directing Facility (CDF) statement was introduced (section 2.1.2) ▶ Conditional code generation is now supported through the use of >>DEFINE, >>IF, >>SET, >>SOURCE and >>TURN Compiler-Directing Facility (CDF) directives (section 2.2) ▶ The COB_LINE_TRACE environment variable was renamed to COB_SET_TRACE (section 8.2.4). ▶ The COB_DISPLAY_WARNINGS (section 8.2.4) environment variable was introduced. ▶ SOURCE-COMPUTER WITH DEBUGGING MODE is now supported (section 4.1.1) ▶ The CHARACTER CLASSIFICATION clause of the OBJECT-COMPUTER clause is now supported (section 4.1.2). ▶ Mnemonic names are now optional for SWITCH declarations in SPECIAL-NAMES (section 4.1.4); Eight new switches (SWITCH-0, SWITCH-9 thru SWITCH-15) are now available; Switches may be specified as SW0 thru SW15 as well as SWITCH-0 thru SWITCH-15; a new print channel designation of CSP is now available; SYMBOLIC CHARACTERS are now supported (section

		<p>4.1.4)</p> <ul style="list-style-type: none"> ▶ The device name DISC may now be used interchangeably with DISK in SELECT statements (section 4.2.1) ▶ Files may now be SELECTed with the “NOT OPTIONAL” designation in addition to “OPTIONAL” (section 4.2.1). ▶ New USAGES of BINARY-INT, BINARY-LONG-LONG and COMPUTATIONAL-6 (Figure 5-10 and section 7.8.3) were introduced. ▶ The LEFTLINE screen attribute was added to the SCREEN SECTION (section 5.6). ▶ New intrinsic functions were introduced: <ul style="list-style-type: none"> ○ MODULE-CALLER-ID (section 6.1.7.47) ○ MODULE-DATE (section 6.1.7.48) ○ MODULE-FORMATTED-DATE (section 6.1.7.49) ○ MODULE-ID (section 6.1.7.50) ○ MODULE-PATH (section 6.1.7.51) ○ MODULE-SOURCE (section 6.1.7.52) ○ MODULE-TIME (section 6.1.7.53) ▶ A new option - WITH KEPT LOCK (section 6.1.9.2) - was added to the READ verb. ▶ USE FOR DEBUGGING is now supported (section 6.1.4) ▶ The following changes were made to the ACCEPT Statement <ul style="list-style-type: none"> ● The TIMEOUT option was added to Format 4 (section 6.4.1.4). ● The non-functional CONVERSION option was added to Format 4 (section 6.4.1.4). ● The LINE NUMBER option (a synonym for LINES) and COLS option (a synonym for COLUMNS) and ESCAPE KEY options were added to Format 6 (section 6.4.1.6) ● A new format – Format 7 – was introduced (section 6.4.1.7) ▶ The ALTER verb (section 6.4.4) is now supported [<i>Editorial Comment: this change was made only because NIST tests need it and not because you should be using it!</i>] ▶ Options (mnemonic-name, STDCALL and STATIC) were added to the CALL verb (section 6.4.5). ▶ The non-functional CONVERSION option was added to Format 4 of the DISPLAY statement (section 6.4.12.4). ▶ The REVERSED option for the OPEN statement is now supported syntactically, even though it is non-functional (section 6.4.29). ▶ The READY TRACE (section 6.4.32) and RESET TRACE (section 6.4.34) statements were introduced. ▶ A new option – STATUS – was added to the STOP verb (section 6.4.42). ▶ The following built-in <u>named</u> subroutines were added: <ul style="list-style-type: none"> ● C\$CALLEDBY (section 8.3.1.1) ● C\$GETPID (section 8.3.1.6) ● CBL_GET_CSR_POS (section 8.3.1.29) ● CBL_GET_SCR_SIZE (section 8.3.1.30) ▶ The following built-in <u>numbered</u> subroutines were added: <ul style="list-style-type: none"> ● X"E4" (section 8.3.2.2) ● X"E5" (section 8.3.2.3)
1 st	17 Sept 2010	<ul style="list-style-type: none"> ▶ Introduced documentation for the hitherto undocumented “COBCPY” environment variable (sections 8.1.4 and 8.1.5). ▶ Corrected “section 0” broken hyperlinks in the document.
	1 Apr 2010	Elaborated on the use of the GLOBAL clause in data item definitions (section 5.3).
	23 Jan 2010	INITIAL RELEASE OF DOCUMENT – corresponds to version 06FEB2009 of OpenCOBOL 1.1

Table of Contents

1. INTRODUCTION	1-2
1.1. What is GNU COBOL?.....	1-2
1.2. Additional References and Documents	1-2
1.3. Introducing COBOL	1-2
1.3.1. "I Heard COBOL is a Dead Language!"	1-2
1.3.2. Programmer Productivity – The "Holy Grail"	1-4
1.3.3. Notable COBOL/GNU COBOL Features	1-5
1.3.3.1. Basic Program Readability	1-5
1.3.3.2. COBOL Program Structure	1-6
1.3.3.3. Copybooks	1-6
1.3.3.4. Structured Data	1-7
1.3.3.5. Files	1-7
1.3.3.6. Table Handling	1-10
1.3.3.7. Sorting and Merging Data	1-10
1.3.3.8. String Manipulation	1-10
1.3.3.9. Textual-User Interface (TUI) Features.....	1-12
1.4. Syntax Description Conventions	1-12
1.5. General GNU COBOL Program Format	1-13
1.5.1. Source Line Format	1-13
1.5.1.1. Fixed Format Mode	1-13
1.5.1.2. Free Format Mode.....	1-14
1.5.2. Program Structure.....	1-15
1.6. In-Program Documentation (i.e. "Comments")	1-16
1.7. Use of Commas and Semicolons	1-16
1.8. Use of Literals	1-17
1.8.1. Numeric Literals	1-17
1.8.2. Alphanumeric Literals	1-17
1.9. Use of Figurative Constants	1-18
1.10. User-Defined Names	1-18
1.11. Use of LENGTH OF.....	1-19
2. THE GNU COBOL COMPILER DIRECTING FACILITY [CDF]	2-1
2.1. Text Manipulation Statements	2-1
2.1.1. The COPY Statement	2-1
2.1.2. The REPLACE Statement.....	2-1
2.2. CDF Directives.....	2-2
2.2.1. The >>DEFINE Directive	2-2
2.2.2. The >>IF Directive	2-3
2.2.3. The >>SET Directive.....	2-4
2.2.4. The >>SOURCE Directive	2-4
2.2.5. The >>TURN Directive	2-5
3. IDENTIFICATION DIVISION	3-1
4. ENVIRONMENT DIVISION	4-1
4.1. CONFIGURATION SECTION	4-1
4.1.1. SOURCE-COMPUTER Paragraph	4-1
4.1.2. OBJECT-COMPUTER Paragraph	4-2
4.1.3. REPOSITORY Paragraph.....	4-3
4.1.4. SPECIAL-NAMES Paragraph	4-3
4.1.4.1. The <i>alphabet-name</i> Clause	4-6
4.1.4.2. The <i>class-name</i> Clause	4-6
4.1.4.3. The <i>switch-definition</i> Clause.....	4-7
4.1.4.4. The <i>symbolic-characters</i> clause	4-7
4.2. INPUT-OUTPUT SECTION.....	4-8
4.2.1. File SELECT Statement	4-9
4.2.1.1. SELECT Without an "organization-clause"	4-11
4.2.1.2. ORGANIZATION SEQUENTIAL Files	4-11

4.2.1.3. ORGANIZATION LINE SEQUENTIAL Files	4-12
ORGANIZATION RELATIVE Files	4-13
4.2.1.4. ORGANIZATION INDEXED Files	4-14
4.2.2. I-O-CONTROL Paragraph	4-15
5. DATA DIVISION	5-1
5.1. File Or Sort/Merge File Descriptions	5-2
5.1.1. Record Descriptions	5-4
5.2. Describing Data Items	5-4
5.2.1. Defining non-SCREEN SECTION Data Items	5-6
5.2.1.1. ANY LENGTH Clause	5-7
5.2.1.2. BASED Clause	5-7
5.2.1.3. BLANK WHEN ZERO Clause	5-7
5.2.1.4. JUSTIFIED Clause	5-7
5.2.1.5. OCCURS Clause	5-8
5.2.1.6. PICTURE Clause	5-8
5.2.1.7. REDEFINES Clause	5-13
5.2.1.8. RENAMES Clause	5-14
5.2.1.9. SIGN Clause	5-14
5.2.1.10. SYNCHRONIZED Clause	5-15
5.2.1.11. USAGE Clause	5-15
5.2.1.12. VALUE Clause	5-18
5.2.2. Defining SCREEN SECTION Data Items	5-20
5.2.2.1. AUTO AUTO-SKIP AUTOTERMINATE Clause	5-21
5.2.2.2. BACKGROUND-COLOR Clause	5-21
5.2.2.3. BEEP BELL Clause	5-21
5.2.2.4. BLANK LINE and BLANK SCREEN Clauses	5-21
5.2.2.5. BLANK WHEN ZERO Clause	5-22
5.2.2.6. BLINK Clause	5-22
5.2.2.7. COLUMN Clause	5-22
5.2.2.8. ERASE EOL and ERASE EOS Clauses	5-22
5.2.2.9. FOREGROUND-COLOR Clause	5-23
5.2.2.10. FROM, TO and USING Clauses	5-23
5.2.2.11. FULL LENGTH-CHECK Clause	5-23
5.2.2.12. HIGHLIGHT and LOWLIGHT Clauses	5-23
5.2.2.13. JUSTIFIED Clause	5-23
5.2.2.14. LEFTLINE, OVERLINE and UNDERLINE Clauses	5-24
5.2.2.15. LINE Clause	5-24
5.2.2.16. OCCURS Clause	5-24
5.2.2.17. PICTURE Clause	5-24
5.2.2.18. PROMPT Clause	5-25
5.2.2.19. REQUIRED EMPTY-CHECK Clause	5-25
5.2.2.20. REVERSE-VIDEO Clause	5-25
5.2.2.21. SECURE NO-ECHO Clause	5-25
5.2.2.22. SIGN Clause	5-25
5.2.2.23. VALUE Clause	5-26
5.2.3. 01-Level Constant Descriptions	5-26
5.2.4. 66-Level Data Descriptions (RENAMES)	5-28
5.2.5. 77-Level Data Descriptions	5-28
5.2.6. 78-Level Constant Descriptions	5-28
5.2.7. 88-Level Condition Names	5-28
6. PROCEDURE DIVISION	6-1
6.1. General PROCEDURE DIVISION Components	6-1
6.1.1. Table References	6-1
6.1.2. Qualification of Data Names	6-4
6.1.3. Reference Modifiers	6-5
6.1.4. Expressions	6-5
6.1.4.1. Arithmetic Expressions	6-6
6.1.4.2. Conditional Expressions	6-8

6.1.5. Use of Periods (.)	6-11
6.1.6. Use of “VERB” / “END-VERB” Constructs	6-12
6.1.7. Intrinsic Functions	6-20
6.1.7.1. ABS(<i>number</i>)	6-20
6.1.7.2. ACOS(<i>cosine</i>)	6-20
6.1.7.3. ANNUITY(<i>interest-rate, number-of-periods</i>)	6-20
6.1.7.4. ASIN(<i>sine</i>)	6-20
6.1.7.5. ATAN(<i>tangent</i>)	6-21
6.1.7.6. BYTE-LENGTH(<i>string</i>)	6-21
6.1.7.7. CHAR(<i>integer</i>)	6-21
6.1.7.8. COMBINED-DATETIME(<i>days, seconds</i>)	6-21
6.1.7.9. CONCATENATE(<i>string-1 [, string-2] ...</i>)	6-21
6.1.7.10. COS(<i>angle</i>)	6-21
6.1.7.11. CURRENCY-SYMBOL	6-22
6.1.7.12. CURRENT-DATE	6-22
6.1.7.13. DATE-OF-INTEGGER(<i>integer</i>)	6-22
6.1.7.14. DATE-TO-YYYYMMDD(<i>yymmdd [, yy-cutoff]</i>)	6-22
6.1.7.15. DAY-OF-INTEGGER(<i>integer</i>)	6-22
6.1.7.16. DAY-TO-YYYYDDD(<i>yyddd [, yy-cutoff]</i>)	6-22
6.1.7.17. E	6-23
6.1.7.18. EXCEPTION-FILE	6-23
6.1.7.19. EXCEPTION-LOCATION	6-23
6.1.7.20. EXCEPTION-STATEMENT	6-23
6.1.7.21. EXCEPTION-STATUS	6-23
6.1.7.22. EXP(<i>number</i>)	6-24
6.1.7.23. EXP10(<i>number</i>)	6-24
6.1.7.24. FACTORIAL(<i>number</i>)	6-24
6.1.7.25. FRACTION-PART(<i>number</i>)	6-24
6.1.7.26. HIGHEST-ALGEBRAIC(<i>numeric-identifier</i>)	6-24
6.1.7.27. INTEGER(<i>number</i>)	6-24
6.1.7.28. INTEGER-OF-DATE(<i>date</i>)	6-24
6.1.7.29. INTEGER-OF-DAY(<i>date</i>)	6-24
6.1.7.30. INTEGER-PART(<i>number</i>)	6-24
6.1.7.31. LENGTH(<i>string</i>)	6-24
6.1.7.32. LENGTH-AN(<i>string</i>)	6-25
6.1.7.33. LOCALE-COMPARE(<i>argument-1, argument-2 [, locale]</i>)	6-25
6.1.7.34. LOCALE-DATE(<i>date [, locale]</i>)	6-25
6.1.7.35. LOCALE-TIME(<i>time [, locale]</i>)	6-25
6.1.7.36. LOCALE-TIME-FROM-SECS(<i>seconds [, locale]</i>)	6-25
6.1.7.37. LOG(<i>number</i>)	6-26
6.1.7.38. LOG10(<i>number</i>)	6-26
6.1.7.39. LOWER-CASE(<i>string</i>)	6-26
6.1.7.40. LOWEST-ALGEBRAIC(<i>numeric-identifier</i>)	6-26
6.1.7.41. MAX(<i>number-1 [, number-2] ...</i>)	6-26
6.1.7.42. MEAN(<i>number-1 [, number-2] ...</i>)	6-26
6.1.7.43. MEDIAN(<i>number-1 [, number-2] ...</i>)	6-26
6.1.7.44. MIDRANGE(<i>number-1 [, number-2] ...</i>)	6-26
6.1.7.45. MIN(<i>number-1 [, number-2] ...</i>)	6-26
6.1.7.46. MOD(<i>value, modulus</i>)	6-26
6.1.7.47. MODULE-CALLER-ID	6-26
6.1.7.48. MODULE-DATE	6-27
6.1.7.49. MODULE-FORMATTED-DATE	6-27
6.1.7.50. MODULE-ID	6-27
6.1.7.51. MODULE-PATH	6-27
6.1.7.52. MODULE-SOURCE	6-27
6.1.7.53. MODULE-TIME	6-27
6.1.7.54. MONETARY-DECIMAL-POINT	6-28
6.1.7.55. MONETARY-THOUSANDS-SEPARATOR	6-28
6.1.7.56. NUMERIC-DECIMAL-POINT	6-28
6.1.7.57. NUMERIC-THOUSANDS-SEPARATOR	6-29
6.1.7.58. NUMVAL(<i>string</i>)	6-29

6.1.7.59. NUMVAL-C(<i>string</i> [, <i>symbol</i>])	6-29
6.1.7.60. NUMVAL-F(<i>string</i>)	6-29
6.1.7.61. ORD(<i>char</i>)	6-30
6.1.7.62. ORD-MAX(<i>char-1</i> [, <i>char-2</i>] ...)	6-30
6.1.7.63. ORD-MIN(<i>char-1</i> [, <i>char-2</i>] ...)	6-30
6.1.7.64. PI	6-30
6.1.7.65. PRESENT-VALUE(<i>rate</i> , <i>value-1</i> [, <i>value-2</i>])	6-30
6.1.7.66. RANDOM [(<i>seed</i>)]	6-30
6.1.7.67. RANGE(<i>number-1</i> [, <i>number-2</i>] ...)	6-31
6.1.7.68. REM(<i>number</i> , <i>divisor</i>)	6-31
6.1.7.69. REVERSE(<i>string</i>)	6-31
6.1.7.70. SECONDS-FROM-FORMATTED-TIME(<i>format</i> , <i>time</i>)	6-31
6.1.7.71. SECONDS-PAST-MIDNIGHT	6-31
6.1.7.72. SIGN(<i>number</i>)	6-31
6.1.7.73. SIN(<i>angle</i>)	6-31
6.1.7.74. SQRT(<i>number</i>)	6-31
6.1.7.75. STANDARD-DEVIATION(<i>number-1</i> [, <i>number-2</i>] ...)	6-32
6.1.7.76. STORED-CHAR-LENGTH(<i>string</i>)	6-32
6.1.7.77. SUBSTITUTE(<i>string</i> , <i>from-1</i> , <i>to-1</i> [, <i>from-n</i> , <i>to-n</i>])	6-32
6.1.7.78. SUBSTITUTE-CASE(<i>string</i> , <i>from-1</i> , <i>to-1</i> [, <i>from-n</i> , <i>to-n</i>])	6-32
6.1.7.79. SUM(<i>number-1</i> [, <i>number-2</i>] ...)	6-32
6.1.7.80. TAN(<i>angle</i>)	6-32
6.1.7.81. TEST-DATE-YYYYMMDD(<i>date</i>)	6-32
6.1.7.82. TEST-DAY-YYYYDDD(<i>date</i>)	6-32
6.1.7.83. TEST-NUMVAL(<i>string</i>)	6-32
6.1.7.84. TEST-NUMVAL-C(<i>string</i> [, <i>symbol</i>])	6-32
6.1.7.85. TEST-NUMVAL-F(<i>string</i>)	6-33
6.1.7.86. TRIM(<i>string</i> [, LEADING TRAILING])	6-33
6.1.7.87. UPPER-CASE(<i>string</i>)	6-33
6.1.7.88. VARIANCE(<i>number-1</i> [, <i>number-2</i>] ...)	6-33
6.1.7.89. YEAR-TO-YYYY (<i>yy</i> [, <i>yy-cutoff</i>])	6-33
6.1.8. Special Registers	6-18
6.1.9. Controlling Concurrent Access to Files	6-13
6.1.9.1. File Sharing	6-13
6.1.9.2. Record Locking	6-14
6.1.10. Common Clauses On Executable Statements	6-14
6.1.10.1. AT END / NOT AT END	6-14
6.1.10.2. CORRESPONDING Option	6-15
6.1.10.3. INVALID KEY / NOT INVALID KEY	6-16
6.1.10.4. ON EXCEPTION / NOT ON EXCEPTION	6-16
6.1.10.5. ON OVERFLOW / NOT ON OVERFLOW	6-17
6.1.10.6. ON SIZE ERROR / NOT ON SIZE ERROR	6-17
6.1.10.7. Rounding Options	6-17
6.2. General Format of the PROCEDURE DIVISION	6-1
6.2.1. General Format for Subprogram Arguments	6-1
6.2.2. General Format for DECLARATIVES Procedures	6-3
6.3. PROCEDURE DIVISION Sections and Paragraphs	6-2
6.4. GNU COBOL Statements	6-18
6.4.1. ACCEPT	6-33
6.4.1.1. ACCEPT Format 1 – Read from Console	6-33
6.4.1.2. ACCEPT Format 2 – Retrieve Command-Line Arguments	6-34
6.4.1.3. ACCEPT Format 3 – Retrieve Environment Variable Values	6-34
6.4.1.4. ACCEPT Format 4 – Retrieve Full-Screen Data	6-36
6.4.1.5. ACCEPT Format 5 – Retrieve Date/Time	6-37
6.4.1.6. ACCEPT Format 6 - Retrieve Screen Information	6-38
6.4.1.7. ACCEPT Format 7 – Retrieve Run-Time Information	6-38
6.4.2. ADD	6-40
6.4.2.1. ADD Format 1 – ADD TO	6-40
6.4.2.2. ADD Format 2 – ADD GIVING	6-40
6.4.2.3. ADD Format 3 – ADD CORRESPONDING	6-41
6.4.3. ALLOCATE	6-42

6.4.4. ALTER	6-43
6.4.5. CALL	6-44
6.4.6. CANCEL	6-47
6.4.7. CLOSE	6-48
6.4.8. COMMIT	6-49
6.4.9. COMPUTE	6-50
6.4.10. CONTINUE	6-51
6.4.11. DELETE	6-52
6.4.12. DISPLAY.....	6-53
6.4.12.1. DISPLAY Format 1 – “UPON “ <i>device</i> ”	6-53
6.4.12.2. DISPLAY Format 2 – Access Command-Line Arguments	6-53
6.4.12.3. DISPLAY Format 3 – Access or Set Environment Variables	6-54
6.4.12.4. DISPLAY Format 4 – Screen Data	6-54
6.4.13. DIVIDE.....	6-56
6.4.13.1. DIVIDE Format 1 – DIVIDE INTO.....	6-56
6.4.13.2. DIVIDE Format 2 – DIVIDE INTO GIVING	6-56
6.4.13.3. DIVIDE Format 3 – DIVIDE BY GIVING	6-57
6.4.14. ENTRY	6-58
6.4.15. EVALUATE	6-59
6.4.16. EXIT	6-61
6.4.17. FREE	6-63
6.4.18. GENERATE	6-64
6.4.19. GOBACK	6-65
6.4.20. GO TO	6-66
6.4.20.1. GO TO Format 1 – Simple GO TO	6-66
6.4.20.2. GO TO Format 2 – GO TO DEPENDING ON	6-66
6.4.21. IF	6-67
6.4.22. INITIALIZE.....	6-68
6.4.23. INITIATE	6-71
6.4.24. INSPECT.....	6-72
6.4.24.1. TALLYING Clause Syntax, Rules and Operation	6-72
6.4.24.2. REPLACING Clause Syntax, Rules and Operation.....	6-73
6.4.24.3. CONVERTING Clause Syntax, Rules and Operation	6-74
6.4.24.4. INSPECT Region Clause, Rules and Operation	6-74
6.4.25. MERGE	6-76
6.4.26. MOVE.....	6-78
6.4.26.1. MOVE Format 1 – Simple MOVE.....	6-78
6.4.26.2. MOVE Format 2 – MOVE CORRESPONDING.....	6-78
6.4.27. MULTIPLY	6-79
6.4.27.1. MULTIPLY Format 1 – MULTIPLY BY.....	6-79
6.4.27.2. MULTIPLY Format 2 – MULTIPLY GIVING	6-79
6.4.28. NEXT SENTENCE	6-80
6.4.29. OPEN.....	6-81
6.4.30. PERFORM	6-83
6.4.30.1. PERFORM Format 1 – Procedural	6-83
6.4.30.2. PERFORM Format 2 – Inline.....	6-86
6.4.31. READ	6-88
6.4.31.1. READ Format 1 – Sequential READ	6-88
6.4.31.2. READ Format 2 – Random Read.....	6-88
6.4.32. READY TRACE	6-90
6.4.33. RELEASE	6-91
6.4.34. RESET TRACE	6-92
6.4.35. RETURN.....	6-93
6.4.36. REWRITE	6-94
6.4.37. ROLLBACK	6-95
6.4.38. SEARCH	6-96
6.4.38.1. SEARCH Format 1 – Sequential Search.....	6-96
6.4.38.2. SEARCH Format 2 – Binary, or Half-interval Search (SEARCH ALL).....	6-97
6.4.39. SET	6-99
6.4.39.1. SET Format 1 – SET ENVIRONMENT	6-99
6.4.39.2. SET Format 2 – SET Program-Pointer	6-99

6.4.39.3. SET Format 3 – SET ADDRESS.....	6-99
6.4.39.4. SET Format 4 – SET Index.....	6-100
6.4.39.5. SET Format 5 – SET UP/DOWN	6-100
6.4.39.6. SET Format 6 – SET Condition Name.....	6-100
6.4.39.7. SET Format 7 – SET Switch	6-100
6.4.39.8. SET Format 8 – SET ATTRIBUTE	6-101
6.4.40. SORT	6-101
6.4.40.1. SORT Format 1 – File-based Sort.....	6-101
6.4.40.2. SORT Format 2 – Table Sort	6-103
6.4.41. START.....	6-105
6.4.42. STOP	6-106
6.4.43. STRING	6-107
6.4.44. SUBTRACT	6-108
6.4.44.1. SUBTRACT Format 1 – SUBTRACT FROM.....	6-108
6.4.44.2. SUBTRACT Format 2 – SUBTRACT GIVING.....	6-108
6.4.44.3. SUBTRACT Format 3 – SUBTRACT CORRESPONDING	6-109
6.4.45. SUPPRESS.....	6-110
6.4.46. TERMINATE	6-111
6.4.47. TRANSFORM	6-112
6.4.48. UNLOCK	6-113
6.4.49. UNSTRING	6-114
6.4.50. WRITE	6-116
7. SUB-PROGRAMMING WITH GNU COBOL	7-1
7.1. Subprograms, Subroutines and User-Defined Functions	7-1
7.2. Specifying and Using Alternate Entry Points	7-1
7.3. Dynamic Versus Static Subprograms.....	7-1
7.4. Subprogram Execution Flow	7-2
7.4.1. Subroutine Execution Flow	7-2
7.4.2. User-Defined Function Execution Flow.....	7-3
7.5. Sharing Data Between Calling and Called Programs.....	7-4
7.5.1. Subprogram Arguments	7-4
7.5.1.1. Calling Program Considerations.....	7-4
7.5.1.2. Called Program Considerations	7-5
7.5.2. GLOBAL Data Items	7-5
7.5.3. EXTERNAL Data Items	7-6
7.6. Nested Subprograms	7-6
7.7. Recursive GNU COBOL Subprograms	7-7
7.8. Combining COBOL and C Programs	7-8
7.8.1. GNU COBOL Run-Time Library Requirements.....	7-8
7.8.2. String Allocation Differences Between GNU COBOL and C	7-8
7.8.3. Matching C Data Types with GNU COBOL USAGES	7-9
7.8.4. GNU COBOL Main Programs CALLING C Subprograms	7-11
7.8.5. C Main Programs CALLING GNU COBOL Subprograms	7-12
8. THE GNU COBOL SYSTEM INTERFACE.....	8-1
8.1. Using the GNU COBOL Compiler (cobc).....	8-1
8.1.1. Introduction.....	8-1
8.1.2. Syntax and Options	8-1
8.1.3. Compiling GNU COBOL Programs	8-3
8.1.3.1. Compiling Directly-Executable GNU COBOL Programs.....	8-3
8.1.3.2. Compiling Dynamically-Loadable GNU COBOL Subprograms.....	8-3
8.1.3.3. Compiling Static GNU COBOL Subprograms.....	8-3
8.1.4. Important Compilation-Time Environment Variables.....	8-4
8.1.5. Locating Copybooks at Compilation Time	8-5
8.1.6. Using Compiler Configuration Files	8-5
8.2. Running GNU COBOL Programs	8-7
8.2.1. Executing Programs Directly	8-7
8.2.2. Using the “cobcrun” Utility	8-7
8.2.3. Program Arguments.....	8-8

8.2.4. Important Execution-Time Environment Variables.....	8-8
8.3. Built-In System Subroutines.....	8-10
8.3.1. "Call by Name" Routines.....	8-10
8.3.1.1. CALL "C\$CALLED BY" USING <i>prog-name-area</i>	8-11
8.3.1.2. CALL "C\$CHDIR" USING <i>directory-path, result</i>	8-11
8.3.1.3. CALL "C\$COPY" USING <i>src-file-path, dest-file-path, 0</i>	8-11
8.3.1.4. CALL "C\$DELETE" USING <i>file-path, 0</i>	8-12
8.3.1.5. CALL "C\$FILEINFO" USING <i>file-path, file-info</i>	8-12
8.3.1.6. CALL "C\$GETPID".....	8-12
8.3.1.7. CALL "C\$JUSTIFY" USING <i>data-item, "justification-type"</i>	8-12
8.3.1.8. CALL "C\$MAKEDIR" USING <i>dir-path</i>	8-12
8.3.1.9. CALL "C\$NARG" USING <i>arg-count-result</i>	8-13
8.3.1.10. CALL "C\$PARAMSIZE" USING <i>argument-number</i>	8-13
8.3.1.11. CALL "C\$PRINTABLE" USING <i>data-item [, char]</i>	8-13
8.3.1.12. CALL "C\$SLEEP" USING <i>seconds-to-sleep</i>	8-13
8.3.1.13. CALL "C\$TOLOWER" USING <i>data-item, BY VALUE convert-length</i>	8-13
8.3.1.14. CALL "C\$TOUPPER" USING <i>data-item, BY VALUE convert-length</i>	8-13
8.3.1.15. CALL "CBL_AND" USING <i>item-1, item-2, BY VALUE byte-length</i>	8-13
8.3.1.16. CALL "CBL_CHANGE_DIR" USING <i>directory-path</i>	8-14
8.3.1.17. CALL "CBL_CHECK_FILE_EXIST" USING <i>file-path, file-info</i>	8-14
8.3.1.18. CALL "CBL_CLOSE_FILE" USING <i>file-handle</i>	8-14
8.3.1.19. CALL "CBL_COPY_FILE" USING <i>src-file-path, dest-file-path</i>	8-14
8.3.1.20. CALL "CBL_CREATE_DIR" USING <i>dir-path</i>	8-15
8.3.1.21. CALL "CBL_CREATE_FILE" USING <i>file-path, 2, 0, 0, file-handle</i>	8-15
8.3.1.22. CALL "CBL_DELETE_DIR" USING <i>dir-path</i>	8-15
8.3.1.23. CALL "CBL_DELETE_FILE" USING <i>file-path</i>	8-15
8.3.1.24. CALL "CBL_ERROR_PROC" USING <i>function, program-pointer</i>	8-15
8.3.1.25. CALL "CBL_EXIT_PROC" USING <i>function, program-pointer</i>	8-16
8.3.1.26. CALL "CBL_EQ" USING <i>item-1, item-2, BY VALUE byte-length</i>	8-17
8.3.1.27. CALL "CBL_FLUSH_FILE" USING <i>file-handle</i>	8-18
8.3.1.28. CALL "CBL_GET_CURRENT_DIR" USING <i>BY VALUE 0, BY VALUE length, BY REFERENCE buffer</i>	8-18
8.3.1.29. CALL "CBL_GET_CSR_POS" USING <i>cursor-locn-buffer</i>	8-18
8.3.1.30. CALL "CBL_GET_SCR_SIZE" USING <i>no-of-lines, no-of-cols</i>	8-18
8.3.1.31. CALL "CBL_IMP" USING <i>item-1, item-2, BY VALUE byte-length</i>	8-19
8.3.1.32. CALL "CBL_NIMP" USING <i>item-1, item-2, BY VALUE byte-length</i>	8-19
8.3.1.33. CALL "CBL_NOR" USING <i>item-1, item-2, BY VALUE byte-length</i>	8-19
8.3.1.34. CALL "CBL_NOT" USING <i>item-1, BY VALUE byte-length</i>	8-20
8.3.1.35. CALL "CBL_OC_NANOSLEEP" USING <i>nanoseconds-to-sleep</i>	8-20
8.3.1.36. CALL "CBL_OPEN_FILE" <i>file-path, access-mode, 0, 0, handle</i>	8-20
8.3.1.37. CALL "CBL_OR" USING <i>item-1, item-2, BY VALUE byte-length</i>	8-20
8.3.1.38. CALL "CBL_READ_FILE" USING <i>handle, offset, nbytes, flag, buffer</i>	8-21
8.3.1.39. CALL "CBL_RENAME_FILE" USING <i>old-file-path, new-file-path</i>	8-21
8.3.1.40. CALL "CBL_TOLOWER" USING <i>data-item, BY VALUE convert-length</i>	8-21
8.3.1.41. CALL "CBL_TOUPPER" USING <i>data-item, BY VALUE convert-length</i>	8-21
8.3.1.42. CALL "CBL_WRITE_FILE" USING <i>handle, offset, nbytes, 0, buffer</i>	8-22
8.3.1.43. CALL "CBL_XOR" USING <i>item-1, item-2, BY VALUE byte-length</i>	8-22
8.3.1.44. CALL "SYSTEM" USING <i>command</i>	8-22
8.3.2. "Call by Number" Subroutines.....	8-22
8.3.2.1. CALL X"91" USING <i>return-code, function-code, binary-variable-arg</i>	8-22
8.3.2.2. CALL X"E4".....	8-23
8.3.2.3. CALL X"E5".....	8-23
8.3.2.4. CALL X"F4" USING <i>byte, table</i>	8-23
8.3.2.5. CALL X"F5" USING <i>byte, table</i>	8-24
8.4. Binary Truncation.....	8-25
9. SAMPLE PROGRAMS.....	10-1
9.1. FileStat-Msgs.cpy – File Status Values.....	10-1
9.2. COBDUMP – A Hex/Char Data Dump Subroutine.....	10-2
9.3. DAY-FROM-DATE – Determine Day of Week From a Date.....	10-8
9.4. GCic – an Interactive GNU COBOL Full-Screen Compiler Front-End.....	10-12
9.5. STREAMIO – A Utility Subroutine to Simplify Stream I/O.....	10-106

9.6. WINSYSTEM – Execute Windows Shell Commands (For Cygwin).....	Error! Bookmark not defined.
10. GLOSSARY OF TERMS	11-1
INDEX	I
GNU FREE DOCUMENTATION LICENSE	IX

Figures

Figure 1-1 - A Sample TUI Screen	1-12
Figure 1-2 – General Format of a GNU COBOL Program	1-15
Figure 1-3 - Figurative Constants.....	1-18
Figure 2-1 - COPY Syntax	2-1
Figure 2-2 - REPLACE (Format 1) Syntax	2-2
Figure 2-3 - REPLACE (Format 2) Syntax	2-2
Figure 2-4 - >>DEFINE Syntax	2-2
Figure 2-5 - >>IF Syntax	2-3
Figure 2-6 - >>IF constant-conditional-expression Format	2-3
Figure 2-7 - >>SET Syntax	2-4
Figure 2-8 - >>SOURCE Syntax	2-5
Figure 2-9 - >>TURN Syntax	2-5
Figure 3-1 - IDENTIFICATION DIVISION Syntax	3-1
Figure 4-1 - ENVIRONMENT DIVISION Syntax.....	4-1
Figure 4-2 - CONFIGURATION SECTION Syntax	4-1
Figure 4-3 - SOURCE-COMPUTER Paragraph Syntax	4-1
Figure 4-4 - OBJECT-COMPUTER Paragraph Syntax.....	4-2
Figure 4-5 - REPOSITORY Paragraph Syntax	4-3
Figure 4-6 - SPECIAL-NAMES Paragraph Syntax.....	4-3
Figure 4-7 – Typical Locale Codes	4-5
Figure 4-8 - Built-In GNU COBOL Device Names.....	4-5
Figure 4-9 - The SPECIAL-NAMES "alphabet-name" Clause	4-6
Figure 4-10 - The SPECIAL-NAMES "class-name" Clause	4-6
Figure 4-11 - The SPECIAL-NAMES "switch-definition" Clause.....	4-7
Figure 4-12 - The SPECIAL-NAMES "symbolic-characters" Clause.....	4-7
Figure 4-13 - INPUT-OUTPUT SECTION Syntax	4-8
Figure 4-14 – File SELECT Statement Syntax.....	4-9
Figure 4-15 – FILE STATUS Values.....	4-11
Figure 4-16 - SELECT “organization-options” For SEQUENTIAL Files	4-11
Figure 4-17 - SELECT "organization-options" for LINE SEQUENTIAL Files.....	4-12
Figure 4-18 - SELECT “organization options” For RELATIVE Files	4-13
Figure 4-19 - SELECT “organization options” For INDEXED Files	4-14
Figure 4-20 - I-O-CONTROL Paragraph Syntax.....	4-15
Figure 5-1 - General DATA DIVISION Format.....	5-1
Figure 5-2 - File Description (FD) and Sort Description (SD) Syntax	5-2
Figure 5-3- LINAGE-specified Page Structure	5-3
Figure 5-4 – Non-SCREEN SECTION Data Item Description Syntax.....	5-6
Figure 5-5 - Data Class-Specification PICTURE Symbols (A/X/9).....	5-8
Figure 5-6 - Numeric Option PICTURE Symbols (P/S/V)	5-9
Figure 5-7 - Numeric Editing PICTURE Symbols	5-10
Figure 5-8 - Sign-Encoding Characters.....	5-14
Figure 5-9 - Effect of the SYNCHRONIZED Clause	5-15
Figure 5-10 - Summary of USAGE Specifications	5-15
Figure 5-11 - SCREEN SECTION Data Item Description Syntax	5-20
Figure 5-12 - The GNU COBOL Color Palette (Windows Console)	5-21
Figure 5-13 - 01-Level Constant Description Syntax.....	5-26
Figure 5-14 - 66-Level Data Description Syntax.....	5-28
Figure 5-15 - 78-Level Constant Description Syntax.....	5-28
Figure 5-16 - 88-Level Condition Name Syntas.....	5-29
Figure 6-1 - Reference Modifier Syntax.....	6-5
Figure 6-2 – Unary “Minus” (-) Operator Syntax	6-6

Figure 6-3 – Unary “Plus” (+) Operator Syntax	6-6
Figure 6-4 - Exponentiation Operator (** or ^) Syntax	6-6
Figure 6-5 - Multiplication Operator (*) Syntax.....	6-6
Figure 6-6 - Division Operator (/) Syntax.....	6-6
Figure 6-7 - Addition Operator (+) Syntax	6-7
Figure 6-8 - Subtraction Operator (-) Syntax	6-7
Figure 6-9 - Class Condition Syntax	6-8
Figure 6-10 - Sign Condition Syntax.....	6-9
Figure 6-11 - Using Switch Conditions	6-9
Figure 6-12 - Relation Condition Syntax	6-10
Figure 6-13 - Combined Condition Syntax	6-10
Figure 6-14 - Negated Condition Syntax.....	6-11
Figure 6-15 - Special Registers.....	6-19
Figure 6-16 - ROUNDED MODE Behavior	6-18
Figure 6-17 - General PROCEDURE DIVISION Syntax.....	6-1
Figure 6-18 - Syntax of a PROCEDURE DIVISION USING Argument	6-1
Figure 6-19 - General DECLARATIVES Procedure Syntax	6-3
Figure 6-20 - ACCEPT (Read from Console) Syntax.....	6-34
Figure 6-21 - ACCEPT (Command Line Arguments) Syntax.....	6-34
Figure 6-22 - ACCEPT (Environment Variable Values) Syntax.....	6-35
Figure 6-23 - ACCEPT (Retrieve Screen Data) Syntax.....	6-36
Figure 6-24 - Screen ACCEPT CRT STATUS Codes	6-37
Figure 6-25 - ACCEPT (Retrieve Date/Time) Syntax.....	6-37
Figure 6-26 - ACCEPT Options for DATE/TIME Retrieval	6-37
Figure 6-27 - ACCEPT (Retrieve Screen Information) Syntax.....	6-38
Figure 6-28 - ACCEPT (Retrieve Run-Time Information) Syntax	6-38
Figure 6-29 - Run-Time Exception Code Values.....	6-39
Figure 6-30 - ADD (TO) Syntax	6-40
Figure 6-31 - ADD (GIVING) Syntax.....	6-40
Figure 6-32 - ADD (CORRESPONDING) Syntax	6-41
Figure 6-33 - ALLOCATE Syntax.....	6-42
Figure 6-34 - ALTER Syntax	6-43
Figure 6-35 - CALL Syntax	6-44
Figure 6-36 - Argument Format When CALL ing a Subroutine	6-45
Figure 6-37 - CANCEL Syntax	6-47
Figure 6-38 - CLOSE Syntax.....	6-48
Figure 6-39 - COMMIT Syntax.....	6-49
Figure 6-40 - COMPUTE Syntax	6-50
Figure 6-41 - CONTINUE Syntax.....	6-51
Figure 6-42 - DELETE Syntax	6-52
Figure 6-43 - DISPLAY (Upon Console) Syntax	6-53
Figure 6-44 - DISPLAY (Access Command-line Arguments) Syntax.....	6-53
Figure 6-45 - DISPLAY (Access / Set Environment Variables) Syntax	6-54
Figure 6-46 - DISPLAY (Screen Data) Syntax	6-54
Figure 6-47 - DIVIDE INTO Syntax	6-56
Figure 6-48 - DIVIDE INTO GIVING Syntax	6-56
Figure 6-49 - DIVIDE BY GIVING Syntax	6-57
Figure 6-50 - ENTRY Syntax.....	6-58
Figure 6-51 - ENTRY Statement Argument Syntax.....	6-58
Figure 6-52 - EVALUATE Syntax	6-59
Figure 6-53 - EXIT Syntax	6-61
Figure 6-54 - Using the EXIT Statement.....	6-61
Figure 6-55 - Using EXIT PARAGRAPH	6-61
Figure 6-56 - Using the EXIT PERFORM Statement.....	6-62
Figure 6-57 - FREE Syntax	6-63
Figure 6-58 - GENERATE Syntax.....	6-64
Figure 6-59 - GOBACK Syntax	6-65
Figure 6-60 - Simple GO TO Syntax.....	6-66
Figure 6-61 – GO TO DEPENDING ON Syntax	6-66
Figure 6-62 - GOTO DEPENDING ON vs IF vs EVALUATE.....	6-66
Figure 6-63 - IF Syntax	6-67

Figure 6-64 - INITIALIZE Syntax.....	6-68
Figure 6-65 - INITIATE Syntax	6-71
Figure 6-66 - INSPECT Syntax.....	6-72
Figure 6-67 - An INSPECT TALLYING Example.....	6-73
Figure 6-68 - MERGE Syntax	6-76
Figure 6-69 - Simple MOVE Syntax.....	6-78
Figure 6-70 - MOVE CORRESPONDING Syntax	6-78
Figure 6-71 - MULTIPLY BY Syntax.....	6-79
Figure 6-72 - MULTIPLY GIVING Syntax	6-79
Figure 6-73 - NEXT SENTENCE Syntax.....	6-80
Figure 6-74 - OPEN Syntax.....	6-81
Figure 6-75 - Procedural PERFORM Syntax	6-83
Figure 6-76 - Simple PERFORM.....	6-83
Figure 6-77 - PERFORM UNTIL EXIT	6-84
Figure 6-78 - PERFORM <i>n</i> TIMES.....	6-84
Figure 6-79 - PERFORM UNTIL.....	6-85
Figure 6-80 - PERFORM VARYING AFTER.....	6-85
Figure 6-81 - Inline PERFORM Syntax	6-87
Figure 6-82 - READ (Sequential) Syntax	6-88
Figure 6-83 - READ (Random) Syntax	6-89
Figure 6-84 - READY TRACE Syntax.....	6-90
Figure 6-85 - RELEASE Syntax	6-91
Figure 6-86 - RESET TRACE Syntax.....	6-92
Figure 6-87 - RETURN Syntax.....	6-93
Figure 6-88 - REWRITE Syntax	6-94
Figure 6-89 - ROLLBACK Syntax	6-95
Figure 6-90 - Sequential SEARCH Syntax	6-96
Figure 6-91 - Binary SEARCH (ALL) Syntax	6-97
Figure 6-92 - SET ENVIRONMENT Syntax.....	6-99
Figure 6-93 - SET Program Pointer Syntax.....	6-99
Figure 6-94 - SET ADDRESS Syntax.....	6-99
Figure 6-95 - SET Index Syntax.....	6-100
Figure 6-96 - SET UP/DOWN Syntax	6-100
Figure 6-97 - SET Condition Name Syntax	6-100
Figure 6-98 - SET Switch Syntax.....	6-101
Figure 6-99 - SET ATTRIBUTE Syntax.....	6-101
Figure 6-100 - File-Based SORT Syntax	6-101
Figure 6-101 - Table SORT Syntax.....	6-104
Figure 6-102 - START Syntax.....	6-105
Figure 6-103 - STOP Syntax.....	6-106
Figure 6-104 - STRING Syntax	6-107
Figure 6-105 - SUBTRACT FROM Syntax	6-108
Figure 6-106 - SUBTRACT GIVING Syntax	6-108
Figure 6-107 - SUBTRACT CORRESPONDING Syntax.....	6-109
Figure 6-108 - SUPPRESS Syntax.....	6-110
Figure 6-109 - TERMINATE Syntax	6-111
Figure 6-110 - TRANSFORM Syntax	6-112
Figure 6-111 - UNLOCK Syntax	6-113
Figure 6-112 - UNSTRING Syntax.....	6-114
Figure 6-113 - WRITE Syntax	6-116
Figure 7-1 - C/GNU COBOL Data Type Matches	7-9
Figure 7-2 - GNU COBOL CALL ing C	7-11
Figure 7-3 - C CALL ing GNU COBOL	7-12
Figure 8-1 - Compiler Environment Variables	8-4
Figure 8-2 - Run-Time Environment Variables.....	8-8
Figure 8-3 - A Binary Truncation Demo Program	8-25
Figure 8-4 - A Non-Scientific Comparison of Numeric Data Item USAGE Performance	8-27

1. Introduction

1.1. What is GNU COBOL?

This document describes the syntax, semantics and usage of the COBOL programming language as implemented by the current version of GNU COBOL, formerly known as OpenCOBOL.

GNU COBOL is an open-source COBOL compiler and runtime environment. The GNU COBOL compiler generates C code which is automatically compiled and linked. While originally developed for UNIX operating systems, GNU COBOL has also been successfully built for OSX computers or Windows computers utilizing the UNIX-emulation features of such tools as Cygwin and MinGW¹. It has also been built as a truly native Windows application utilizing Microsoft's freely-downloadable Visual Studio Express package to provide the C compiler and linker/loader.

The principal developers of GNU COBOL are Keisuke Nishida and Roger While. They may be contacted at the GNU COBOL website - www.gnu-cobol.org.

This document was intended to serve as a full-function reference and user's guide suitable for both those readers learning COBOL for the first time as well as those already familiar with some dialect of the COBOL language. The author of this document is Gary Cutler, who may be reached via postings at the www.gnu-cobol.org forum, or by email at CutlerGL@gmail.com.

1.2. Additional References and Documents

For those wishing to learn COBOL for the first time, I can strongly recommend the following resources.

If you like to hold a book in your hands, I strongly recommend "Murach's Structured COBOL", by Mike Murach, Anne Prince and Raul Menendez (2000) - ISBN 9781890774059. Mike Murach and his various writing partners have been writing outstanding COBOL textbooks for decades, and this text is no exception. It's an excellent book for those familiar with the concepts of programming in other languages, but unfamiliar with COBOL.

Would you prefer a web-based tutorial? Try the University of Limerick (Ireland) COBOL web site - <http://www.csis.ul.ie/cobol/>.

1.3. Introducing COBOL

If you already know a programming language, and that language isn't COBOL, chances are that language is Java, C or C++. You will find COBOL a much different programming language than those – sometimes those differences are a good thing and sometimes they aren't. The thing to remember about COBOL is this – it was designed to solve business problems. It was designed to do that in the 1950s.

COBOL was the first programming language to become standardized such that a COBOL program written on computer "A" made by company "X" would be able to be compiled and executed on computer "B" made by company "Y". This may not seem like such a big deal today, but it was a radical departure from all programming languages that came before it and even many that came after it.

The name "COBOL" actually says it all – COBOL is an acronym that stands for "COMmon Business Oriented Language". Note the fact that the word "common" comes before all others. The word "business" is a close second. Therein lies the key to COBOL's success.

1.3.1. "I Heard COBOL is a Dead Language!"

Phoenician is a dead language. Mohegan is a dead language. Sanskrit is a dead language. What makes these languages dead is the fact that no one speaks them anymore. COBOL is NOT a dead language, and despite pontifications that come down to us from the ivory towers of academia, it isn't even on life support.

What made those other languages die is the fact that they became both obsolete and irrelevant. As the peoples that spoke them were overrun or superseded by other populations that eventually replaced them, no one saw any need to speak their languages.

¹ The MinGW approach is a personal favorite with the author of this manual because it creates a GNU COBOL compiler and runtime that require only a single MinGW DLL to be available to GNU COBOL tools and user programs. That DLL is freely distributable under the terms of the GNU General Public License. A MinGW build of GNU COBOL fits easily on and runs from a 128MB flash drive with no need to install any software onto the Windows computer that will be using it. Some functionality of the language, dealing with the sharing of files between concurrently executing GNU COBOL programs and record locking on certain types of files, is sacrificed however as the underlying operating system routines needed to implement them aren't available to Windows.

COBOL is different. Certainly, there were more people that “spoke” COBOL back in the 1980s than there are now. Remember, however, the second word in COBOL’s acronym – business. Businesses are complex social and economic organisms that exist for but a single purpose – to make money. One of the approaches businesses take to satisfy that all-important survival trait is the avoidance of unnecessary expenses.

This avoidance of expense turns out to have been key to the survival of COBOL because those programmers of the 1980s (give or take a decade) were very busy programmers. Estimates are that as many as several hundred billion lines of COBOL code were written for businesses world-wide. Because of the first word in COBOL’s name (“Common”), as businesses replaced their older, slower and less-reliable computer systems with newer, faster and more-reliable ones, they found that the massive investment they had in their COBOL software inventory paid dividends by remaining functional on those new systems - many times with no changes needed whatsoever!

Unwilling to endorse change merely for the sake of change, businesses replaced COBOL code only when absolutely necessary and only when financially justifiable. That justification appeared to have come as the 20th century was nearing the end.

Written long before the end of the century was near, many COBOL applications used 2-digit years instead of four digit years because, when the programs were written, computer storage of any kind was expensive. Why should millions and millions of bytes of storage be wasted by all those “19” sequences when the software can just simply assume them? Since their software would suddenly think the current year was “1900” after the stroke of midnight, December 31st 1999, businesses knew they were going to have to do something about the “Y2K” (programmer “geek speak” for “Year 2000”) problem.

At last! Y2K was going to be the massive asteroid strike that finally killed off the COBOL dinosaur.

Unfortunately for those seeking the extinction of COBOL, that proved to be wishful thinking.

Always concerned with the bottom line, businesses actually analyzed the problems with their programs. Many applications were replaced with newer and “better” versions that used more appropriate (translation: more politically correct) languages and computer systems. BUT ... many applications were not replaced. These were the absolutely essential applications whose replacement would cripple the business if everything didn’t go absolutely perfectly. These COBOL applications were modified to use 4-digit years instead of 2-digit ones. At the same time, many of them received cosmetic “face lifts” to make their computer/human interfaces more acceptable, frequently with the help of modules developed in the newer languages.

The result is that even today, after the Y2K “extinction event”, there are, by some industry estimates, over 220 billion lines of COBOL code still running the businesses of the 21st century. A fact that is disturbing to some is that – just as tiny little furry mammals evolved to cope with the original “extinction event” holocaust – COBOL has also evolved into a leaner and meaner “animal” capable of competing in niches and providing services unthought-of back in 1968. That fact is confirmed by the fact that those lines of COBOL code being tracked by industry analysts are actually *growing* at the rate of about 4 billion a year.

Evolution, you see, is in COBOLs DNA. Over time, COBOL evolved in form and function, first via work done by the American National Standards Institute (ANSI) and eventually through the efforts of the International Standards Organization (ISO).

The first widely-adopted standard for COBOL was published by ANSI in 1968². Named the ANS68 standard, this version of COBOL was originally standardized for use primarily as the business programming tool of the US Defense Department; it quickly was adopted by other Government agencies and private businesses alike.

Subsequent standards published in 1974 and 1985 (ANS74 and ANS85, respectively) added new features and evolved the language toward adoption of the programmer-productivity tool of the time – “Structured Programming”.

As the 21st century dawned, programming had moved out of the board room and into the Game Room, the Living Room and even the Kitchen. As computers became more and more inexpensive they appeared in games, entertainment devices and appliances. Even the automobile became home to computers galore. These computers need software, and that software is written in the so-called “modern” languages.

Combined with Y2K, these trends became the impetus for COBOL to evolve even newer features and capabilities. The COBOL2002 standard³ introduced object-oriented features and syntax that make the language more

² To that point, in 1968 the US Government made it a requirement that any computer system sold to them must run a version of COBOL that adhered to the ANS68 standard. The requirement that computers sold to the US Government had to support the current COBOL standard remained for many, many years.

³ “Popular” names for COBOL standards no longer include an organization’s name, and now use Y2K-compliant 4-digit years.

programmer-friendly to those trained by today's programming curricula. The COBOL20xx standard, currently under development, carries the evolution forward to the point where a COBOL20xx implementation will be fully as "modern" as any other programming language.

Through all this evolution, however, care was taken with each new standard to protect the investment businesses (or anyone, for that matter) had in COBOL software. Generally, a new COBOL standard – once implemented and adopted by a business - required minimal, if any, changes to existing applications. When changes were necessary, those changes could frequently be made using tools that mechanically upgraded entire libraries of source code with little or no need for human intervention.

The GNU COBOL implementation of the COBOL language supports virtually the entire ANS85 standard as well as some significant features of the COBOL2002 standard, although the truly object-oriented features are not there (yet).

1.3.2. Programmer Productivity – The “Holy Grail”

Throughout the history of computer programming, the search for new ways to improve of the productivity of programmers has been the all-important consideration. Sometimes this search has taken the form of introducing new features in programming languages, or even new languages altogether. Sometimes it has evolved new ways of using the existing languages. Other than hobbyists, programming is an activity performed for money. Businesses abhor spending anything more than is absolutely necessary. Even government agencies try to spend as little money on projects as is absolutely necessary⁴.

The amount of programming necessary to accomplish a given task – including rework needed by any errors found during testing (*testing*: “that time during which an application is actually in production use attempting to serve the purpose for which it was designed” ☺) is the measure of *programmer productivity*. Anything that reduces that effort will therefore reduce the time spent in such activities therefore reducing the expense of same. When the expense of programming is reduced, programmer productivity is increased.

While many technological and procedural developments have made evolutionary improvements to programmer productivity, each of the following has been responsible for revolutionary improvements:

- ▶ The development of so-called “higher-level” programming languages that enable a programmer to specify in a single statement of the language an action that would have required many more separate statements in a prior programming language. The standardization of such languages, making them usable on a wide variety of computers and operating systems, was a key aspect of this development. COBOL was a pioneering development in this area, being one of the first higher-level languages and the first to become standardized.
- ▶ The establishment of programming techniques that make programs easier to read and therefore easier to understand. Not only do such techniques reduce the amount of rework necessary simply to make a program work as designed, but they also reduce the amount of time a programmer needs to study an existing program in order how to best adapt it to changing business requirements. The foremost development in this area was *structured programming*. Introduced in the late 1970s, this approach to programming spawned new programming languages (PASCAL, ALGOL, PL/1) designed around it. With the ANSI85 standard, COBOL embraced the principles espoused by structured programming mavens as well as any of the languages designed strictly around it.
- ▶ The establishment of programming techniques AND the introduction of programming language capabilities to facilitate the reusability of program code. Anything that supports *code reusability* can have a profound impact to the amount of time it takes to develop new applications or to make significant changes to existing ones. In recent years, object-oriented programming has been the industry “poster child” for code reusability. By enabling program logic and the data structures that logic manipulates to be encapsulated into easily stored and retrieved (and therefore “reusable”) modules called *classes*, the object-oriented languages such as Java, C++ and C# have become the favorites of academia. Since students are being trained in these technologies and only these, by and large, it's no surprise that – today - object-oriented programming languages are the darlings of the industry.

The reality is, however, that good programmers have been practicing code reusability for more than a half-century. Up until recently, COBOL programmers have had some of the best code reusability tools available - they've been doing it with copybooks and subprograms rather than classes, methods and

⁴ This is a religious issue because it is an assertion that – sadly – must be taken purely on faith; there is, unfortunately, all too little real-world evidence to support it. It makes sense though, so one can only hope it is true.

attributes but the net results have been similar. With the COBOL2002 standard and the improvements made by the COBOL20xx standard, the playing field is rapidly becoming leveled in this regard.

1.3.3. Notable COBOL/GNU COBOL Features

1.3.3.1. Basic Program Readability

The most vociferous critics of COBOL always focus on the wordiness of the language, often citing the case of an infamous “Hello World” program as the “proof” that COBOL is so much more tedious to program in than more “modern” languages. This tedium is cited as such a significant impact to programmer productivity that – in their minds – the critics believe that COBOL can’t go away quickly enough for them.

Here are two different “Hello World” applications – one written in Java and the second in COBOL2002:

Java “Hello World”	COBOL2002 “Hello World” (Free-form Mode) ⁵
<pre>Class HelloWorld { public static void main(String[] args) { System.out.println("Hello World!"); } }</pre>	<pre>identification division. program-id. HelloWorld. procedure division. display "Hello World!".</pre>

Both programs could have been written on a single line, if desired, and both languages allow a programmer to use (or not use) indentation as they see fit to improve program readability. Sounds like a tie so far.

Let’s look at how much more “wordy” COBOL is than Java. Count the characters in the two programs. The Java program has 95 (not counting carriage returns and any indentation). The COBOL program has 89 (again, not counting carriage returns and indentation)! Technically, it could have been only 65 because the “identification division.” header is actually optional.

Clearly, “Hello World” doesn’t look any better in Java than it does in COBOL.

Let’s look at a different problem. Surely a program that asks a user to input a positive integer, generates the sum of all positive integers from 1 to that number and then prints the result will be MUCH easier to code in Java than in COBOL, right?

You can be the judge.

Java Sum of Integers	COBOL2002 Sum of Integers (Free-form Mode) ⁶
<pre>import java.util.Scanner; public class sumofintegers { public static void main(String[] arg) { System.out.println("Enter a positive integer"); Scanner scan=new Scanner(System.in); int n=scan.nextInt(); int sum=0; for (int i=1;i<=n;i++) { sum=sum+i; } System.out.println("The sum is "+sum); } }</pre>	<pre>identification division. program-id. sumofintegers. data division. working-storage section. 01 n binary-int. 01 i binary-int. 01 sum binary-int. procedure division. display "Enter a positive integer" accept n perform varying i from 1 by 1 until i>n add i to sum end-perform display "The sum is " sum.</pre>

My familiarity with COBOL may be prejudicing my opinion, but it doesn’t appear to me that the Java code is any simpler than the COBOL code. In case you’re interested in character counts, the Java code comes in at 281 (not counting indentation characters). The COBOL code is 287 (263 without the “identification division.” header).

The more complex the programming logic being implemented, the more concise the Java code will appear to be, even compared to 2002-standard COBOL. That conciseness comes with a price though – program code readability. Java (or C or C++ or C#) programs are generally intelligible only to trained programmers. COBOL

⁵ One of the features of the COBOL2002 standard is its ability to allow programs to be coded in free-form mode, where line breaks and indentation are pretty much left to the discretion of the programmer. It wasn’t always this way, and the pre-2002 standards for COBOL are quite rigid when it comes to that sort of thing. Maybe the COBOL critics

⁶ One of the features of the COBOL2002 standard is its ability to allow programs to be coded in free-form mode, where line breaks and indentation are pretty much left to the discretion of the programmer. It wasn’t always this way, and the pre-2002 standards for COBOL are quite rigid when it comes to that sort of thing. Maybe the COBOL critics

programs can be quite intelligible to non-programmers, however. This is actually a side-effect of the wordiness of the language, where COBOL statements use natural English words to describe their actions. This inherent readability has come in handy many times throughout my career when I've had to learn obscure business (or legal) processes by reading COBOL program code that supports them.

The “modern” languages, like COBOL, also have their own “boilerplate” infrastructure overhead that must be coded in order to write the logic that is necessary in the program. Take for example the “`public static void main(String[] arg) {`” and “`import java.util.Scanner;`” statements. The critics tend to forget about this when they criticize COBOL for its structural “overhead.”

When it first was developed, COBOL's easily-readable syntax made it profoundly different from anything that had been seen before. For the first time, it was possible to specify logic in a manner that was – at least to some extent – comprehensible even to non-programmers. Take for example, the following code written in FORTRAN – a language developed only a year before COBOL:

```
E = P * Q
I = I + E
```

With its original limitation on the length of variable names (one letter or a letter followed by a number), and its use of algebraic notation to express actions being taken, FORTRAN wasn't a particularly readable language, even by programmers. Compare this with the equivalent COBOL code:

```
MULTIPLY PRICE BY QUANTITY GIVING EXTENDED-AMOUNT
ADD EXTENDED-AMOUNT TO INVOICE-TOTAL
```

Clearly, even a non-programmer could at least conceptually understand what was going on! Over time, languages like FORTRAN evolved more robust variable names, and COBOL introduced a more formula-based syntactical capability for arithmetic operations, but FORTRAN was never as readable as COBOL.

Because of its inherent readability, I would MUCH rather be handed an assignment to make significant changes to a COBOL program about which I know nothing than to be asked to do the same with a C, C++, C# or Java program.

Those that argue that it is too boring/wasteful/time-consuming/insulting (pick one) to have to code a COBOL program “from scratch” are clearly ignorant of the following facts:

- ▶ Many systems have program-development tools available to ease the task of coding programs; those tools that concentrate on COBOL are capable of providing templates for much of the “overhead” verbiage of any program...
- ▶ Good programmers have – for decades – maintained their own skeleton “template” programs for a variety of program types; simply load a template into a text editor and you've got a good start to the program...
- ▶ Legend has it that there's actually only been ONE program ever written in COBOL – all programs ever “written” thereafter were simply derivatives of that one!

1.3.3.2. COBOL Program Structure

COBOL programs are structured into four major areas of coding, each with its own purpose. These four areas are known as DIVISIONS.

Each DIVISION may consist of a variety of SECTIONS and each SECTION consists of one or more PARAGRAPHS. A PARARAPH consists of SENTENCES, each of which consists of one or more STATEMENTS.

This hierarchical structure of program components standardizes the composition of all COBOL programs. Much of this manual describes the various divisions, sections, paragraphs and statements that may comprise any COBOL program.

See Also...

The Four Divisions of a Program	1.5	The DATA DIVISION	5
The IDENTIFICATION DIVISION	3	The PROCEDURE DIVISION	6
The ENVIRONMENT DIVISION	4		

1.3.3.3. Copybooks

A “copybook” is a segment of program code that may be utilized by multiple programs simply by having that program use the **COPY** statement to import that code into the program. This code may define files, data structures or procedural code.

Today’s current programming languages have a statement (usually, this statement is named “import”, “include” or “#include”) that performs this same function. What makes the COBOL copybook feature different than the “include” facility in current languages, however, is the fact that the COBOL **COPY** statement can edit the imported source code as it is being copied. This capability makes copybook libraries extremely valuable to making code reusable.

See Also...

The **COPY** Statement [2.1.1](#)

1.3.3.4. Structured Data

COBOL introduced the concept of structured data back in the 1960s. Structured data is data which may be accessed as a single item or may be broken down into sub-items based upon their character position of occurrence within the structure. These structures called *group items*. At the bottom of any structure are data items that aren’t broken down into sub-items. COBOL refers to these as *elementary items*.

1.3.3.5. Files

One of COBOLs main strengths is the wide variety of files it is capable of accessing. GNU COBOL programs, like those created with other COBOL implementations, need to have the structure of any files they will be reading and/or writing described to them. The highest-level characteristic of a file’s structure is defined by specifying the **ORGANIZATION** (section) of the file, as follows:

ORGANIZATION IS LINE SEQUENTIAL

These are files with the simplest of all internal structures. Their contents are structured simply as a series of data records, each terminated by a special end-of-record delimiter character. An ASCII line-feed character (hexadecimal 0A) is the end-of-record delimiter character used by any UNIX or pseudo-UNIX (MinGW, Cygwin, OSX) GNU COBOL build. A truly native Windows build would use a carriage-return, line-feed (hexadecimal 0D0A) sequence.

Records in this type of file need not be the same length.

Records must be read from or written to these files in a purely sequential manner. The only way to read (or write) record number 100 would be to have read (or written) records number 1 thru 99 first.

When the file is written by a GNU COBOL program, the delimiter sequence will be automatically added to each data record as it is written to the file. **WRITEs** to this type of file will be done using an implied “**BEFORE ADVANCING 1 LINE**” clause in the absence of an explicitly-specified **ADVANCING** clause.

When the file is read, the GNU COBOL runtime system will strip the trailing delimiter sequence from each record and pad the data (to the right) with **SPACES** if the data just read is shorter than the area described for data records in the program. If the data is too long, it will be truncated and the excess will be lost.

These files should not be defined to contain any exact binary data fields because the contents of those fields could inadvertently have the end-of-record sequence as part of their values – this would confuse the runtime system when reading the file, and it would interpret that value as an actual end-of-record sequence.

LINE ADVANCING files

These are files with an internal structure similar to that of the **LINE SEQUENTIAL** file. These files are defined (without an explicit **ORGANIZATION** specification) using the **LINE ADVANCING** clause on their **SELECT** statement.

When this kind of file is written by a GNU COBOL program, the delimiter sequence will be automatically added to each data record as it is written to the file. **WRITEs** to this type of file will be done using an implied “**AFTER ADVANCING 1 LINE**” clause in the absence of an explicitly-specified **ADVANCING** clause.

Like **ORGANIZATION LINE SEQUENTIAL** files, these files should not be defined to contain

any exact binary data fields because the contents of those fields could inadvertently have the end-of-record sequence as part of their values – this would confuse the runtime system when reading the file, and it would interpret that value as an actual end-of-record sequence.

ORGANIZATION IS RECORD BINARY SEQUENTIAL

These files also have a simple internal structure. Their contents are structured simply as an arbitrarily-long sequence of data characters. This sequence of data characters will be treated as a series of fixed-length data records simply by logically splitting the sequence of data characters up into a series of fixed-length segments each as long as the maximum record size defined in the program. There are no special end-of-record delimiter characters in the file and when the file is written to by a GNU COBOL program, no delimiter sequence is appended to the data.

Records in this type of file are all the same physical length, except possibly for the very last record in the file, which may be shorter than the others. If variable-length logical records are defined to the program, the space occupied by each physical record in the file will occupy the maximum possible space.

So, if a file contains 1275 characters of data, and a program defines the structure of that file as containing 100-character records, then the file contents will consist of twelve (12) 100-character records with a final record containing only 75 characters.

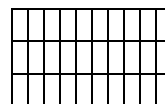
Even though it appears that it should be possible to locate and process any record in the file directly simply by calculating its starting character position based upon the program-defined record size, records must be still be read or written to these files in a purely sequential manner. The only way to read (or write) record number 100 would be to have read (or written) records number 1 thru 99 first.

When the file is read, the data is transferred into the program exactly as it exists in the file. In the event that a short record is read as the very last record, that record will be **SPACE** padded.

Care must be taken that programs reading such a file describe records whose length is exactly the same as that used by the programs that created the file. For example, the following shows the contents of a **RECORD BINARY SEQUENTIAL** file created by a program that wrote five 6-character records to it. The “A”, “B”, ... values and the background colors reflect the records that were written to the file:



Now, assume that another program reads this file, but described 10-character records rather than 6. Here are the records that program will read:



There may be times where this is exactly what you were looking for. More often than not, however, this is not desirable behavior. *Suggestion*: use a copybook to describe the record layouts of any file; this guarantees that multiple programs accessing that file will “see” the same record sizes and layouts.

These files can contain exact binary data fields because the contents of record fields are irrelevant to the reading process as there is no end-of-record delimiter.

ORGANIZATION IS RELATIVE

The contents of these files consist of a series of fixed-length data records prefixed with a four-byte record header. The record header contains the length of the data, in bytes. The byte-count does not include the four-byte record header.

Records in this type of file are all the same physical length. If variable-length logical records are defined to the program, the space occupied by each physical record in the file will occupy the maximum possible space.

This file organization was defined to accommodate either sequential or random processing. With a **RELATIVE** file, it is possible to read or write record 100 directly, without having to have first read or written records 1-99. The GNU COBOL runtime system uses the program-defined maximum record size to calculate a relative byte position in the file where the record header and data begin, and then transfers the necessary data to or from

the program.

When the file is written by a GNU COBOL program, no delimiter sequence is appended to the data, but a record-length field is added to the beginning of each physical record.

When the file is read, the data is transferred into the program exactly as it exists in the file.

Care must be taken that programs reading such a file describe records whose length is exactly the same as that used by the programs that created the file. It won't be a pretty site when the GNU COBOL runtime library ends up interpreting a four-byte ASCII character string as a record length when it transfers data from the file into the program!

Suggestion: use a copybook to describe the record layouts of any file; this guarantees that multiple programs accessing that file will “see” the same record sizes and layouts.

These files can contain exact binary data fields. The contents of record fields are irrelevant to the reading process as there is no end-of-record delimiter.

ORGANIZATION IS INDEXED

This is the most advanced file structure available to GNU COBOL programs. It's not possible to describe the physical structure of such files because that structure will vary depending upon which advanced file-management facility was included into the GNU COBOL build you will be using (Berkeley Database [BDB], VBISAM, etc.). We will – instead – discuss the logical structure of the file.

There will be multiple structures stored for an **INDEXED** file. The first will be a data component, which may be thought of as being similar to the internal structure of a **RELATIVE** file. Data records may not, however, be directly accessed by their record number as would be the case with a **RELATIVE** file, nor may they be processed sequentially by their physical sequence in the file.

The remaining structures will be one or more index components. An index component is a data structure that (somehow) enables the contents of a field, called a *primary key*, within each data record (a customer number, an employee number, a product code, a name, etc.) to be converted to a record number so that the data record for any given primary key value can be directly read, written and/or deleted. Additionally, the index data structure is defined in such a manner as to allow the file to be processed sequentially, record-by-record, in ascending sequence of the primary key field values. Whether this index structure exists as a binary-searchable tree structure (btree), an elaborate hash structure or something else is pretty much irrelevant to the programmer – the behavior of the structure will be as it was just described. The runtime system will not allow two records to be written to an indexed file with the same primary key value.

The capability exists for an additional field to be defined as what is known as an *alternate key*. Alternate key fields behave just like primary keys, allowing both direct and sequential access to record data based upon the alternate key field values, with one exception. That exception is the fact that alternate keys may be allowed to have duplicate values, depending upon how the alternate key field is described to the GNU COBOL compiler.

There may be any number of alternate keys, but each key field comes with a disk space penalty as well as an execution time penalty. As the number of alternate key fields increases, it will take longer and longer to write and/or modify records in the file.

These files can contain exact binary data fields. The contents of record fields are irrelevant to the reading process as there is no end-of-record delimiter.

All files are initially described to a GNU COBOL program using a **SELECT** statement coded in the **FILE-CONTROL** paragraph of the **INPUT-OUTPUT SECTION** of the **ENVIRONMENT DIVISION**. In addition to defining a name by which the file will be referenced within the program, the **SELECT** statement will specify the name and path by which the file will be known to the operating system along with its **ORGANIZATION**, locking and sharing attributes.

A file description in the **FILE SECTION** of the **DATA DIVISION** will define the structure of records within the file, including whether or not variable-length records are possible and – if so – what the minimum and maximum length might be. In addition, the file description entry can specify file I/O block sizes.

See Also...

Defining the Characteristics of a File 4.2.1	File Sharing 6.1.9.1
Describing the Structure of a File 5.1	Record Locking 6.1.9.2

(FD/SD)

1.3.3.6. Table Handling

Other programming languages have arrays, COBOL has tables. They're basically the same thing. What makes COBOL tables special are two special statements that exist in the COBOL language – **SEARCH** and **SEARCH ALL**.

The first can search a table sequentially, stopping only when either a table entry matching one of any number of search conditions is found, or when all table entries have been checked against the search criteria and none matched any of those criteria.

The second can perform an extremely fast search against a table sorted by and searched against a “key” field contained in each table entry. The algorithm used for such a search is a binary search (also known as a half-interval search). This algorithm ensures that only a small number of entries in the table need to be checked in order to find a desired entry or to determine that the desired entry doesn't exist in the table. The larger the table, the more effective this search becomes. For example, a table containing 32,768 entries will be able to locate a particular entry or will determine the entry doesn't exist by looking at no more than fifteen (15) entries! The algorithm is explained in detail in the **SEARCH ALL** documentation.

See Also...

Defining Tables 0

The **SEARCH** Statement [6.4.38.1](#)

The **SEARCH ALL** Statement [6.4.38.2](#)

1.3.3.7. Sorting and Merging Data

The COBOL language includes a powerful **SORT** statement that can sort large amounts of data according to arbitrarily complex key structures. This data may originate from within the program or may be contained in one or more external files. The sorted data may be written automatically to one or more output files or may be processed, record-by-record in the sorted sequence.

A special form of the **SORT** statement also exists just to sort the data that resides in a table. This is particularly useful if you wish to use **SEARCH ALL** against the table.

A companion statement – **MERGE**– can combine the contents of multiple files together, provided those files are all sorted in a similar manner according to the same key structure(s). The resulting output will consist of the contents of all of the input files, merged together and sequenced according to the common key structure(s). The output of a **MERGE** may be written automatically to one or more output files or may be processed internally by the program.

See Also...

The **MERGE** Statement [6.4.25](#)

The **SORT** Statement (File Sort) [6.4.40.1](#)

The **SORT** Statement (Table Sort) [6.4.40.2](#)

1.3.3.8. String Manipulation

There have been programming languages designed specifically for the processing of text strings, and there have been programming languages designed for the sole purpose of performing high-powered numerical computations. Most programming languages fall somewhere in the middle, between these two extremes. COBOL is no exception, although it does include some very powerful string manipulation capabilities; GNU COBOL actually has even more string-manipulation capabilities than many other COBOL implementations.

See Also...

Concatenate Two Or More Strings	CONCATENATE Intrinsic Function 6.1.7.9
	STRING Statement 6.4.43
Conversion Of A Numeric Time Or Date To A Formatted Character String	LOCALE-TIME Intrinsic Function 6.1.7.35
	LOCALE-DATE Intrinsic Function 6.1.7.32
Convert A Binary Value To Its Corresponding Character In The Program's Character Set	CHAR Intrinsic Function; add 1 to argument before invoking the function; The description of the CHAR function shows a technique that utilizes the MOVE statement that will accomplish the same thing without the need of adding 1 to 6.1.7.7

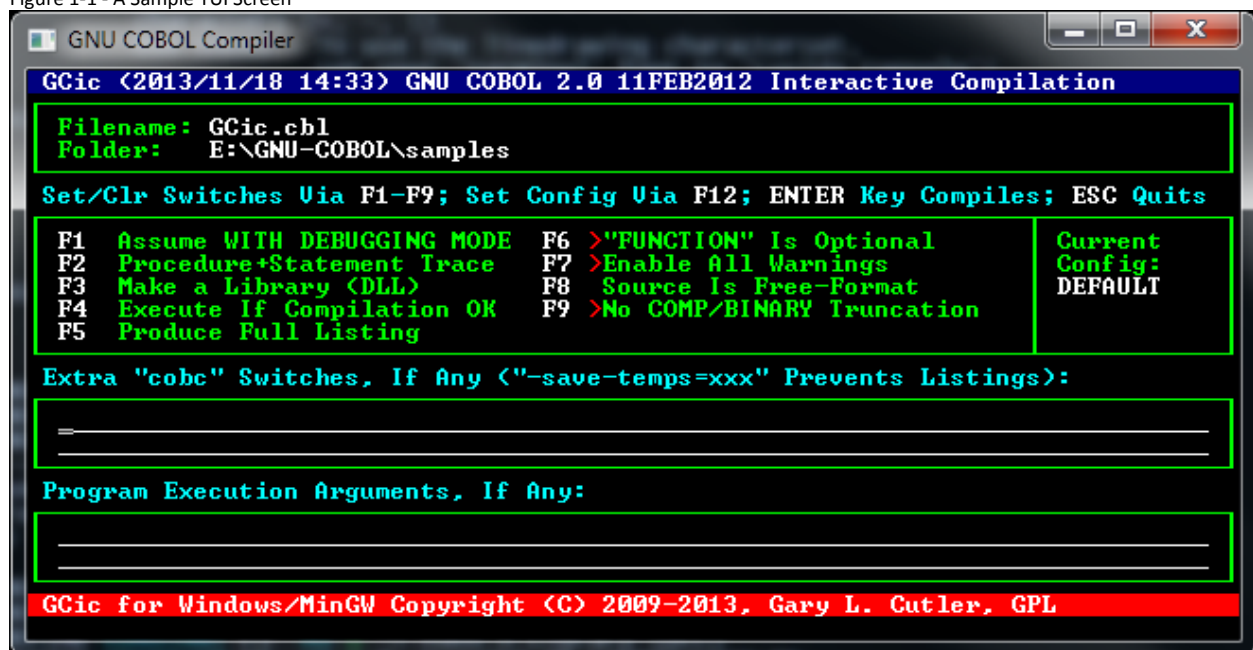
	the numeric argument value first	
Convert A Character String To Lower-Case	LOWER-CASE Intrinsic Function	6.1.7.39
	C\$TOLOWER Built-in Subroutine	8.3.1.13
	CBL_TOLOWER Built-in Subroutine	8.3.1.40
Convert A Character String To Upper-Case	UPPER-CASE Intrinsic Function	6.1.7.87
	C\$TOUPPER Built-in Subroutine	8.3.1.14
	CBL_TOUPPER Built-in Subroutine	8.3.1.41
Convert A Character String To Only Printable Characters, Changing Any Non-Printable Characters To A Default (“.”) Or Programmer-Specified Replacement Character.	C\$PRINTABLE Built-in Subroutine	8.3.1.11
Convert A Character To Its Numeric Value In The Program’s Characterset	ORD Intrinsic Function; subtract 1 from the result; The description of the ORD function shows a technique that utilizes the MOVE statement that will accomplish the same thing without the need of adding 1 to the numeric argument value first	6.1.7
Count Occurrences Of Substrings In A Larger String	INSPECT Statement with TALLYING Option	6.4.24
Decode A Formatted Numeric String Back To A Numeric Value (For Example, Decode “\$12,342.19-” To A -12342.19 Value)	NUMVAL Intrinsic Function	6.1.7.54
	NUMVAL-C Intrinsic Function (handles currency-formatted strings)	6.1.7.59
Determine The Length Of A String Or Data-Item Capable Of Storing Strings	LENGTH Intrinsic Function	6.1.7.31
	BYTE-LENGTH Intrinsic Function	6.1.7.6
Extract A Substring Of A String Based On Its Starting Character Position And Length	Use of a reference modifier on the string field.	6.1.3
Format A Numeric Item For Output, Including Thousands-Separators (“,” In The USA), Currency Symbols (“\$” In The USA), Decimal Points, Credit/Debit Symbols, Leading Or Trailing Sign Characters	MOVE Statement with picture-symbol editing applied to the receiving field	5.3 and 6.4.26
Justification (Left, Right Or Centered) Of A String Field	C\$JUSTIFY built-in subroutine	8.3.1.6
Monoalphabetic Substitution Of One Or More Characters In A String With Different Characters	INSPECT Statement with CONVERTING Option	6.4.24
	TRANSFORM Statement	6.4.47
	SUBSTITUTE Intrinsic Function	6.1.7.77
	SUBSTITUTE-CASE Intrinsic Function	6.1.7.78
Parse A String, Breaking It Up Into Substrings Based Upon One Or More Delimiting Character Sequences; These Delimiters May Be Single Characters, Multiple-Character Strings Or Multiple Consecutive Occurrences Of Either	UNSTRING Statement	6.4.49
Removal Of Leading Or Trailing Spaces From A String	TRIM Intrinsic Function	6.1.7.83
Substitution Of A Single Substring With Another <u>Of The Same Length</u> , Based Upon The Substrings Starting Character Position And Length	MOVE Statement with a reference modifier on the “receiving” field	6.1.3 and 6.4.26.1

Substitution Of One Or More Substrings In A String With Replacement Substrings <u>Of The Same Length</u> , Regardless Of Where They Occur	INSPECT Statement with REPLACING Option	6.4.24
	SUBSTITUTE Intrinsic Function	6.1.7.77
	SUBSTITUTE-CASE Intrinsic Function	6.1.7.78
Substitution Of One Or More Substrings In A String With Replacement Substrings <u>Of A Potentially Different Length</u> , Regardless Of Where They Occur	SUBSTITUTE Intrinsic Function	6.1.7.77
	SUBSTITUTE-CASE Intrinsic Function	6.1.7.78

1.3.3.9. Textual-User Interface (TUI) Features

The COBOL2002 standard formalizes extensions to the COBOL language that allow for the definition and processing of text-based screens, as is a typical function on mainframe computers. GNU COBOL implements virtually all the screen-handling features described by COBOL2002. Here is an example of such a screen as it might appear in the console window of a Windows computer:

Figure 1-1 - A Sample TUI Screen



Screens such as this⁷ are defined in the **SCREEN SECTION** of the **DATA DIVISION**. Once defined, screens are used at run-time via the **ACCEPT** and **DISPLAY** statements.

The COBOL2002 standard only covers textual-user interface (TUI) screens and not the more-advanced graphical-user interface (GUI) screen design and processing capabilities built into most modern operating systems. There are subroutine-based packages available that can do full GUI development, but none are open-source.

See Also...

Defining Screens [5.6](#)

The **ACCEPT** Statement (Screen Data) [6.4.1.4](#)

The **DISPLAY** Statement (Screen Data) [6.4.12.4](#)

1.4. Syntax Description Conventions

Syntax of the GNU COBOL language will be described in this manual with conventions familiar to COBOL programmers, with a few coloring conventions thrown in to aid in readability and interpretation. The following is a description of those syntactical-description techniques:

Black Syntactical elements that are part of the GNU COBOL language (including required punctuation symbols, operators and so on) will appear in black. Other colors such as red and blue will be used to highlight those elements that are merely part of the syntax description.

⁷ This screen comes from the program named GCic – a full-screen front-end to the GNU COBOL compiler – the source code of which is included as a sample in this manual. See section [10.4](#) for the listing of the program.

UPPERCASE	COBOL language keywords and implementation-dependent names (the so-called “reserved words” of the COBOL language) will appear in BOLD UPPERCASE .
<u>UNDERLINING</u>	reserved words that are <u>UNDERLINED</u> are required in whatever syntactical context they are shown. If a reserved word is not underlined, it is optional and its presence or absence has no effect on the program.
<i>lowercase-italic</i>	Generic terms representing substitutable items will be shown in <i>italic lowercase</i> .
[<i>optional-syntax</i>]	Red Square brackets are used to enclose optional syntax. Any clauses not enclosed in square brackets are mandatory. These are also used sometimes in conjunction with the ellipsis (...) to indicate an optional syntactical item that could be repeated.
<i>choice-1</i> <i>choice-2</i>	Simple choices may be indicated with a red vertical bar separating them. Although not typically used in COBOL syntactical diagrams, this convention is an effective alternative that may be used when square brackets would make a syntax diagram too complicated. For example, THRU THROUGH would indicate that either of the required reserved words THRU or THROUGH may be used.
{ <i>choice-1</i> } <i>choice-2</i> }	Red braces are used to enclose choices. <u>Exactly one</u> of the choices contained within the braces must be selected. These are also used sometimes in conjunction with the ellipsis (...) to indicate a choice of syntactical items that may be repeated.
...	A red three-dot sequence (called an “ellipsis”) may appear following [], { } or <i>lowercase italic entries</i> to indicate that the syntax element preceding the ellipsis may occur multiple times.
Shaded Areas	Shaded areas are used to highlight syntax elements that are <u>recognized</u> by the GNU COBOL compiler but will either have no effect on the generated code or will have a compiler warning issued announcing that feature is unsupported. Such elements are either present in the GNU COBOL language to facilitate the porting of programs from other COBOL environments, reflect syntax elements that are not yet fully implemented or syntax elements that have become obsolete.

1.5. General GNU COBOL Program Format

1.5.1. Source Line Format

1.5.1.1. Fixed Format Mode

Prior to the COBOL2002 standard, source statements in COBOL programs were oriented around 80-column punched cards. This means that each source line in a COBOL program consisted of five different “areas”, defined by their column number.

This structure is enforced by GNU COBOL when the compiler is operating in **Fixed Format Mode**; Fixed Format Mode is the default mode in effect when the compiler begins execution.

Column Numbers	Area Name	Usage
1-6	Sequence Number Area	<p>Historically back in the days when punched-cards were used to submit COBOL program source to a COBOL compiler, this part of a COBOL statement was reserved for a six-digit sequence number.</p> <p>While the contents of this area are ignored by COBOL compilers, it existed so that a program actually punched on 80-character cards could – if the card deck were dropped on the floor – be run through a card sorter machine and restored to it’s proper sequence. Of course, this isn’t necessary today; if truth be told, it hasn’t been necessary for a long time.</p> <p>See Section 9.1 for a discussion of how this area tends to be used today.</p>

Column Numbers	Area Name	Usage
7	Indicator Area	Column 7 serves as an indicator in which one of five possible values will appear – space, “D” (or “d”), “-” (dash), “/” or “*”. The vast majority of COBOL source file lines have a space in this position. The values “D”, “*” and “/” are three different types of “comment” indicators, telling the compiler to (normally) ignore this source line. A value of “-” served as a continuation character in the event that a literal value, reserved word or programmer-defined name needed to be split across two lines of code. This is/was rarely used and – when it does – is/was almost always used to continue an alphanumeric literal (character string).
8-11	“Area A”	Language DIVISION, SECTION and paragraph section headers must begin in Area A, as must the level numbers 01, 77 in data description entries and the “FD” and “SD” file and SORT description headers.
12-72	“Area B”	All other COBOL programming language components are coded in these columns.
73-80	Program Name Area	This is another area of COBOL statements that is ignored by COBOL compilers. This part of every statement also hails back to the day when programs were punched on cards – it was expected that the name of the program (or at least the first 8 characters of it) would be punched here so that – if a dropped COBOL source deck contained more than one program, that handy card sorter machine could be used to first separate the cards by program name and then sort them by sequence number. Today’s COBOL compilers (including GNU COBOL) simply ignore anything past column 73

The GNU COBOL compiler (cobc) operates in fixed format mode by default (you may explicitly specify the “-fixed” switch, if you wish, but that is the default mode), unless you specify otherwise in one of the following ways:

- ▶ You run the compiler with the “-free” switch to turn on free-format mode.
- ▶ You use the “>>SET SOURCEFORMAT AS FREE” CDF directive to turn on free-format mode
- ▶ You use the “>>SOURCE FORMAT IS FREE” CDF directive to turn on free format mode

See Also...

Coding Comments in Programs 1.6	The Compiler Directing Facility (CDF) 2.2
Alphanumeric Literals 1.8.2	

1.5.1.2. Free Format Mode

As of the COBOL2002 standard, a second mode now exists for COBOL source code statements – Free Format Mode.

In this mode of operation, GNU COBOL statements may each be up to 255 characters long, with no specific requirements as to what should appear in which columns.

The GNU COBOL compiler (cobc) can be commanded to operate in free format mode in any of the following ways:

- ▶ You run the compiler with the “-free” switch
- ▶ You use the >>SET SOURCEFORMAT AS FREE CDF directive to turn on free-format mode
- ▶ You use the >>SOURCE FORMAT IS FREE CDF directive to turn on free format mode

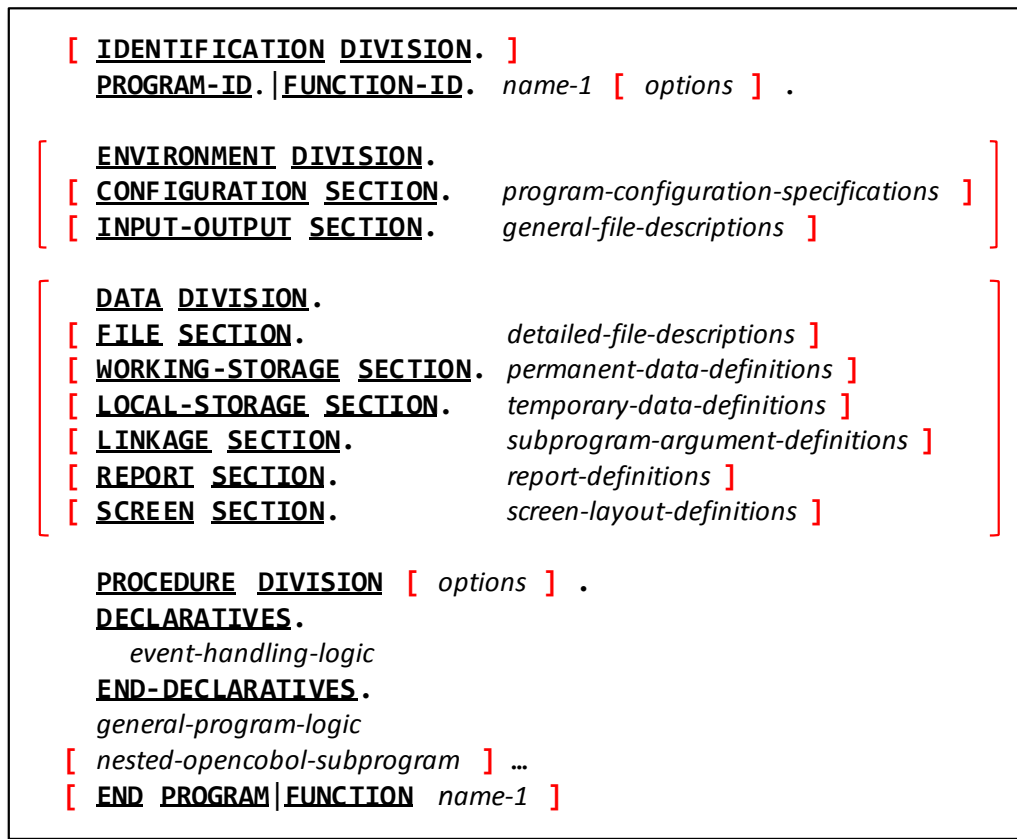
Using >>SET and >>SOURCE directives in your source code, you may switch back and forth between fixed and free format mode at will.

See Also...

Coding Comments in Programs 1.6	The Compiler Directing Facility (CDF) 2.2
Alphanumeric Literals 1.8.2	

1.5.2. Program Structure

Figure 1-2 – General Format of a GNU COBOL Program



What you see here is the general format of a GNU COBOL program. Each program consists of up to four DIVISIONS (major groupings of language statements that all relate to a common purpose). Not all divisions are needed in every program, but they must be specified in the order shown when they are used.

This general program structure looks quite intimidating, but bear in mind that each DIVISION and SECTION you see here serves a very specific function, and it is rare to find a program that needs each and every one of those functions!

1. A single file of COBOL source code may contain:
 - a. A portion of a program; these files are known as copybooks
 - b. A single program. In this case, the **END PROGRAM / END FUNCTION** statement is optional.
 - c. Multiple programs, separated from one another by **END PROGRAM / END FUNCTION** statements. The final program in such a source code file need not have an **END PROGRAM / END FUNCTION** statement.
2. Program “B” may be *nested* inside program “A” by including program B’s source code at the end of program A’s **PROCEDURE DIVISION** without an intervening **END PROGRAM A / END FUNCTION A** statement. For now, that’s all that will be said about nesting. Regardless of how many programs comprise a single GNU COBOL source file (see #1c), only a single output executable program will be generated from that source file when the file is compiled.
3. Here is a brief summary of the purpose of each DIVISION in a program:

DIVISION	Purpose
IDENTIFICATION	The IDENTIFICATION DIVISION provides basic identification of the program (or function) by giving it a name. While the IDENTIFICATION DIVISION is required in all programs, the actual “ IDENTIFICATION DIVISION ” header – as of the COBOL2002 standard – is not.
ENVIRONMENT	The ENVIRONMENT DIVISION defines the external computer environment in which the program will be operating. This includes defining any files that the program may be accessing.
DATA	The DATA DIVISION is used to define all data that will be processed by a program.
PROCEDURE	The PROCEDURE DIVISION contains all executable program code.

See Also...

Copybooks 1.3.3.3	The IDENTIFICATION DIVISION 3
Subprograms Subroutines vs Functions 7.1	The ENVIRONMENT DIVISION 4
Details Of Nested Subprograms 7.6	The DATA DIVISION 5
	The PROCEDURE DIVISION 6

1.6. In-Program Documentation (i.e. “Comments”)

The following chart documents how comments may be imbedded into GNU COBOL program source to provide documentation.

Type of Comment	When in “FIXED” Mode...	When in “FREE” Mode...
Blank lines	Blank lines may be inserted as desired.	Blank lines may be inserted as desired.
Full-line comments	An entire source line will be treated as a comment (and will be ignored by the compiler) by coding an asterisk (“*”) in column seven (7).	An entire source line will be treated as a comment (and will be ignored by the compiler) by coding the sequence “*>”, starting in any column, as the first non-blank characters on the line.
Full-line comments with form-feed	An entire source line will be treated as a comment by coding a slash (“/”) in column seven (7). In addition, most COBOL compilers capable of generating source program listings will issue a form-feed in the listing so that the “/” line is at the top of a new page of the listing. The GNU COBOL compiler (cobc) does <u>not</u> support this form-feed behavior, although it does treat “/” lines as comments. The GNU COBOL Interactive Compiler, or GCic, <u>does</u> support this form-feed behavior when it generates program source listings! GCic is a GNU COBOL program that provides a full-screen front-end to the actual GNU COBOL compiler. You can see a screenshot of it in section 1.3.3.9 .	There is no FREE-mode equivalent to “/”.
Partial-line comments	Any text following the character sequence “*>” on a source line will be treated as a comment. The “*” must appear in column seven (7) or beyond.	Any text following the character sequence “*>” on a source line will be treated as a comment. The “*” may appear in any column.
Comments that may be treated as code (typically for debugging purposes)	By coding a “D” in column 7 (upper- or lower-case), an otherwise valid GNU COBOL source line will be treated as a comment by the compiler.	By specifying the character sequence “>>D” (upper- or lower-case) as the first non-blank characters on a source line, an otherwise valid GNU COBOL source line will be treated as a comment by the compiler.
	Such statements may be compiled either by specifying the “-fdebugging-line” switch on the GNU COBOL compiler or by adding the “WITH DEBUGGING MODE” clause to the SOURCE-COMPUTER paragraph.	

See Also...

The SOURCE-COMPUTER Paragraph 4.1.1	Sample Program Listing: GCic 9.4
------------------------------------------------------------	---------------------------------------------------------

1.7. Use of Commas and Semicolons

A comma (“,”) or a semicolon (“;”) may be inserted into a GNU COBOL program to improve readability at any spot where white space would be legal (except, of course, within alphanumeric literals). These characters are always optional.

The use of comma characters can cause “confusion” to a COBOL compiler if the **DECIMAL POINT IS COMMA** clause is used in **SPECIAL-NAMES**. The following statement, which calls a subroutine passing it two arguments (the numeric constants 1 and 2):

```
CALL “SUBROUTINE” USING 1,2
```

would – with **DECIMAL POINT IS COMMA** in effect – actually be interpreted as a subroutine call with ONE argument (the non-integer numeric constant 1.2).

See Also...

The **SPECIAL-NAMES** Paragraph [4.1.4](#)

1.8. Use of Literals

Literals are constant values that will not change during the execution of a program. There are two fundamental types of literals – numeric and alphanumeric.

1.8.1. Numeric Literals

Numeric literals are numeric constants which may be used as array subscripts, as values in arithmetic expressions, or in any procedural statement where a numeric value may be used. Numeric literals may take any of the following forms:

- ▶ Integers such as 1, 56, 2192 or -54.
- ▶ Non-integer fixed point values such as 1.12 or -2.95.
- ▶ Floating-point values using “*Enn*” notation such as 9.92E25 (representing 9.92×10^{25}) or 5.7E-14 (representing 5.7×10^{-14}). Both the mantissa (the number before the **E**) and the exponent (the number after the **E**) may be explicitly specified as positive (with a +), negative or unsigned (and therefore implicitly positive). A floating-point literal’s value must be within the range -1.7×10^{308} to $+1.7 \times 10^{308}$ with no more than 15 decimal digits of precision.
- ▶ Hexadecimal numeric literals such as H’1F’ ($1F_{16} = 31_{10}$), h’22’ ($22_{16} = 34_{10}$) or H’DEAD’ ($DEAD_{16} = 57005_{10}$). The **H** character may either be upper- or lower-case and either single quote (‘) or double-quote (“) characters may be used. Hexadecimal numeric literals are limited to a maximum value of H’FFFFFFFFFFFFFF’ (a 64-bit value).

1.8.2. Alphanumeric Literals

Alphanumeric literals are character strings suitable for display on a computer screen, printing on a report, transmission through a communications connection or storage in **PICTURE X** or **PICTURE A** data items. These are NOT valid for use in arithmetic expressions unless they can first be converted to their numeric computational equivalent via the **NUMVAL** and **NUMVAL-C** intrinsic functions.

Alphanumeric literals may take any of the following forms:

- ▶ Any sequence of characters enclosed by a pair of single-quote (‘) characters or a pair of double-quote (“) characters constitutes a *string literal*. The double-quote character (“) may be used as a data character within an apostrophe-delimited string literal, and an apostrophe may be used as a data character within a double-quote-delimited string literal. If an apostrophe character must be included as a data character within an apostrophe-delimited string literal, express that character as two consecutive apostrophes (’’). If a double-quote character must be included as a data character within a double-quote-delimited string literal, express that character as two consecutive double-quotes (’’’’).
- ▶ A literal formed according to the same rules as for a string literal (above), but prefixed with the letter “**Z**” (upper- or lower-case) constitutes a *zero-delimited string literal*. These literals differ from ordinary string literals in that they will be explicitly terminated with a byte of hexadecimal value 00. This facilitates the “sharing” of such literals with C programs⁸.
- ▶ A hexadecimal literal such as X’4A4B4C’ ($4A4B4C_{16}$ = the ASCII string ‘JKL’), x’20’ (20_{16} = a space) or X’30313233’ (30313233_{16} = the ASCII string ‘0123’). The “**X**” character may either be upper- or lower-case and either single quote (‘) or double-quote (“) characters may be used. These hexadecimal alphanumeric literals should always consist of an even number of hexadecimal digits, because each character is represented by eight bits worth of data (2 hex digits). Hexadecimal alphanumeric literals may be of almost unlimited length.

Alphanumeric literals too long to fit on a single line may be continued to the next line in one of two ways:

⁸ In the C programming language, strings must be terminated with a null byte (X’00’).

1. If you are using Fixed Format Mode, the alphanumeric literal can be run right up to and including column 72. The literal may then be continued on the next line anywhere after column 11 by coding another quote or apostrophe (whichever was used to begin the literal originally). The continuation line must also have a hyphen (-) coded in the indicator area (column 7). Here is an example:

```

1         2         3         4         5         6         7         8
1234567890123456789012345678901234567890123456789012345678901234567890
01 LONG-LITERAL-VALUE-DEMO PIC X(60) VALUE "This is a long l
-                                           "iteral that must
-                                           " be continued."
```

2. Regardless of whether the compiler is operating in Fixed or Free Format Mode, GNU COBOL allows alphanumeric literals to be broken up into separate fragments. These fragments have their own beginning and ending quote/apostrophe characters and are "glued together" at compilation time using "&" characters. No continuation indicator is needed. Here's an example:

```

1         2         3         4         5         6         7         8
1234567890123456789012345678901234567890123456789012345678901234567890
01 LONG-LITERAL-VALUE-DEMO PIC X(60) VALUE "This is a" &
                                           " long literal that must " &
                                           "be continued."
```

If your program is using Free Format Mode, there's less need to continue long alphanumeric literals because statements may be as long as 255 characters.

Numeric literals may be split across lines just as alphanumeric literals are, using either of the above techniques and both reserved and user-defined words can be split across lines too (using the first technique). The continuation of numeric literals and user-defined/reserved words is provided merely to provide compatibility with older COBOL versions and programs, but should not be used with new programs – it just makes for ugly-looking programs.

See Also...

Fixed-Format Source Code 1.5.1.1	The NUMVAL Intrinsic Function 6.1.14.58
Defining a Data Item's PICTURE 5.2.1.6	The NUMVAL-C Intrinsic Function 6.1.14.59

1.9. Use of Figurative Constants

Figurative constants are reserved words that may be used in lieu of certain literals. In general, a figurative constant may be freely used anywhere its corresponding value could have been used; when used, their value is interpreted were an arbitrarily long sequence of the characters in question.

The following chart lists the GNU COBOL figurative constants and their respective equivalent values.

Figure 1-3 - Figurative Constants

Figurative Constant	Type of Literal	Equivalent Value
ZERO, ZEROS, ZEROES	Numeric	0
SPACE, SPACES	Alphanumeric	Blank
QUOTE, QUOTES	Alphanumeric	Double-quote character(s)
LOW-VALUE, LOW-VALUES	Alphanumeric	The character whose value in the programs collating sequence is lowest. If a program is using the ASCII collating sequence, this will represent a sequence of characters comprised entirely of 0-bits.
HIGH-VALUE, HIGH-VALUES	Alphanumeric	The character whose value in the programs collating sequence is highest. If a program is using the ASCII collating sequence, this will represent a sequence of characters comprised entirely of 1-bits.
NULL	Alphanumeric	A character comprised entirely of zero-bits (regardless of the programs collating sequence).

1.10. User-Defined Names

When you write GNU COBOL programs, you'll need to create a variety of names to represent various aspects of the program, the programs data and the external environment in which the program is running.

User-defined names may be composed from the characters “A” through “Z” (upper- and/or lower-case), “0” through “9”, dash (“-”) and underscore (“_”). User-defined names may neither start nor end with hyphen or underscore characters.

With the exception of [procedure names](#), user-defined names must contain at least one letter.

When user-defined names are created as names for data, they will be referenced in this document under the term [identifier](#).

1.11. Use of LENGTH OF

Alphanumeric literals and identifiers may optionally be prefixed with the clause “**LENGTH OF**”. In such cases, the literal actually is a numeric literal with a value equal to the number of bytes in the alphanumeric literal. For example, the following two GNU COBOL statements both display the same result (27):

LENGTH OF { <i>numeric-literal-1</i> } <i>identifier-1</i>

```

01 Demo-Identifier          PIC X(27).  *> This is a 27-character data-item
.
.
.
DISPLAY LENGTH OF "This is a LENGTH OF Example"
DISPLAY LENGTH OF Demo-Identifier
DISPLAY 27

```

The **LENGTH OF** clause on a *literal* or *identifier* reference may generally be used anywhere a numeric literal might be specified, with the following exceptions:

1. In place of a literal on a **DISPLAY** statement.
2. As part of a **WRITE** or **RELEASE** statement’s **FROM** clause.
3. As part of the **TIMES** clause of a **PERFORM**.

2. The GNU COBOL Compiler Directing Facility [CDF]

The Compiler Directing Facility is a means of controlling the compilation of GNU COBOL programs, providing a mechanism for dynamically setting or resetting certain compiler switches, introducing new source code from one or more source code libraries, making dynamic source code modifications or conditionally processing / ignoring source statements.

When the compiler is operating in FIXED mode, all CDF statements must begin in column eight (8) or beyond.

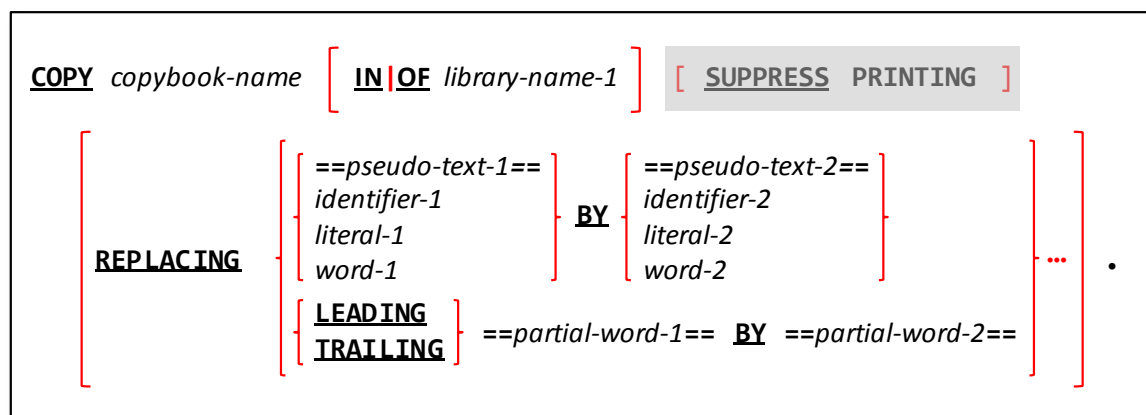
There are two types of supported CDF statements in GNU COBOL – Text Manipulation Statements and Compiler Directives.

2.1. Text Manipulation Statements

CDF text manipulation statements are used to introduce new code into programs either with or without changes, or may be used to modify existing statements already in the program.

2.1.1. The COPY Statement

Figure 2-1 - COPY Syntax



COPY statements are used to import copybooks into a program.

GNU COBOL completely supports the use of copybooks. These are separate source files containing **ANY GNU COBOL SYNTAX WHATSOEVER**, including other CDF statements.

1. **COPY** statements may be used anywhere within a COBOL program where the code contained within the copybook would be syntactically valid.
2. The syntax diagram above places great emphasis on a period at the end of the **COPY** statement and any **REPLACING** clauses it may have. A period is absolutely mandatory at the end of every **COPY** statement, even if the **COPY** statement occurs within the scope of a command where a period might appear disruptive (such as within the scope of an **IF...END-IF** sequence; the period on the **COPY** command will not, however, affect the command scope in which the **COPY** occurs.
3. All **COPY** statements are resolved and the contents of the corresponding copybooks inserted into the program source code before the actual compilation process begins.
4. The optional “**REPLACING**” clause allows any reserved words (*word-1*, *word-2*), data items (*identifier-1*, *identifier-2*), literals (*literal-1*, *literal-2*) or whitespace-delimited phrases to be replaced. Any number of such substitutions may be made as a copybook is included into a program.

See Also...

Copybooks [1.3.3.3](#)

How the Compiler Finds Copybooks [8.1.5](#)

2.1.2. The REPLACE Statement

Format 1:

Figure 2-2 - REPLACE (Format 1) Syntax

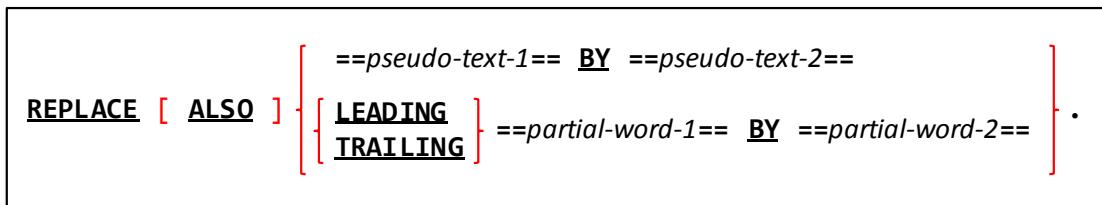
**Format 2:**

Figure 2-3 - REPLACE (Format 2) Syntax



The **REPLACE** statement provides a mechanism for changing all or part of one or more GNU COBOL statements.

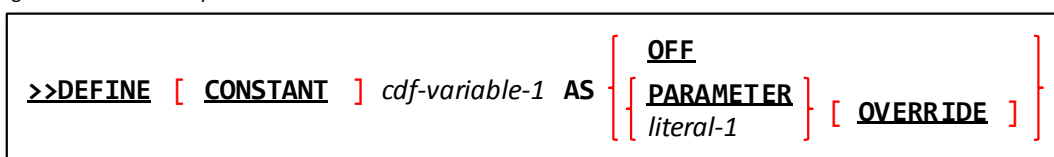
1. The syntax diagrams above place great emphasis on a period at the end of the **REPLACE**. A period is absolutely mandatory at the end of every **REPLACE** statement, even if the **REPLACE** statement occurs within the scope of a command where a period might appear disruptive (such as within the scope of an **IF...END-IF** sequence; the period on the **REPLACE** command will not, however, affect the command scope in which the **REPLACE** occurs.
2. The **REPLACE** statement can be used to make changes to program source code in much the same way as the **REPLACING** option of the **COPY** statement can.
3. Once a Format 1 **REPLACE** statement is encountered in the compilation unit, it will remain in-effect – continuing to make those source code changes it specifies – until one of the following occurs:
 - a. Another Format 1 **REPLACE** is encountered; in such a case, the change rules defined by the former Format 1 **REPLACE** will be replaced by those defined by the new **REPLACE**, unless the newly-encountered **REPLACE** statement includes the “**ALSO**” keyword; in this instance, the **REPLACE** currently in-effect will be “remembered” and then replaced by one combining the effects of the currently in-effect **REPLACE** and the new one.
 - b. A Format 2 **REPLACE** is encountered. If the Format 2 **REPLACE** includes the “**LAST**” keyword, the currently in-effect **REPLACE** will be terminated and the most-recently “remembered” **REPLACE** will be re-activated. If the Format 2 **REPLACE** does not include the “**LAST**” keyword, the currently in-effect **REPLACE** will be terminated and all “remembered” prior **REPLACES** will be discarded; no further changes will be made until such a point as another Format 1 **REPLACE** (if any) is encountered.
 - c. The last line of source code in the compilation unit has been processed.

2.2. CDF Directives

Compiler Directing Facility directives, or statements, are denoted by the presence of a “>>” character sequence as part of the statement name itself – are used to influence the process of program compilation.

2.2.1. The >>DEFINE Directive

Figure 2-4 - >>DEFINE Syntax



Use **>>DEFINE** to create CDF variables and (optionally) assign them either literal or environment variable values.

1. CDF variables defined in this way become undefined once an **END PROGRAM** or **END FUNCTION** directive is encountered in the input source.
2. The **>>DEFINE** statement is one way to create CDF variables that may be processed by other CDF statements such as **>>IF**. The **>>SET** statement provides another way to create them.

3. CDF variable names follow the rules for standard GNU COBOL user-defined names, and may not duplicate any CDF reserved word. CDF variable names may duplicate COBOL reserved words, provided the **CONSTANT** option is not specified, but such names are not recommended.
4. The **CONSTANT** option, valid only in conjunction with *literal-1*, defines a CDF variable that may be used within your regular COBOL code as if it were a literal value. Without the **CONSTANT** option, the CDF variable may only be referenced on other CDF statements.
5. The **OFF** option is used to create a variable without assigning it any value.
6. The **PARAMETER** option is used to create a variable whose value is that of the environment variable of the same name. Note that this value assignment occurs at compilation time, not program execution time.
7. The "*literal-1*" option is used to specify a numeric or alphanumeric literal, as previously discussed.
8. In the absence of the **OVERRIDE** option, *cdf-variable-1* must not yet have been **DEFINED**.
9. When the **OVERRIDE** option is specified, *cdf-variable-1* will be created with the specified value, if it had not yet been **DEFINED**, or it will be re-**DEFINED** with the new value if it had already been **DEFINED**.

10. See Also...

Literals	1.8	The >>SET CDF Statement	2.2.3
User-defined Names	1.10		

2.2.2. The >>IF Directive

Figure 2-5 - >>IF Syntax

```

>>IF constant-conditional-expression-1
  [ program-source-lines-1 ]

[ >>ELIF constant-conditional-expression-1 ] ...
  [ program-source-lines-2 ]

[ >>ELSE
  [ program-source-lines-n ] ]

>>END-IF

```

Conditionally process or ignore COBOL source statements and/or CDF text-manipulation statements depending upon the value of one or more conditional expressions based upon CDF variables.

1. Each >>IF statement must be terminated by an >>END-IF statement.
2. There may be any number of >>ELIF clauses following an >>IF, including zero.

3. The syntax of a *constant-conditional expression* is as follows:

Figure 2-6 - >>IF constant-conditional-expression Format

```

{ cdf-variable-1
  literal-1 } IS [ NOT ] { DEFINED
  SET
  cdf-relational-operator { cdf-variable-2
  literal-2 } }

```

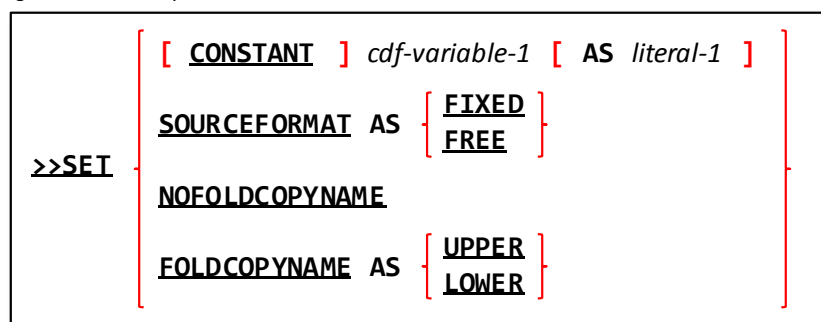
4. The *text-1*, *text-2* and *text-n* entries represent lines of source code that may consist of any number of GNU COBOL statements and/or CDF text-manipulation statements (including none at all). Currently, *text-1*, *text-2* and *text-n* should not contain any CDF compiler directives ("*>>*" statements).
5. Each *constant-conditional-expression* will be evaluated in the sequence in which they are coded in the >>IF statement and any >>ELIF clauses that may be present until one evaluates to **TRUE**. Once one of them evaluates to **TRUE**, the corresponding *text* block of statements will be processed by the compiler and all others within the scope of the >>IF statement will be skipped. If none of them evaluate to **TRUE**, the *text-n* block of statements (following the >>ELSE clause) will be processed by the compiler and all others within the scope of the >>IF statement will be skipped. If none of the *constant-conditional-expressions* evaluate to **TRUE** and there is no >>ELSE clause, then none of the *text* blocks of statements within the scope of the >>IF will be processed by the compiler.
6. The following rules pertain to *constant-conditional-Expressions*

- The **DEFINED** option tests for whether *variable-1* has been defined, but not yet assigned a value (**>>DEFINE ... OFF**); use the **NOT** option to test for the variable not being defined.
- The **SET** option tests for whether *variable-1* has been given a value, either via a **>>SET** statement or via a **>>DEFINE** without the **OFF** option.
- Two CDF variables, two literals or a single CDF variable and a single literal may be compared against each other using a relational operator. Unlike the standard GNU COBOL **IF** statement, multiple comparisons cannot be **“AND”**ed or **“OR”**ed together; you may nest a second **>>IF** inside the first, however, to simulate an **“AND”** and an **“OR”** may be simulated via the **>>ELIF** option. Valid relational operators are as follows (you may use either words or symbols):

GREATER THAN OR EQUAL TO	>=
GREATER THAN	>
LESS THAN OR EQUAL TO	<=
LESS THAN	<
EQUAL TO	=
	<> (meaning “not equal”)

2.2.3. The >>SET Directive

Figure 2-7 - >>SET Syntax



The **>>SET** statement provides an alternate means of performing the actions of the **>>DEFINE** and **>>SOURCE** statements, as well as a means of controlling the **“-free”**, **“-fixed”** and **“-ffold-copy”** compiler switches from within program source code itself.

- CDF variables defined in this way become undefined once an **END PROGRAM** or **END FUNCTION** directive is encountered in the input source.
- The **FOLDCOPYNAME** option provides the equivalent of specifying the compiler **“-ffold-copy=xxx”** switch, where **“xxx”** is either **“UPPER”** or **“LOWER”**.
- The **NOFOLDCOPYNAME** option turns off the effect of either the **>>SET FOLDCOPYNAME** statement or the **“-ffold-copy”** switch.
- If the **“CONSTANT”** option is used, the **“AS”** option must also be used.
- The remaining options of the **>>SET** statement provide equivalent functionality to the **>>DEFINE** and **>>SOURCE** statements, as shown in the following table:

>>SET Statement	Equivalent >>DEFINE or >>SOURCE Statement
>>SET cdf-variable	>>DEFINE cdf-variable AS OFF
>>SET cdf-variable AS literal-1	>>DEFINE cdf-variable AS literal-1
>>SET CONSTANT cdf-variable-1 AS literal-1	>>DEFINE CONSTANT cdf-variable-1 AS literal-1
>>SET SOURCEFORMAT AS FIXED	>>SOURCE FORMAT IS FIXED ; sets the “-fixed” compiler switch
>>SET SOURCEFORMAT AS FREE	>>SOURCE FORMAT IS FREE ; sets the “-free” compiler switch

See Also...

Compiler Switches Reference [8.1.2](#)

2.2.4. The >>SOURCE Directive

Figure 2-8 - >>SOURCE Syntax

```
>>SOURCE FORMAT IS { FIXED
                     FREE }
```

The >>SOURCE statement puts the compiler into FIXED or FREE source-code format mode. This, in effect, provides yet another mechanism for controlling the “-free” and “-fixed” compiler switches.

1. You may switch between **FIXED** and **FREE** mode as desired.
2. You may also use the >>SET statement to perform this function.
3. If the compiler is already in the specified mode, this statement will have no effect.

See Also...

The >>SET CDF Statement [2.2.3](#)

Compiler Switches Reference [8.1.2](#)

2.2.5. The >>TURN Directive

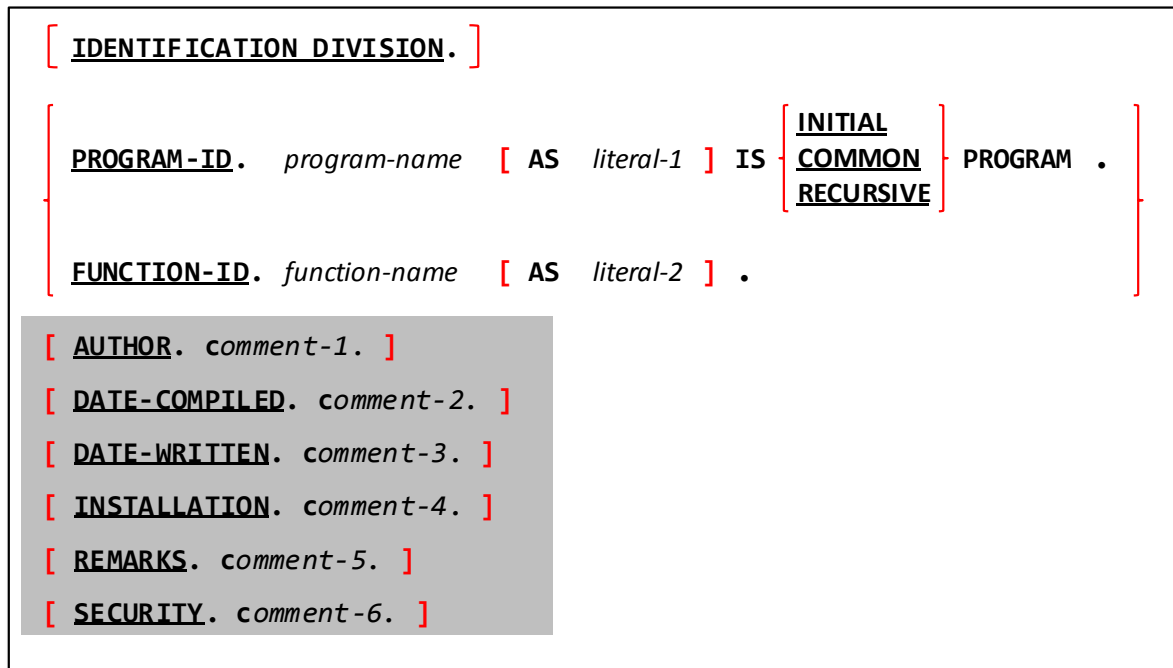
Figure 2-9 - >>TURN Syntax

```
>>TURN { exception-name-1 [ file-name-1 ] ... } ... CHECKING { ON [ WITH LOCATION ]
                                                                    OFF }
```

The >>TURN statement, while accepted syntactically, is currently non-functional.

3. IDENTIFICATION DIVISION

Figure 3-1 - IDENTIFICATION DIVISION Syntax



The **IDENTIFICATION DIVISION** provides basic identification of the program by giving it a name, and optionally defining some high-level characteristics.

1. While the actual **IDENTIFICATION DIVISION** header is optional, the **PROGRAM-ID / FUNCTION-ID** clause is not.
2. The **AUTHOR**, **DATE-COMPILED**, **DATE-WRITTEN**, **FUNCTION-ID**, **INSTALLATION**, **PROGRAM-ID**, **REMARKS** and **SECURITY** clauses may be specified in any sequence. These clauses are supported by GNU COBOL only to provide compatibility with programs written for the ANS1974 (or earlier) standards. As of the ANS1985 standard, these clauses have been obsolete and should not be used in new programs.

The “-**Wobsolete**” compilation switch will cause the GNU COBOL compiler to issue warnings messages if these (or any other obsolete syntax) is used in a program.

3. Both *literal-1* and *literal-2* must be actual alphanumeric literals and may not be figurative constants.
4. The **PROGRAM-ID** and **FUNCTION-ID** clause serve to identify the program to the external (i.e. operating system) environment. If there is no **AS** clause present, the program-name or function-name will serve as that external identification. If there is an **AS** clause specified, that specified literal will serve as the external identification. For the remainder of this document, that “external identification” will be referred to as the *primary entry-point name*.
5. The **INITIAL**, **COMMON** and **RECURSIVE** clauses are used only within subprograms serving as subroutines. The **COMMON** clause should be used only within subprograms that are nested subprograms. The **INITIAL** clause, if specified, guarantees the subprogram will be in its initial (i.e. compiled) state each and every time it is executed, not just the first time. The **COMMON** clause may only be specified within a nested subprogram. A nested subprogram declared as **COMMON** may be called from any nested program in the source file being compiled, not just those “above” it in the nesting structure. The **RECURSIVE** clause, if any, marks a subprogram as being able to invoke itself. User-defined functions are always **RECURSIVE**.

See Also...

Subprograms Subroutines vs Functions [7.1](#)

Details Of Nested Subprograms [7.6](#)

Recursive Subprogramming [7.7](#)

4. ENVIRONMENT DIVISION

Figure 4-1 - ENVIRONMENT DIVISION Syntax

<u>ENVIRONMENT DIVISION.</u>	
<u>CONFIGURATION SECTION.</u>	
[<u>SOURCE-COMPUTER.</u>	<i>compilation-computer-specifications</i>]
[<u>OBJECT-COMPUTER.</u>	<i>execution-computer-specifications</i>]
[<u>REPOSITORY.</u>	<i>function-specifications</i>]
[<u>SPECIAL-NAMES.</u>	<i>program-configuration-specifications</i>]
<u>INPUT-OUTPUT SECTION.</u>	
[<u>FILE-CONTROL.</u>	<i>general-file-descriptions</i>]
[<u>I-O-CONTROL.</u>	<i>file-buffering-specifications</i>]

The **ENVIRONMENT DIVISION** defines the external computer environment in which the program will be operating. This includes defining any files that the program may be accessing.

1. If none of the features provided by the **ENVIRONMENT DIVISION** are required by a program, the **ENVIRONMENT DIVISION** may be omitted from the program.

4.1. CONFIGURATION SECTION

Figure 4-2 - CONFIGURATION SECTION Syntax

<u>CONFIGURATION SECTION.</u>	
[<u>SOURCE-COMPUTER.</u>	<i>compilation-computer-specifications</i>]
[<u>OBJECT-COMPUTER.</u>	<i>execution-computer-specifications</i>]
[<u>REPOSITORY.</u>	<i>function-specifications</i>]
[<u>SPECIAL-NAMES.</u>	<i>program-configuration-specifications</i>]

The **CONFIGURATION DIVISION** defines the computer system upon which the program is being compiled and executed and also specifies any special environmental configuration or compatibility characteristics.

1. The **CONFIGURATION SECTION** is not allowed in a nested subprogram – nested programs will inherit the **CONFIGURATION SECTION** settings of their parent program.
2. If none of the features provided by the **CONFIGURATION SECTION** are required by a program, the entire **CONFIGURATION SECTION** may be omitted from the program.
3. The sequence in which the **CONFIGURATION SECTION** paragraphs are specified is irrelevant.

See Also...

Details Of Nested Subprograms [7.6](#)

4.1.1. SOURCE-COMPUTER Paragraph

Figure 4-3 - SOURCE-COMPUTER Paragraph Syntax

<u>SOURCE-COMPUTER.</u>
<i>computer-name</i> [<u>WITH DEBUGGING MODE</u>] .

The **SOURCE-COMPUTER** paragraph defines the computer upon which the program is being compiled and provides one way in which debugging code imbedded within the program may be activated.

1. The **SOURCE-COMPUTER** paragraph is not allowed in a nested subprogram – nested programs will inherit the **SOURCE-COMPUTER** settings of their parent program.
2. The value specified for *computer-name* is irrelevant, provided it is a valid COBOL word that does not match any GNU COBOL reserved word. The *computer-name* may include spaces. This need not match the *computer-name* used with the **OBJECT-COMPUTER** paragraph, if any

- The **WITH DEBUGGING MODE** clause, if present, will signal the compiler that debugging lines – normally treated as comments - are to be compiled.
- Even without the **WITH DEBUGGING MODE** clause, it is still possible to compile debugging lines. Debugging lines may also be compiled by specifying the “-fdebugging-line” switch to the GNU COBOL compiler.

5. *See Also...*

Coding Comments in Programs [1.6](#)

Details Of Nested Subprograms [7.6](#)

4.1.2. OBJECT-COMPUTER Paragraph

Figure 4-4 - OBJECT-COMPUTER Paragraph Syntax

```

OBJECT-COMPUTER.
  [ computer-name ]
  [ MEMORY SIZE IS integer-1 { WORDS | CHARACTERS } ]
  [ PROGRAM COLLATING SEQUENCE IS alphabet-name-1 ]
  [ SEGMENT-LIMIT IS integer-2 ]
  [ CHARACTER CLASSIFICATION IS { locale-name-1 | LOCALE | USER-DEFAULT | SYSTEM-DEFAULT } ]
  .

```

The **OBJECT-COMPUTER** paragraph describes the computer upon which the program will execute. This paragraph is not merely documentation.

- The value specified for *computer-name*, if any, is irrelevant provided it is a valid COBOL word that does not match any GNU COBOL reserved word. The *computer-name* may include spaces. This need not match the *computer-name* used with the **SOURCE-COMPUTER** paragraph, if any
- The **OBJECT-COMPUTER** paragraph is not allowed in a nested subprogram – nested programs will inherit the **OBJECT-COMPUTER** settings of their parent program.
- The **MEMORY SIZE** and **SEGMENT-LIMIT** clauses are supported for compatibility purposes, but are non-functional in GNU COBOL.
- The **PROGRAM COLLATING SEQUENCE** clause allows you to specify a customized character collating sequence to be used when alphanumeric values are compared to one another. Data will still be stored in the character set native to the computer, but the logical sequence in which characters are ordered for comparison purposes can be altered from that inherent to the computer’s native character set. The *alphabet-name-1* you specify needs to be defined in the **SPECIAL-NAMES** paragraph.
- If no **PROGRAM COLLATING SEQUENCE** clause is specified, the collating sequence implied by the character set native to the computer (usually ASCII) will be used.
- The optional **CHARACTER CLASSIFICATION** clause may be used to specify a locale for the environment in which the program will be executing, for the purpose of influencing the uppercase and lowercase mappings of characters for the **UPPER-CASE** and **LOWER-CASE** intrinsic functions and the classification of characters for the **ALPHABETIC**, **ALPHABETIC-LOWER** and **ALPHABETIC-UPPER** class tests.

The definitions of these classes will be taken from the cultural convention specification (LC_CTYPE) from the specified locale.

The meanings of the four locale specifications are as follows:

- ▶ *locale-name-1* references a **LOCALE** definition that must occur within the **SPECIAL-NAMES** paragraph.
- ▶ The keyword **LOCALE** refers to the current locale (in effect at the time the program is executed)
- ▶ The keyword **USER-DEFAULT** references the default locale specified for the user currently executing this program.
- ▶ The keyword **SYSTEM-DEFAULT** denotes the default locale specified for the computer upon which the program is executing.

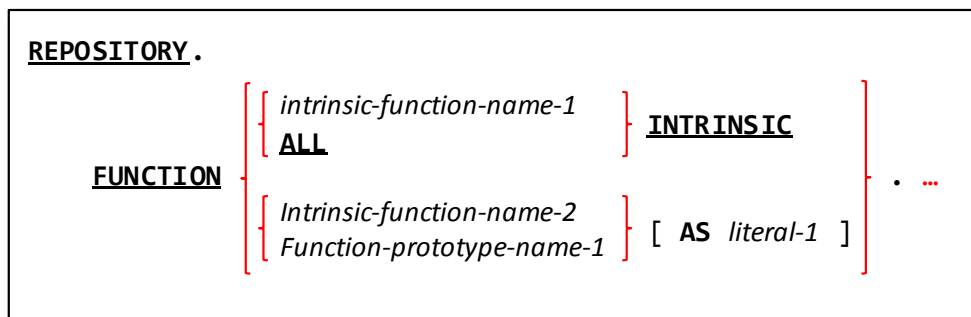
Absence of a **CHARACTER CLASSIFICATION** clause will cause character classification to occur according to the rules for the computer’s native character set (ASCII, EBCDIC, ...).

See Also...

The SPECIAL-NAMES Paragraph 4.1.4	UPPER-CASE Intrinsic Function 6.1.7.87
Class Tests 6.1.4.2.2	Details Of Nested Subprograms 7.6
LOWER-CASE Intrinsic Function 6.1.7.39	

4.1.3. REPOSITORY Paragraph

Figure 4-5 - REPOSITORY Paragraph Syntax



The **REPOSITORY** paragraph provides a mechanism for controlling access to the various built-in intrinsic functions and any user-defined functions that your program will be using.

1. The **REPOSITORY** paragraph is not allowed in a nested subprogram – nested programs will inherit the **REPOSITORY** settings of their parent program.
2. The “**INTRINSIC**” clause allows you to flag one or more (or **ALL**) built-in intrinsic functions as being usable without the need to code the keyword “**FUNCTION**” in front of the function names.
3. As an alternative to using the “**FUNCTION ALL INTRINSIC**” clause, you may instead compile your GNU COBOL programs using the “**-ffunctions-all**” switch.
4. The *function-prototype-name-1* option is required to specify the name of a user-defined function your program will be using. Optionally, should you desire, you may specify an alias name by which you will reference that user-defined function. Should you wish, you may also use the “**AS**” clause to provide an alias name for a built-in intrinsic function.

The following example accomplishes these objectives:

- ▶ It enables all intrinsic functions to be specified without the use of the “**FUNCTION**” keyword.
- ▶ It names two user-defined functions that will be used by the program: “MY-FUNCTION-1” and “USER-DEFINED-FUNCTION-NUMBER-2”
- ▶ It specifies the alias names “SIGMA” for the intrinsic function “STANDARD-DEVIATION” and “UDF2” for “USER-DEFINED-FUNCTION-NUMBER-2”.

```

REPOSITORY.
  FUNCTION ALL INTRINSIC.
  FUNCTION MY-FUNCTION-1.
  FUNCTION USER-DEFINED-FUNCTION-NUMBER-2 AS “UDF2”.
  FUNCTION STANDARD-DEVIATION AS “SIGMA”.
    
```

A SPECIAL NOTE ABOUT USER-DEFINED FUNCTIONS – because you must name a user-defined function that your program will be using in the **REPOSITORY** paragraph, you may always reference that function from your program’s **PROCEDURE DIVISION** without needing to use the “**FUNCTION**” keyword.

See Also...

Intrinsic Functions 6.1.7	Details Of Nested Subprograms 7.6
User-defined Functions 7.4.2	

4.1.4. SPECIAL-NAMES Paragraph

Figure 4-6 - SPECIAL-NAMES Paragraph Syntax

The **SPECIAL-NAMES** paragraph

SPECIAL-NAMES .

[**CALL-CONVENTION** *integer-1* **IS** *mnemonic-name-1*]

[**CONSOLE IS CRT**]

[**CRT STATUS IS** *identifier-1*]

[**CURRENCY SIGN IS** *literal-1*]

[**CURSOR IS** *identifier-2*]

[**DECIMAL-POINT IS COMMA**]

[**EVENT STATUS IS** *identifier-3*]

[**LOCALE** *locale-name-1* **IS** *literal-2*] ...

[**NUMERIC SIGN IS TRAILING SEPARATE**]

[**SCREEN CONTROL IS** *identifier-4*]

[*device-name-1* **IS** *mnemonic-name-2*] ...

[*feature-name-1* **IS** *mnemonic-name-3*] ...

[*alphabet-name-clause*] ...

[*class-definition-clause*] ...

[*switch-definition-clause*] ...

[*symbolic-characters-clause*] ...

•

provides a means for specifying various program and operating environment configuration options.

1. The **SPECIAL-NAMES** paragraph is not allowed in a nested subprogram – nested programs will inherit the **SPECIAL-NAMES** settings of their parent program.
2. The various clauses that may be specified within the **SPECIAL-NAMES** paragraph may be coded in any order.
3. Only the final clause specified within the **SPECIAL-NAMES** paragraph should be terminated with a period.
4. The **CALL-CONVENTION** clause allows a decimal integer, representing a series of ON/OFF switch settings, to be associated with a mnemonic name which may then be coded on **CALL** statements. The switch settings defined by this mnemonic will then control how the linkage to the subroutine (invoked by the **CALL** statement that references *mnemonic-name-1*) will be handled.
5. The **CONSOLE IS CRT** clause, if specified, will cause any **DISPLAY** or **ACCEPT** statements lacking explicit **UPON** clauses to be treated as full-screen **DISPLAYs** or **ACCEPTs**.
6. If the **CRT STATUS** clause is not specified, an implicit **COB-CRT-STATUS** identifier (with a **PICTURE** of 9(4)) will be allocated for the purpose of receiving screen **ACCEPT** statuses. If it is specified, then *identifier-1* must be defined in the program as a **PIC 9(4)** field.
7. The **CURRENCY SIGN** clause may be used to define any single character as the currency sign used in **PICTURE** symbol editing. The default currency sign is a dollar-sign (\$).
8. The **CURSOR IS** clause allows you to specify a 4- or 6-character data item into which the cursor screen location at the time a screen **ACCEPT** is satisfied. The value will be returned as *rrcc* or *rrccc*, depending upon the length of the specified *identifier-2*, where “*rr*” and “*rrr*” represent the row number (starting at zero) and “*cc*” and “*ccc*” represent the column number (also starting at zero). There is no default data item allocated for this data if the **CURSOR IS** clause is not specified.
9. The **DECIMAL POINT IS COMMA** clause reverses the definition of the “,” and “.” characters when they are used as **PICTURE** editing symbols and numeric literals. This can have unwanted side-effects.
10. The **LOCALE** clause may be used to associate external OS-defined locale names (*literal-6*) with an internal name (*locale-name-1*) that may then be referenced within the program. Locale names are defined by the Operating System and/or C compiler GNU COBOL will be utilizing on your computer.

The following table provides a list of possible locale codes, fgor example, that would be available on a Windows computer running a GNU COBOL that was built utilizing the MinGW Unix-emulator and the GNU C compiler (gcc):

Figure 4-7 – Typical Locale Codes

af_ZA	be_BY	en_CA	es_MX	ga_IE	kk_KZ	nl_NL	si_LK	tn_ZA
am_ET	bg_BG	en_GB	es_NI	gbz_AF	kl_GL	nn_NO	sk_SK	tr_IN
ar_AE	bn_IN	en_IE	es_PA	gl_ES	kn_IN	ns_ZA	sl_SI	tr_TR
ar_BH	bo_BT	en_IN	es_PE	gsw_FR	ko_KR	oc_FR	sma_NO	tt_RU
ar_DZ	bo_CN	en_JM	es_PR	gu_IN	kok_IN	or_IN	sma_SE	ug_CN
ar_EG	br_FR	en_MY	es_PY	ha_Latn_NG	ky_KG	pa_IN	smj_NO	uk_UA
ar_IQ	bs_Cyrl_BA	en_NZ	es_SV	he_IL	lb_LU	pl_PL	smj_SE	ur_PK
ar_JO	bs_Latn_BA	en_PH	es_US	hi_IN	lo_LA	ps_AF	smn_FI	uz_Cyrl_UZ
ar_KW	ca_ES	en_SG	es_UY	hr_BA	lt_LT	pt_BR	sms_FI	uz_Latn_UZ
ar_LB	cs_CZ	en_TT	es_VE	hr_HR	lv_LV	pt_PT	sq_AL	vi_VN
ar_LY	cy_GB	en_US	et_EE	hu_HU	mi_NZ	qut_GT	sr_Cyrl_BA	wen_DE
ar_MA	da_DK	en_ZA	eu_ES	hy_AM	mk_MK	quz_BO	sr_Cyrl_CS	wo_SN
ar_OM	de_AT	en_ZW	fa_IR	id_ID	ml_IN	quz_EC	sr_Latn_BA	xh_ZA
ar_QA	de_CH	es_AR	fi_IR	ig_NG	mn_Cyrl_MN	quz_PE	sr_Latn_CS	yo_NG
ar_SA	de_DE	es_BO	fil_PH	ii_CN	mn_Mong_CN	rm_CH	sv_FI	zh_CN
ar_SY	de_LI	es_CL	fo_FO	is_IS	moh_CA	ro_RO	sv_SE	zh_HK
ar_TN	de_LU	es_CO	fr_BE	it_CH	mr_IN	ru_RU	sw_KE	zh_MO
ar_YE	dsb_DE	es_CR	fr_CA	it_IT	ms_BN	rw_RW	syr_SY	zh_SG
arn_CL	dv_MV	es_DO	fr_CH	iu_Cans_CA	ms_MY	sa_IN	ta_IN	zh_TW
as_IN	el_GR	es_EC	fr_FR	iu_Latn_CA	mt_MT	sah_RU	te_IN	zu_ZA
az_Cyrl_AZ	en_029	es_ES	fr_LU	ja_JP	nb_NO	se_FI	tg_Cyrl_TJ	
az_Latn_AZ	en_AU	es_GT	fr_MC	ka_GE	ne_NP	se_NO	th_TH tk_TM	
ba_R	en_BZ	es_HN	fy_NL	kh_KH	nl_BE	se_SE	tmz_Latn_DZ	

- The **NUMERIC SIGN IS TRAILING SEPARATE** specification causes all signed numeric **USAGE DISPLAY** data items to be created as if the **SIGN IS TRAILING SEPARATE CHARACTER** clause was included in their definitions.
- While the **SCREEN CONTROL** and **EVENT STATUS** clauses are clearly noted at compilation time as being unsupported, the **CURSOR IS** clause is not; currently, however, it appears to be non-functional at runtime.
- The “*device-name IS mnemonic-name-2*” clause allows you to specify an alternate name for one of the built-in GNU COBOL device names specified before the “**IS**”. The list of device names built-into GNU COBOL, and the physical device associated with that name, are as follows:

Figure 4-8 - Built-In GNU COBOL Device Names

Built-In GNU COBOL Device Name	Associated Actual Device
CONSOLE	This is the (screen-mode) display of the PC or Unix system
STDIN SYSIN SYSIPT	Standard system input (pipe 0). On a PC or UNIX system, this is typically the keyboard. Can be specified to a GNU COBOL program from a file by adding the sequence “ 0< filename ” to the end of the programs execution command.
PRINTER STDOUT SYSLIST SYSLST SYSOUT	Standard system output (pipe 1). On a PC or UNIX system, this is typically the display. Can be sent to a file by adding the sequence “ 1> filename ” to the end of the programs execution command.
STDERR SYSERR	Standard system error output (pipe 2). On a PC or UNIX system, this is typically the display. Can be sent to a file by adding the sequence “ 2> filename ” to the end of the programs execution command.

- The “*feature-name-1 IS mnemonic-name-3*” clause allow for mnemonic names to be assigned to up to the 13 printer channel (i.e. vertical page positioning) position feature names “**C01**” through “**C12**” and “**CSP**”. Once a channel position has been assigned a mnemonic name, statements of the form “**WRITE record-name AFTER ADVANCING mnemonic-name-3**”⁹ may be coded to write the specified print record at the channel position assigned to mnemonic-name-3.

Printers supporting channel positioning are generally mainframe-type line printers. When writing to printers that do not support channel positioning, a formfeed will be issued to the printer.

⁹ **BEFORE ADVANCING** is possible also. See the **WRITE** statement in section 6.2.50 for additional information.

The **CSP** positioning option stands for “No Spacing”. Testing on a MinGW build of GNU COBOL shows that this too results in a formfeed being issued.

See Also...

Using Commas and Semicolons 1.7	The ACCEPT Statement (Screen Data) 6.4.1.4
OBJECT-COMPUTER And LOCALES 4.1.2	The CALL Statement 6.4.5
Defining a Data Item’s PICTURE 5.2.1.6	Details Of Nested Subprograms 7.6

4.1.4.1. The alphabet-name Clause

Figure 4-9 - The SPECIAL-NAMES "alphabet-name" Clause

```
ALPHABET alphabet-name-1 IS
{
  NATIVE
  STANDARD-1
  STANDARD-2
  EBCDIC
  literal-1 [ { THRU | THROUGH literal-2 } ] ...
  { ALSO literal-3 } ...
}
```

The **ALPHABET** clause provides a means for relating a name to a specified character code set or collating sequence, including those you define yourself using the “*literal-1*” option. You may specify an alphanumeric literal for any of the *literal-1*, *literal-2* or *literal-3* specifications. You may also specify any of the figurative constants **SPACE**, **SPACES**, **ZERO**, **ZEROS**, **ZEROES**, **QUOTE**, **QUOTES**, **HIGH-VALUE**, **HIGH-VALUES**, **LOW-VALUE** or **LOW-VALUES**.

1. The reserved word “**THROUGH**” may be used interchangeably with “**THRU**”.

4.1.4.2. The class-name Clause

Figure 4-10 - The SPECIAL-NAMES "class-name" Clause

```
CLASS class-name-1 IS
  { literal-1 [ THRU | THROUGH literal-2 ] } ...
```

User-defined classes are defined using the **CLASS** clause.

1. The reserved word **THROUGH** may be used interchangeably with **THRU**.
2. Both *literal-1* and *literal-2* must be alphanumeric literals of length 1.
3. The literal(s) specified on that clause define the possible characters that may be found in a data item’s value in order to be considered part of the class.

For example, the following defines a class called “Hexadecimal”, the definition of which specifies the only characters that may be present in an alphanumeric data item if that data item is to be part of the “Hexadecimal” class:

```
CLASS Hexadecimal IS ‘0’ THRU ‘9’
                       ‘A’ THRU ‘F’
                       ‘a’ THRU ‘f’
```

4. See section for an example of how this user-defined class might be used.

*See Also...*Class Tests [6.1.4.2.2](#)

4.1.4.3. The switch-definition Clause

Figure 4-11 - The SPECIAL-NAMES "switch-definition" Clause

```

switch-name-1 [ IS mnemonic-name-1 ]
[ { ON STATUS IS condition-name-1 } ... ]
[ { OFF STATUS IS condition-name-2 } ... ]

```

The *switch-definition* clause associates a *condition-name* with a run-time execution switch so that the status of that switch may be tested from within a program.

1. The valid switch-names are **SWITCH-0** through **SWITCH-15**.
2. If the program is compiled with the "**-fsyntax-extension**" compiler switch, the switch names "**SW0**" through "**SW15**" are also valid; they correspond to "**SWITCH-0**" through "**SWITCH-15**", respectively.
3. At execution time, each switch will be associated with an environment variable named "**COB_SWITCH_n**", where "*n*" will have the value "0" through "15". Any of these sixteen environment variables that have the value "ON" (regardless of upper- or lower-case value) will be considered to be set "on". Any of these sixteen environment variables having no value at all or a value other than "ON" will be considered "off".
4. Each specified switch must have at least one of a "**IS mnemonic-name**", **ON STATUS** or an **OFF STATUS** option defined for it (otherwise there will be no way to reference the switch from within a GNU COBOL program).
5. The "**IS mnemonic-name**" syntax provides a means for setting the switch to either an **ON** or **OFF** value via the **SET** statement.
6. The **ON STATUS** and **OFF STATUS** syntax provides a way of associating a condition-name with either the on or off status of the switch, so that status may be tested at execution time via the **IF** statement.

*See Also...*Condition Names [6.1.4.2.1](#)Switch-Status Conditions [6.1.4.2.4](#)The **IF** Statement [6.2.21](#)The **SET SWITCH** Statement [6.4.39.7](#)

4.1.4.4. The symbolic-characters clause

Figure 4-12 - The SPECIAL-NAMES "symbolic-characters" Clause

SYMBOLIC CHARACTERS

```

{ { symbolic-character-1 } ... ARE { integer-1 } ... } ...
[ IN alphabet-name-1 ]

```

The **SYMBOLIC CHARACTERS** clause may be used to define your own figurative constants.

1. The word **IS** may be substituted for the word **ARE**, if desired.
2. There must be exactly as many *integer-1* values specified after the word **ARE** (or **IS**) as there are *symbolic-character-1* names specified before it.
3. Each symbolic character name will be associated with the corresponding "*integer-1*"th character in the alphabet named in the **IN** clause. The integer values are selecting characters from the alphabet by their ordinal position and not by their numeric value; thus, an integer of 15 will select the 15th character in the specified alphabet, regardless of the actual numeric value of the bit pattern that constitutes that character.
4. If no *alphabet-name-1* is specified, the systems native character set will be assumed.

The following two code examples define the same set of figurative constant names for five ASCII control characters (assuming that ASCII is the system's native character set). The two examples are identical in their effects, even though the manner in which the figurative constants are defined is different.

SPECIAL-NAMES.

```

SYMBOLIC CHARACTERS NUL IS 1
                    SOH IS 2
                    BEL IS 8
                    DC1 IS 18

```

SPECIAL-NAMES.

```

SYMBOLIC CHARACTERS NUL SOH BEL DC1 DC2
                    ARE 1 2 8 18 19.

```

DC2 IS 19.

4.2. INPUT-OUTPUT SECTION

Figure 4-13 - INPUT-OUTPUT SECTION Syntax

```
INPUT-OUTPUT SECTION.  
[ FILE-CONTROL.      general-file-descriptions ]  
[ I-O-CONTROL.      file-buffering-specifications ]
```

The **INPUT-OUTPUT** section provides for the definition of any files the program will be accessing as well as control of the I/O buffering process against those files.

1. If the compiler “config” file you are using has “relaxed-syntax-check” set to “yes”, the **FILE-CONTROL** and **I-O-CONTROL** paragraphs may be specified without the **INPUT-OUTPUT SECTION** header having been specified.
2. If the program uses no files, it needs neither a **FILE-CONTROL** or **I-O-CONTROL** paragraph.

See Also...

GNU COBOL “config” Files [8.1.6](#)

4.2.1. File SELECT Statement

Figure 4-14 – File SELECT Statement Syntax

```

SELECT [ [ NOT ] OPTIONAL ] file-name-1

[ ASSIGN TO [ [ EXTERNAL ] ] [ DYNAMIC ] ] [ DISC | DISK | TAPE | RANDOM | DISPLAY | KEYBOARD | LINE ADVANCING | PRINTER ] [ { literal-1 } | { identifier-1 } ]

[ [ FILE | SORT ] STATUS IS identifier-2 [ identifier-3 ] ]

[ COLLATING SEQUENCE IS alphabet-name-1 ]

[ LOCK MODE IS [ EXCLUSIVE | MANUAL | AUTOMATIC ] [ WITH LOCK ON MULTIPLE RECORDS | WITH LOCK ON RECORD | WITH ROLLBACK ] ]

[ RECORD DELIMITER IS STANDARD-1 ]

[ RESERVE integer-1 AREAS ]

[ SHARING WITH [ ALL OTHER | NO OTHER | READ ONLY ] ]

[ organization-clause ] .

```

The **SELECT** statement of the **FILE-CONTROL** paragraph creates a definition of a file and links that COBOL definition to the external operating system environment .

What is shown here are those clauses of the **SELECT** statement that are common to all types of files.

Upcoming sections will discuss special **SELECT** clauses that only pertain to certain types of files.

1. The **COLLATING SEQUENCE**, **RECORD DELIMITER**, **RESERVE** and **SHARING WITH ALL OTHER** clauses, as well as the specification of a secondary **FILE-STATUS** field and **LOCK MODE ... WITH ROLLBACK**, while syntactically recognized, are not currently supported by GNU COBOL.
2. The **OPTIONAL** clause, to be used only for files that will be used to provide input data to the program, indicates the file may or may not actually be available at run-time. Attempts to **OPEN** an **OPTIONAL** file when the file does not exist will receive a special non-fatal file status value (see status 05 in [Figure 4-15](#) below) indicating the file is not available; a subsequent attempt to **READ** that file will return an **AT END** (end-of-file) condition. Optionally, files may be designated as **NOT OPTIONAL**, if desired. This is useful when specifying the “-foptional-file” compiler switch.
3. The *file-name-1* value that you specify will be the name by which you will reference the file within your program. This name should be formed according to the rules for user-defined names.
4. The **EXTERNAL** option flags the file as being sharable with other GNU COBOL programs that include the same **SELECT** statement. Those other programs must either be executed as subprograms from this one or must execute this one as a subprogram. Once an **EXTERNAL** file has been **OPENed** by one of the programs **SELECTing** the **EXTERNAL** file, that file is available for **READING**, **WRITEing** and the like from any of the programs that share it. Similarly, once one program **CLOSEs** the file, no other program sharing that file may access the file further unless the file is re-**OPENed**.

5. The **DYNAMIC** option specifies that the actual pathname of the file being **SELECT**ed will be specified at execution time as the contents of *identifier-1*. If you use the **DYNAMIC** option, you must specify *identifier-1*. If you specify *identifier-1* on the **SELECT**, the **DYNAMIC** option will be assumed if not specified.
6. Optionally, you may define the type of device the file will be assigned to, as follows.
 - a. The **DISK** and **DISC** devices (the two are synonymous with one another) are typically used in conjunction with a "*literal-1*" or "*identifier-1*" option. If neither the "*literal-1*" nor "*identifier-1*" option is provided, the **SELECT** will reference a file named "*file-name-1*" in whatever folder is current at the time the file is **OPEN**ed.
 - b. The **TAPE** and **RANDOM** devices behave in a manner similar to **DISC** (or **DISK**) and are included into GNU COBOL to facilitate the compilation of COBOL source from other COBOL implementations.
 - c. The **KEYBOARD**, **DISPLAY** and **PRINTER** devices refer to the PC keyboard and display and STDOUT devices, respectively. When either *literal-1* or *identifier-1* are specified with these device types, the effect will be the same as if **DISC** or **DISK** had been used. When neither *literal-1* nor *identifier-1* are used, these devices will be associated with the **STDIN** (**KEYBOARD**) and **STDOUT** (**DISPLAY** or **PRINTER**) devices, respectively (see [Figure 4-8](#)).
 - d. A file **ASSIGN**ed to the **PRINTER** device must be defined with an **ORGANIZATION IS LINE SEQUENTIAL** (if no **ORGANIZATION** is specified, **LINE SEQUENTIAL** will be assumed).
 - e. The **LINE ADVANCING** device defines the file as a special form of **LINE SEQUENTIAL** file. When this device is used, either *literal-1* or *identifier-1* must be specified.
7. The "*identifier-1*" option references an alphanumeric data item, the contents of which at the time the file is **OPEN**ed will define the path and filename of the actual data file to be processed.
8. If the "*literal-1*" option is used on the **ASSIGN** clause, it defines the linkage of the COBOL file to an actual operating system file as follows:
 - a. If an environment variable named "*DD_literal-1*" exists, its value will be treated as the full path/filename of the file. If not, then ...
 - b. If an environment variable named "*dd_literal-1*" exists, its value will be treated as the full path/filename of the file. If not, then ...
 - c. If an environment variable named "*literal-1*" exists, its value will be treated as the full path/filename of the file. If not, then...
 - d. The literal itself will be treated as the full path/filename to the file.

This behavior will be influenced by the "filename-mapping" setting in the config file you are using when compiling your programs. The behavior stated above applies only if "filename-mapping: yes" is in-effect. If "filename-mapping: no" is used, only the last option (treating the literal itself as the full name of the file) is possible.
9. The **FILE STATUS** or **SORT STATUS** clause (they are both equivalent and only one or the other, if any, should be specified) is used to specify the name of a **PIC 9(2)** data item into which an I/O status code will be saved after every I/O verb that is executed against the file. This does not actually allocate the data item – you still need to allocate the item yourself somewhere in the **DATA DIVISION**.

10. Possible status codes that can be returned to a **FILE STATUS** data item are as follows:

Figure 4-15 – FILE STATUS Values

Status Value	Meaning	Status Value	Meaning
00	Success	39	Conflicting attribute
02	Success (Duplicate Record Key Written)	41	File already OPEN
05	Success (Optional File Not Found)	42	File not OPEN
07	Success (No Unit)	43	Read not done
10	End of file reached if READING forward or beginning-of-file reached if READING backward	44	Record overflow
14	Out of key range	46	READ error
21	Key invalid	47	OPEN INPUT denied
22	Attempt to duplicate key value	48	OPEN OUTPUT denied
23	Key not found	49	OPEN I-O denied
30	Permanent I/O error	51	Record locked
31	Inconsistent filename	52	End of page
34	Boundary violation	57	LINAGE specifications invalid
35	File not found	61	File sharing failure
37	Permission denied	91	File not available
38	Closed with lock		

11. The **LOCK** and **SHARING** clauses define the conditions under which this file will be usable by other programs executing concurrently with this one.

See Also...

Types of Files	1.3.3.5	The OPEN Statement	6.4.29
User-defined Names	1.10	The READ Statement	6.4.31
File Sharing	6.1.9.1	Compiler Switches Reference	8.1.2
Record Locking	6.1.9.2	GNU COBOL “config” Files	8.1.6
Handling End-of-File Conditions (AT END)	6.1.12.1		

4.2.1.1. SELECT Without an “organization-clause”

A SELECT statement coded without an ORGANIZATION explicitly coded will be handled as if the following ORGANIZATION clause had been specified:

```
ORGANIZATION IS RECORD BINARY SEQUENTIAL
ACCESS MODE IS SEQUENTIAL
PADDING CHARACTER IS “ “
```

4.2.1.2. ORGANIZATION SEQUENTIAL Files

Figure 4-16 - SELECT “organization-options” For SEQUENTIAL Files

```
[ ORGANIZATION IS ] RECORD BINARY SEQUENTIAL
[ ACCESS MODE IS SEQUENTIAL ]
```

Files declared as **ORGANIZATION RECORD BINARY SEQUENTIAL** will consist of records with no explicit end-of-record delimiter character sequences; records in such files are “delineated” by a calculated byte-offset (based on record length) into the file .

1. The keyword “**ORGANIZATION**” is optional to provide compatibility with those (few) COBOL implementations that consider that word to be optional. Most COBOL implementations do require the word **ORGANIZATION**, so it should be used in new programs.

- These files cannot be prepared with any standard text-editing or word processing software as all such programs will imbed delimiter characters at the end of records. Such files may contain either **USAGE DISPLAY** or **USAGE COMPUTATIONAL** (of any variety) data since no character sequence can be accidentally interpreted as an end-of-record delimiter.
- Both fixed- and variable-length record formats are supported. Variable-length records will always be written in their maximum size, however.
- Specifying **ORGANIZATION IS RECORD BINARY SEQUENTIAL** is the same as specifying **ORGANIZATION SEQUENTIAL**.
- The **ACCESS MODE IS SEQUENTIAL** clause is optional because, if absent, it will be assumed anyway for this type of file. The internal structure of **RECORD BINARY SEQUENTIAL** files is such that the data in those files can only be processed in a sequential manner; in order to read the 100th record in such a file, for example, you first must read records 1 through 99.
- SEQUENTIAL** files are processed using the **CLOSE**, **COMMIT**, **DELETE**, **MERGE**, **OPEN**, **READ**, **REWRITE**, **SORT**, **UNLOCK** and **WRITE** statements.

See Also...

Types of Files	1.3.3.5	The OPEN Statement	6.4.29
Storage Format of Data (USAGE)	5.2.1.11	The READ Statement	6.4.31
Handling End-of-File Conditions (AT END)	6.1.12.1	The REWRITE Statement	6.4.36
The CLOSE Statement	6.4.7	The SORT Statement (File Sort)	6.4.40.1
The COMMIT Statement	6.4.8	The UNLOCK Statement	6.4.48
The DELETE Statement	6.4.11	The WRITE Statement	6.4.50
The MERGE Statement	6.4.25		

4.2.1.3. ORGANIZATION LINE SEQUENTIAL Files

Figure 4-17 - SELECT "organization-options" for LINE SEQUENTIAL Files

```
[ ORGANIZATION IS ] LINE SEQUENTIAL
[ ACCESS MODE IS SEQUENTIAL ]
[ PADDING CHARACTER IS { literal-1
                           { identifier-1 } } ]
```

Files declared as **ORGANIZATION LINE SEQUENTIAL** will consist of records terminated by an end-of-record delimiter character or character sequence.

- The keyword "**ORGANIZATION**" is optional to provide compatibility with those (few) COBOL implementations that consider that word to be optional. Most COBOL implementations do require the word **ORGANIZATION**, so it should be used in new programs.
- This is the only **ORGANIZATION** valid for files that are assigned to the **PRINTER** device.
- These files could be prepared with any standard text-editing or word processing software capable of writing text files. Such files should not contain any **USAGE COMPUTATIONAL** or **BINARY** (of any variety) data since such fields could accidentally contain byte sequences that could be interpreted as an end-of-record delimiter.
- Both fixed- and variable-length record formats are supported.
- The end-of-record delimiter sequence will be X'0A' (an ASCII line-feed character) or a X'0D0A' (an ASCII carriage-return/line-feed sequence).
- The **PADDING CHARACTER** clause, while syntactically recognized, is currently non-functional.
- When reading a **LINE SEQUENTIAL** file, records in excess of the size implied by the file's FD will be truncated while records shorter than that size will be padded to the right with **SPACES**.
- The **ACCESS MODE IS SEQUENTIAL** clause is optional because, if absent, it will be assumed anyway for this type of file. The internal structure of **LINE SEQUENTIAL** files is such that the data in those files can only be processed in a sequential manner; in order to read the 100th record in such a file, for example, you first must read records 1 through 99.

9. Files ASSIGNED to PRINTER or CONSOLE should be specified as ORGANIZATION LINE SEQUENTIAL.
10. **LINE SEQUENTIAL** files are processed using the **CLOSE**, **COMMIT**, **DELETE**, **MERGE**, **OPEN**, **READ**, **REWRITE**, **SORT**, **UNLOCK** and **WRITE** statements.

See Also...

Types of Files	1.3.3.5
Storage Format of Data (USAGE)	5.2.1.11
Handling End-of-File Conditions (AT END)	6.1.12.1
The CLOSE Statement	6.4.7
The COMMIT Statement	6.4.8
The DELETE Statement	6.4.11
The MERGE Statement	6.4.25

The OPEN Statement	6.4.29
The READ Statement	6.4.31
The REWRITE Statement	6.4.36
The SORT Statement (File Sort)	6.4.40.1
The UNLOCK Statement	6.4.48
The WRITE Statement	6.4.50

4.2.1.4. ORGANIZATION RELATIVE Files

Figure 4-18 - SELECT "organization options" For RELATIVE Files

[ORGANIZATION IS] RELATIVE
[ACCESS MODE IS { SEQUENTIAL DYNAMIC RANDOM }]
[RELATIVE KEY IS <i>identifier-1</i>]

RELATIVE files are files with an internal organization such that records may be processed in a sequential manner based upon their physical location in the file or in a random manner by allowing records to be read, written or updated by specifying the relative record number in the file.

1. The keyword "**ORGANIZATION**" is optional to provide compatibility with those (few) COBOL implementations that consider that word to be optional. Most COBOL implementations do require the word **ORGANIZATION**, so it should be used in new programs.
2. **ORGANIZATION RELATIVE** files cannot be assigned to **CONSOLE**, **DISPLAY**, **LINE ADVANCING** or **PRINTER**.
3. The **RELATIVE KEY** clause is optional only if **ACCESS MODE SEQUENTIAL** is specified.
4. While records in a **ORGANIZATION RELATIVE** file may be defined as having variable-length records, the file will be structured in such a manner as to reserve the maximum possible space for each record.
5. An **ACCESS MODE** of **SEQUENTIAL** indicates that the records of the file will be processed in a sequential manner, according to their physical sequence in the file.
An **ACCESS MODE** of **RANDOM** means that records will be processed in random sequence by specifying their record number in the file every time the file is read or written.
A **DYNAMIC ACCESS MODE** indicates the program will switch back and forth between **SEQUENTIAL** and **RANDOM** mode during execution. The file starts out initially in **SEQUENTIAL** mode when first **OPENED** but the program may use the **START** verb to switch between the other two access modes.
6. The default **ACCESS MODE** is **SEQUENTIAL**.
7. The **RELATIVE KEY** data item is a numeric data item that cannot be a field within records of this file. Its purpose is to return the current relative record number of a **RELATIVE** file that is being processed in **SEQUENTIAL** access mode and to be a retrieval key that specifies the relative record number to be read or written when processing a **RELATIVE** file in **RANDOM** access mode.
8. **RELATIVE** files are processed using the **CLOSE**, **COMMIT**, **DELETE**, **MERGE**, **OPEN**, **READ**, **REWRITE**, **SORT**, **START**, **UNLOCK** and **WRITE** statements.

See Also...

Types of Files	1.3.3.5
Handling End-of-File Conditions (AT END)	6.1.12.1
The CLOSE Statement	6.4.7

The READ Statement	6.4.31
The REWRITE Statement	6.4.36
The SORT Statement (File Sort)	6.4.40.1

The **COMMIT** Statement [6.4.8](#)The **DELETE** Statement [6.4.11](#)The **MERGE** Statement [6.4.25](#)The **OPEN** Statement [6.4.29](#)The **START** Statement [6.2.41](#)The **UNLOCK** Statement [6.4.48](#)The **WRITE** Statement [6.4.50](#)

4.2.1.5. ORGANIZATION INDEXED Files

Figure 4-19 - SELECT "organization options" For INDEXED Files

```
[ ORGANIZATION IS ] INDEXED
  [ ACCESS MODE IS { SEQUENTIAL
                    { DYNAMIC
                    { RANDOM } } ]
  RECORD KEY IS identifier-1 [ { = SOURCE IS } identifier-2 ]
  [ ALTERNATE RECORD KEY IS identifier-3 [ { = SOURCE IS } identifier-4 ] ...
    [ WITH DUPLICATES ]
```

INDEXED files, like **RELATIVE** files, may have their records processed either sequentially or in a random manner. Unlike **RELATIVE** files, however, the actual location of a record in an **INDEXED** file is based upon the value(s) of one or more alphanumeric fields within records of the file.

For example, an **INDEXED** file containing product data might use the product identification code as a **RECORD KEY**. This means you may read, write or update the "A6G4328"th record or the "Z8X7723"th record directly, based upon the product id value of those records!

1. The keyword "**ORGANIZATION**" is optional to provide compatibility with those (few) COBOL implementations that consider that word to be optional. Most COBOL implementations do require the word **ORGANIZATION**, so it should be used in new programs.
2. **ORGANIZATION INDEXED** files cannot be assigned to **CONSOLE**, **DISPLAY**, **LINE ADVANCING** or **PRINTER**.
3. The specification of so-called "split keys", while syntactically recognized (the "= / SOURCE IS" clauses), are not currently supported by GNU COBOL.
4. An **ACCESS MODE** of **SEQUENTIAL** indicates that the records of the file will be processed in a sequential manner with respect to the values of the **RECORD KEY** or an **ALTERNATE RECORD KEY**.

An **ACCESS MODE** of **RANDOM** means that records will be processed in random sequence by accessing the record with specific **RECORD KEY** or **ALTERNATE RECORD KEY** values.

DYNAMIC ACCESS MODE allows the file will be processed either in **RANDOM** or **SEQUENTIAL** mode; the program may switch between the two modes as needed. The **START** verb is used to make the switch between modes.

5. The default **ACCESS MODE** is **SEQUENTIAL**.
6. The **PRIMARY KEY** clause defines the field(s) within the record used to provide the primary access to records within the file. No two records may have the same **PRIMARY KEY** field value.
7. The **ALTERNATE RECORD KEY** clause, if used, defines an additional field within the record that provides an alternate means of directly accessing records or an additional field by which the file's contents may be processed sequentially. You have the choice of allowing records to have duplicate alternate key values, if necessary.
8. There may be multiple **ALTERNATE RECORD KEY** clauses, each defining an additional alternate key for the file.
9. **INDEXED** files are processed using the **CLOSE**, **COMMIT**, **DELETE**, **MERGE**, **OPEN**, **READ**, **REWRITE**, **SORT**, **START**, **UNLOCK** and **WRITE** statements.

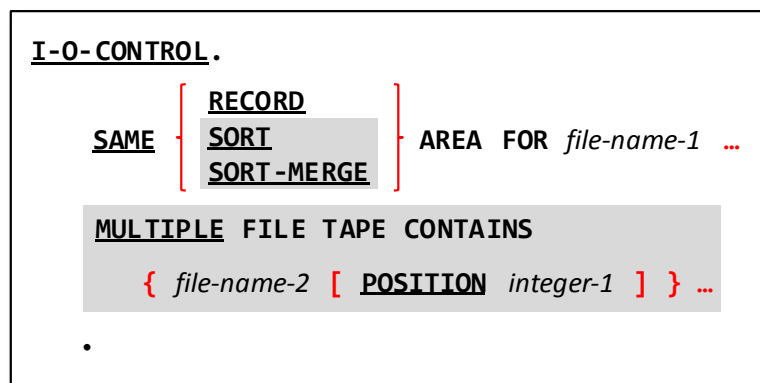
See Also...

Types of Files	1.3.3.5
Handling End-of-File Conditions (AT END)	6.1.12.1
The CLOSE Statement	6.4.7
The COMMIT Statement	6.4.8
The DELETE Statement	6.4.11
The MERGE Statement	6.4.25
The OPEN Statement	6.4.29

The READ Statement	6.4.31
The REWRITE Statement	6.4.36
The SORT Statement (File Sort)	6.4.40.1
The START Statement	6.2.41
The UNLOCK Statement	6.4.48
The WRITE Statement	6.4.50

4.2.2. I-O-CONTROL Paragraph

Figure 4-20 - I-O-CONTROL Paragraph Syntax

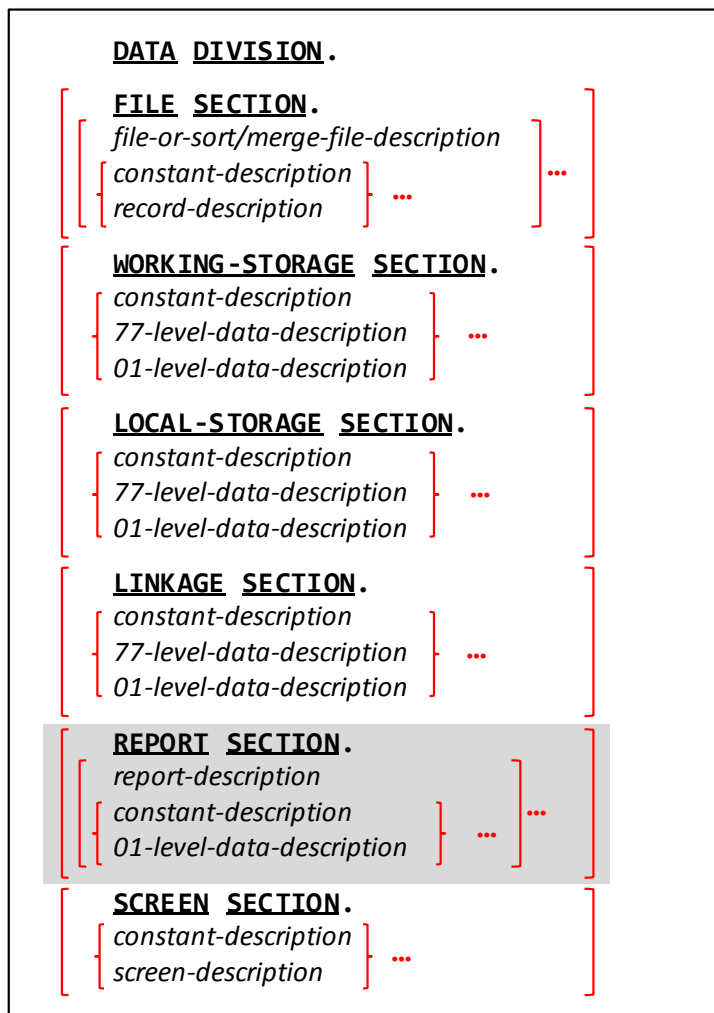


The **I-O-CONTROL** Paragraph can be used to optimize certain aspects of file processing.

1. The **SAME SORT AREA** and **SAME SORT-MERGE AREA** clauses are non-functional. The **SAME RECORD AREA** is functional, however.
2. The **MULTIPLE FILE TAPE** clause is obsolete and is therefore recognized but not functional.
3. The **SAME RECORD AREA** clause allows you to specify that multiple files should share the same input and output memory buffers. These buffers can sometimes get quite large, and by having multiple files share the same buffer memory you may significantly cut down the amount of memory the program is using (thus making “room” for more procedural code or data). If you do use this feature, take care to ensure that no more than one of the specified files are ever **OPEN** simultaneously.

5. DATA DIVISION

Figure 5-1 - General DATA DIVISION Format



The **DATA DIVISION** is used to define all data that will be processed by a program. The contents of the various sections are as follows:

FILE SECTION

Provides a detailed specification as to the blocking characteristics and record layouts of each SELECTed file.

WORKING-STORAGE SECTION

Definitions of the various internal data items used by the program.

LOCAL-STORAGE SECTION

Similar to WORKING-STORAGE, but describes data within a subprogram that will be dynamically allocated and initialized (automatically) each time the subprogram is executed (WORKING-STORAGE is automatically initialized only the 1st time a subprogram is executed).

LINKAGE SECTION

Describes data within a subprogram that serves as input arguments to or output arguments from the subprogram.

REPORT SECTION

Describes the layout of printed reports as well as many of the functional aspects of the generation of reports.

SCREEN SECTION

Describes the visual layout of entire screens.

1. Any **SECTION**s that are used must be specified in the order shown. If no **DATA DIVISION** sections are needed, the **DATA DIVISION** header itself may be omitted.
2. The **REPORT SECTION** is syntactically recognized but will – if used – be rejected as unsupported. GNU COBOL does not support the RWCS¹⁰ (it does support the **LINAGE** clause in an FD, however).
3. **LOCAL-STORAGE** cannot be used in nested subprograms.

See Also...

A Sample GNU COBOL Screen [1.3.3.9](#)

Defining Data Items [5.2](#)

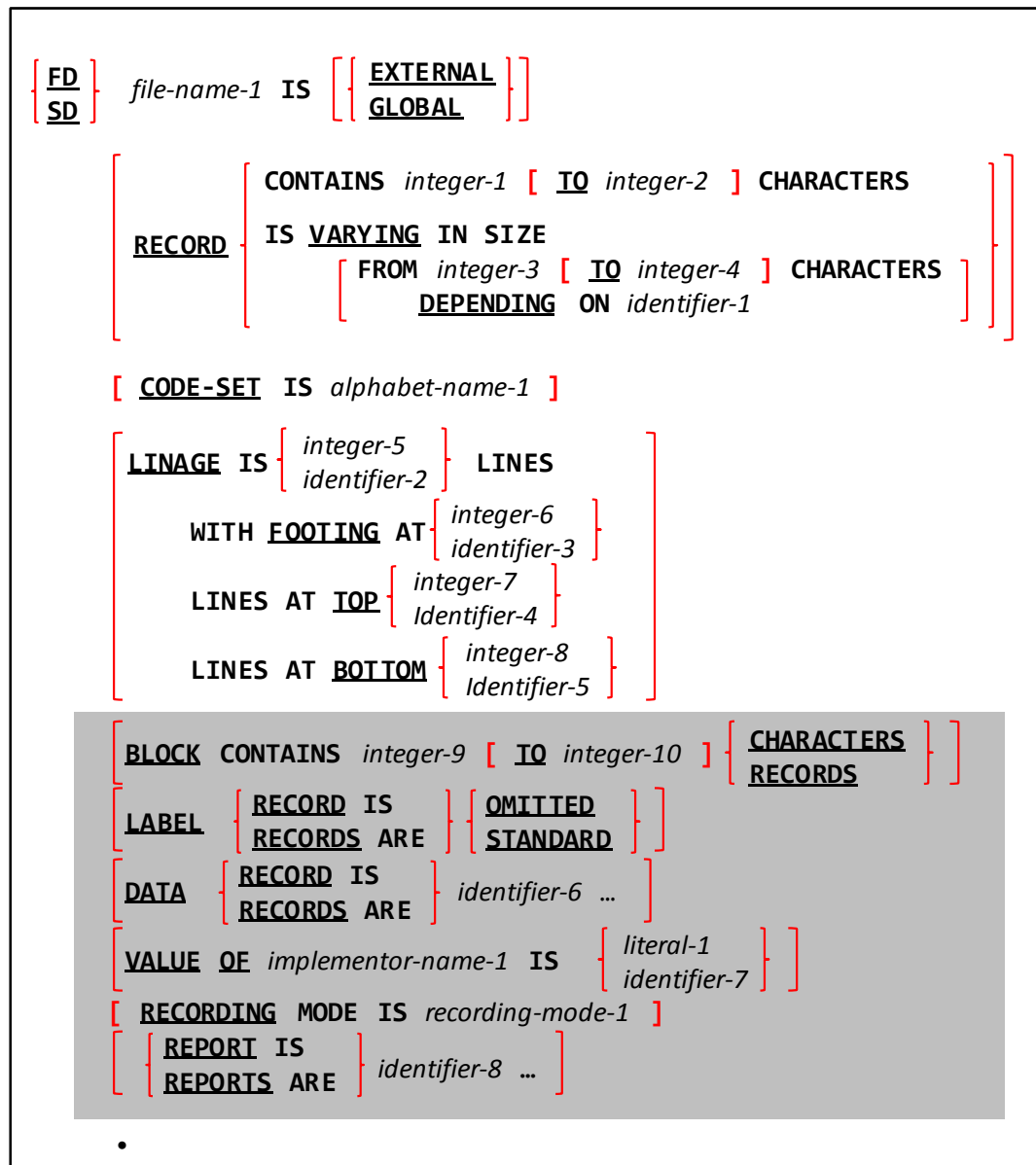
Defining Screens [5.2.2](#)

¹⁰ Report-Writer Control System

5.1. File Or Sort/Merge File Descriptions

Every file that has been **SELECT**ed in the **FILE-CONTROL** paragraph must be described in the **FILE SECTION** of the **DATA DIVISION**. Files destined for use as sort/merge work files must be described with a Sort/Merge File Description (**SD**) while every other file is described with a File Description (**FD**). Each of these descriptions will be followed with at least one Record Description.

Figure 5-2 - File Description (FD) and Sort Description (SD) Syntax



There must be a detailed description for every file **SELECT**ed in your program. These detailed descriptions will be coded in the **FILE SECTION**.

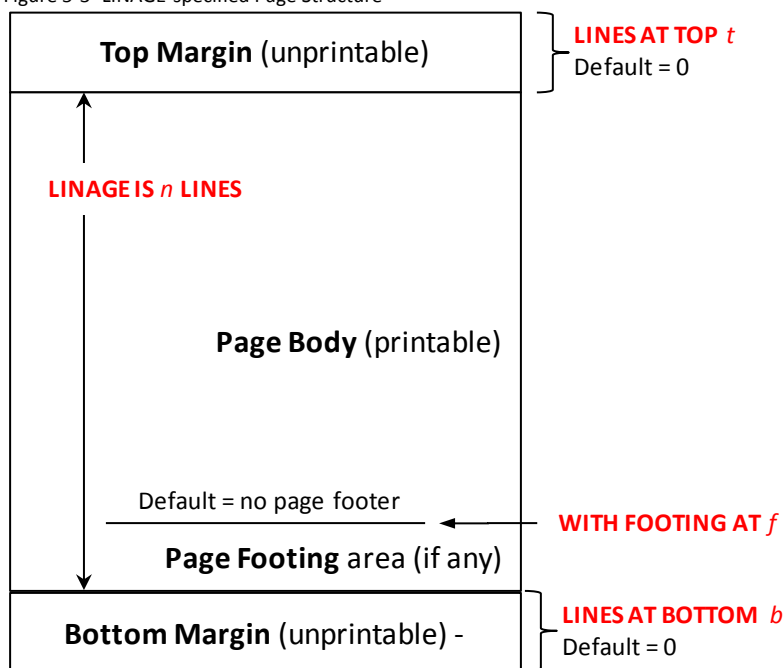
1. A file description for a file used as a sort/merge work file must be specified as an **SD**. The descriptions of all other files must be specified as **FDs**.
2. The name specified as *file-name-1* must exactly match the name specified on the file's **SELECT** statement.
3. By specifying the **EXTERNAL** clause, the file description is capable of being shared between all programs executed from the same execution thread, provided the file description is coded (with an **EXTERNAL** clause) in each program requiring it. This sharing allows the file to be **OPEN**ed, read and/or written and **CLOSE**d in different programs. This sharing applies to the record descriptions subordinate to the file description too.
4. By specifying the **GLOBAL** clause, the file description is capable of being shared between a program and any nested subprograms within it, provided the file description is coded (with a **GLOBAL** clause) in each program requiring it. This sharing allows the file to be **OPEN**ed, read and/or written and **CLOSE**d in different programs. Separately compiled programs cannot share a **GLOBAL** file description, but they can share an **EXTERNAL** file description. This sharing applies to the record descriptions subordinate to the file description too.

5. The **RECORD CONTAINS** and **RECORD IS VARYING** clauses are ignored (with a warning message issued) when used with **LINE SEQUENTIAL** files. With other file organizations these mutually-exclusive clauses define the length of data records within the file. The data item specified as *identifier-1* must be defined within one of the record descriptions of *file-name-1*.
6. The **CODE-SET**, clause allows a custom alphabet (defined in the **SPECIAL-NAMES** paragraph of the **CONFIGURATION SECTION**) to be associated with a file. This clause is valid only when used with **RECORD BINARY SEQUENTIAL** or **LINE SEQUENTIAL** files.
7. The **REPORT IS** clause is syntactically recognized but will cause an error since the Report Writer Control System (RWCS) is not currently supported by GNU COBOL.
8. The **BLOCK CONTAINS** clause is syntactically recognized by the GNU COBOL compiler, but is currently non-functional.
9. The **LABEL RECORD**, **DATA RECORD**, **RECORDING MODE** and **VALUE OF** clauses are obsolete. If used, they will have no impact on the generated code. The identifiers specified on the **DATA RECORD** clause will be verified as being defined within the program, but the compiler won't care whether they are actually specified as records of the file or not.
10. The **LINAGE** clause can only be specified for **ORGANIZATION RECORD BINARY SEQUENTIAL** or **ORGANIZATION LINE SEQUENTIAL** files. It cannot be used within an **SD**. If used on an **ORGANIZATION RECORD BINARY SEQUENTIAL** file, the definition of that file will be implicitly changed to **LINE SEQUENTIAL**.

11. The **LINAGE** clause is used to specify the logical boundaries (in terms of numbers of lines) of various areas on a printed page, as shown in [Figure 5-3](#).

This page structure – once defined - can be automatically enforced by the the **WRITE** statement.

Figure 5-3- LINAGE-specified Page Structure



12. The following special rules apply only to sort/merge work files (**SDs**):
 - a. Sort/merge work files should be assigned to **DISK** (or **DISC**).
 - b. **SORTs** and **MERGEs** will be performed in memory, if the amount of data being sorted allows.
 - c. Should actual disk work files be necessary due to the amount of data being **SORTed** or **MERGEed**, they will be automatically allocated to disk in a folder defined by the **TMPDIR**, **TMP** or **TEMP** environment variables. These disk files will be automatically purged upon **SORT** / **MERGE** termination. They will also be purged if the program terminates abnormally before the **SORT** or **MERGE** finishes. Should you ever need to know, temporary sort/merge work files will be named "cob*.tmp".
 - d. If you specify a specific filename in the sort/merge work file's **SELECT**, it will be ignored.

See Also...

The SPECIAL-NAMES Paragraph	4.1.4	The OPEN Statement	6.4.29
Defining File Characteristics (SELECT)	4.2.1	The SORT Statement (File Sort)	6.4.40.1
Describing Record Layouts	5.1.1	The WRITE Statement	6.4.50
The CLOSE Statement	6.4.7	Execution-time Environment Variables	8.2.4

The **MERGE** Statement [6.4.25](#)

5.1.1. Record Descriptions

Every file description must be followed by at least one record description. If there are multiple record descriptions present, the one with the longest length will define the size of the record buffer into which **READ** statements deliver data read from the file and from which **WRITE** statements take the data to be written to the file. The various record descriptions for a file description implicitly share that one common record buffer (thus, they provide different ways to view the structure of data that can exist within the file). Record buffers can be shared between files by using the **SAME RECORD AREA** clause within the **I-O-CONTROL** paragraph of the **ENVIRONMENT DIVISION**.

Record descriptions for all files take the form of 01-level data items that are coded immediately following the file description. These data items are constructed according to all the rules specified for defining non **SCREEN SECTION** data items, except that the **VALUE** clause may not be used.

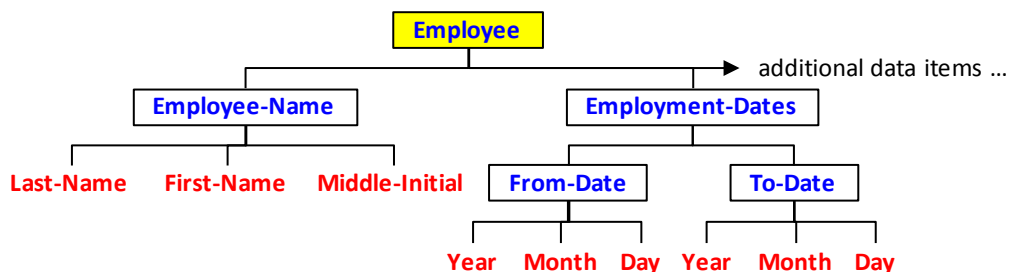
See Also...

Sharing Record Buffers Between Files [4.2.2](#)

Defining Records And Their Fields [5.2.1](#)

5.2. Describing Data Items

GNU COBOL data items, like those of other COBOL implementations, are described in a hierarchical manner. This accommodates the fact that data items frequently need to be able to be broken up into subordinate items. Take for example, the following logical layout of a portion of a data item named “Employee”:



The “Employee” data item consists of two subordinate data items – an “Employee-Name” and an “Employment-Dates” data item (presumably there would be a lot of others too, but we don’t care about them right now). As the diagram shows, each of those data items are – in turn – broken down into subordinate data items. This hierarchy of data items can get rather “deep”, and GNU COBOL has no problem dealing with it.

In GNU COBOL, data items that are broken down into other data items are referred to as **group items**, while those that aren’t broken down are called **elementary items**. A group item that doesn’t belong to any other data item (the one at the top of a chart like this one) is called a **record**. In the chart above, the names of all the elementary items are shown in **red** (without a box around it), the names of all the group items are shown in **blue** (with a box around it) and the record data item’s box is shaded yellow.

GNU COBOL uses the concept of a “level number” to indicate the level at which a data item occurs in a data structure such as the example shown above. Then these data items are defined, they are all defined together with a number in the range 1-49 specified in front of their names. Over the years, a convention has come to exist among COBOL programmers that level numbers are always coded as two-digit numbers – they don’t have to be specified as two-digit numbers, but every example you see in this document will take that approach!

The record data item (the one at the top) always has a level number of **01**. After that, you may assign level numbers as you wish (01 – 02 – 03 – 04 - ..., 01 – 05 – 10 – 15 - ..., etc.) as you see fit, as long as you follow these simple rules:

1. Every data item at the same “level” of a hierarchy diagram such as the one you see here (if you were to make one which you rarely – if ever – will once you get used to this concept) must have the same level number.
2. Every level uses a level number that is strictly greater than the one used in the prior (next higher) level.
3. You never use a level number greater than 49.

So, the definition of these data items in a GNU COBOL program would go something like this:

```
01 Employee
  05 Employee-Name
    10 Last-Name
    10 First-Name
    10 Middle-Initial
  05 Employment-Dates
    10 From-Date
      15 Year
      15 Month
      15 Day
    10 To-Date
      15 Year
      15 Month
      15 Day
```

The indentation is purely at the discretion of the programmer to make things easier for humans to read (the compiler couldn't care less). Historically, COBOL implementations that required Fixed Format Mode source programs required that the "01" level begin in Area A and that everything else begin in Area B. GNU COBOL only requires that all data definition syntax occur in columns 8-72. In Free Format Mode, of course, there aren't even those limitations.

The coding example shown above is incomplete – it only describes the data item names and their hierarchical relationships to one other. In addition, any valid data item definitions will also need to describe what type of data is to be contained in a data item (Numeric? Alphanumeric? Alphabetic?), how much data can "fit" and a multitude of other characteristics.

See Also...

Fixed-Format Source Code [1.5.1.1](#)

Defining Data Items [5.2](#)

5.2.1. Defining non-SCREEN SECTION Data Items

Figure 5-4 – Non-SCREEN SECTION Data Item Description Syntax

```

level-number [ { Identifier-1 } ] [ IS { EXTERNAL } ]
           [ FILLER ] [ { GLOBAL } ]
           [ ANY LENGTH ]
           [ BASED ]
           [ BLANK WHEN ZERO ]
           [ JUSTIFIED RIGHT ]
           [ OCCURS integer-1 [ TO integer-2 ] TIMES [ DEPENDING ON identifier-2 ]
           [ { ASCENDING } KEY IS identifier-3 ]
           [ { DESCENDING } ]
           [ INDEXED BY identifier-4 ]
           [ PICTURE picture-string ]
           [ REDEFINES identifier-2 ]
           [ RENAMES identifier-3 [ THRU | THROUGH identifier-4 ]
           [ SIGN IS { LEADING } [ SEPARATE CHARACTER ]
           [ { TRAILING } ]
           [ SYNCHRONIZED [ { LEFT } ]
           [ { RIGHT } ] ]
           [ USAGE IS data-item-usage ]
           [ VALUE IS [ ALL ] literal-1 ]
           .

```

The syntax skeleton shown here describes the manner in which data items are defined in all **DATA DIVISION** sections except the **SCREEN SECTION**.

1. The only valid level numbers are 01-49, 66, 77, 78 and 88. Level numbers 01 through 49 are used to define data items that may be part of a hierarchical structure of data items. Level number 01 can also be used to define a constant – an item with an unchangable value specified at compilation time. Level numbers 66, 77, 78 and 88 all have special uses, and are covered in upcoming sections (the “See Also” table at the end of this section provides links to those discussions).
2. Not specifying an *identifier-name-1* or **FILLER** immediately after the level number has the same effect as if **FILLER** were specified. A data item named **FILLER** cannot be referenced directly; these items are generally used to specify an unused portion of the total storage allocated to a group item.
3. By specifying the **EXTERNAL** clause, the data item is capable of being shared between all programs executed from the same execution thread, provided the data item is coded (with an **EXTERNAL** clause) in each program requiring it.
4. By specifying the **GLOBAL** clause, the data item is capable of being shared between a program and any nested subprograms within it, provided the data item is coded (with a **GLOBAL** clause) in each program requiring it.
5. The **EXTERNAL** clause may only be specified at the 77 or 01 level.
6. An **EXTERNAL** item must have a data name (i.e. *identifier-1*) and that name cannot be **FILLER**.
7. **EXTERNAL** cannot be combined with **GLOBAL**, **REDEFINES** or **BASED**.

- Every data item description must be terminated with a period.

See Also...

Describing Record Layouts 5.1.1	Defining Level-77 Data Items 5.2.5
Defining Screens 5.2.2	Defining Level 78 Constants 5.2.6
Defining Level-01 Constants 5.2.3	Defining Level-88 Condition Names 5.2.7
Defining Level-66 RENAMES Data Items 5.2.4	

5.2.1.1. ANY LENGTH Clause

- Data items declared with the **ANY LENGTH** attribute have no fixed compile-time length. Such items may only be defined in the **LINKAGE SECTION** of a subprogram as they may only serve as subroutine argument descriptions. **ANY LENGTH** items must have a **PICTURE** clause that specifies exactly one A, X or 9 symbol.
- The **ANY LENGTH** and **BASED** clauses cannot be used together in the same data item description.

ANY LENGTH

5.2.1.2. BASED Clause

- Data items declared with **BASED** are allocated no storage at compilation time. At run-time, the **ALLOCATE** or **SET ADDRESS** verbs are used to allocate space for and (optionally) initialize such items.
- The **BASED** and **ANY LENGTH** clauses cannot be used together in the same data item description.
- The **BASED** clause may only be used on level 01 and level 77 data items.

BASED*See Also...*

The ALLOCATE Statement 6.4.3	The SET ADDRESS Statement 6.4.39.3
-----------------------------------------------------	-----------------------------------------------------------

5.2.1.3. BLANK WHEN ZERO Clause

- The **BLANK WHEN ZERO** clause can only be used with a **PIC 9 USAGE DISPLAY** data item; it will cause that item's value to be automatically transformed into SPACES if a value of 0 is ever **MOVED** to the item.

BLANK WHEN ZERO

5.2.1.4. JUSTIFIED Clause

- The **JUSTIFIED RIGHT** clause, valid only on an alphabetic (**PIC A**) or alphanumeric (**PIC X**) data item, will cause values shorter than the length of the data item to be right-justified and space-filled when they are **MOVED** into the data item (the default behavior is to left-justify and space fill).
- The word **JUSTIFIED** may be abbreviated as **JUST**.

JUSTIFIED RIGHT

5.2.1.5. OCCURS Clause

- The **OCCURS** clause is used to create a data structure called a *table*¹¹ that repeats multiple times. For example:

```
OCCURS integer-1 [ TO integer-2 ] TIMES [ DEPENDING ON identifier-2 ]
      [ { ASCENDING } KEY IS identifier-3 ]
      [ INDEXED BY identifier-4 ]
```

05 QUARTLY-REVENUE OCCURS 4 TIMES PIC 9(7)V99.

Will allocate the following:

QUARTLY-REVENUE (1)	QUARTLY-REVENUE (2)	QUARTLY-REVENUE (3)	QUARTLY-REVENUE (4)
----------------------------	----------------------------	----------------------------	----------------------------

Each occurrence is referenced using the subscript syntax (a numeric literal, arithmetic expression or numeric identifier enclosed within parenthesis) shown in the diagram. The **OCCURS** clause may be used at the group level too, in which case the entire group structure repeats, as follows:

```
05 X OCCURS 3 TIMES.
  10 A PIC X(1).
  10 B PIC X(1).
  10 C PIC X(1).
```

	X (1)			X (2)			X (3)		
	A (1)	B (1)	C (1)	A (2)	B (2)	C (2)	A (3)	B (3)	C (3)

- The optional **DEPENDING ON** clause can be added to an **OCCURS** to create a variable-length table. Such tables will be allocated out to the maximum size specified as *integer-2*. At execution time the value of *identifier-2* will determine how many of the table elements are accessible.
- See the documentation of the **SEARCH**, **SEARCH ALL** and **SORT** verbs for explanations of the **KEY** and **INDEXED BY** clauses.
- The **OCCURS** clause cannot be specified in a data description entry that has a level number of 01, 66, 77, or 88.

5.2.1.6. PICTURE Clause

- The word **PICTURE** may be abbreviated as **PIC**.
- The **PICTURE** clause defines the class (numeric, alphabetic or alphanumeric) of the data that may be contained by the data item being defined. A **PICTURE** also (sometimes in conjunction with **USAGE**) defines the amount of storage reserved for the data item. The three basic class-specification **PICTURE** symbols have the following uses:

PICTURE picture-string

Figure 5-5 - Data Class-Specification PICTURE Symbols (A/X/9)

Basic Symbol	Meaning and Usage
9	Defines a spot reserved for a single decimal digit. The actual amount of storage occupied will depend on the specified USAGE .
A	Defines a place reserved for a single alphabetic character ("A"-“Z”, “a”-“z”). Each “A” represents a single byte of storage.
X	Defines a place reserved for a single character of storage. Each “X” represents a single byte of storage.

These three symbols are used repeatedly in a **PICTURE** clause to define how many of each class of data may be contained within the field. For example:

PIC 9999 Allocates a data item that can store four-digit positive numbers (we’ll see shortly how negative values can be accounted for). If the **USAGE** of the field is **DISPLAY** (the default), four bytes of storage will be allocated and each byte may contain the character “0”, “1”, “2”, ... , “8” or “9”. There is no run-time enforcement of the fact that only digits are allowed. A compilation-time

¹¹ Other programming languages with which you might be familiar refer to this sort of structure as an *array*.

WARNING will be issued if literal value that violates the digits-only rule is **MOVED** to the field. A run-time violation is detectable using a class condition test.

- PIC 9(4) Identical to the above – a repeat count enclosed within parenthesis can be used with any **PICTURE** symbols that allows repetition.
- PIC X(10) This data item can hold a string of any ten characters.
- PIC A(10) This data item can hold a string of any ten letters. There is no enforcement of the fact that only letters are allowed, but a violation is detectable via a class condition test.
- PIC AA9(3)A This is exactly the same as specifying X(6), but it documents the fact that values should be two letters followed by 3 digits followed by a single letter. There is no enforcement and no capability of detecting violations other than a “brute force” check by character position.

Data items containing “A” or “X” **PICTURE** symbols cannot be used in arithmetic calculations.

In addition to the above [Figure 5-6](#) shows the numeric option **PICTURE** symbols that may be used with “**PIC 9**” Data Items

Figure 5-6 - Numeric Option PICTURE Symbols (P/S/V)

Numeric Option Symbol	Meaning and Usage
P	<p>Defines an implied digit position that will be considered to be a 0 when the data item is referenced at run-time. This symbol is used to allow data items that will contain very large values to be allocated using less storage by assuming a certain number of trailing zeros (one per “P”) to exist at the end of values.</p> <p>All computations and other operations performed against such a data item will behave as if the zeros were actually there.</p> <p>When values are stored into such a field they will have the digit positions defined by the “P” symbols stripped from the values as they are stored.</p> <p>For example, let’s say you need to allocate a data item that contains however many millions of dollars of revenue your company has in gross revenues this year:</p> <p>01 Gross-Revenue PIC 9(9).</p> <p>In which case 9 bytes of storage will be reserved. The values 000000000 thru 999999999 will represent the gross-revenues. But, if only the <u>millions</u> are tracked (meaning the last six digits are always going to be 0), you could define the field as:</p> <p>01 Gross-revenue PIC 9(3)P(6).</p> <p>Whenever Gross-Revenue is referenced in the program, the actual value in storage will be treated as if each P symbol (6 of them, in this case) were a zero.</p> <p>If you wanted to store the value 128 million into that field, you would do so as if the “P”s were “9”s:</p> <p>MOVE 128000000 TO Gross-Revenue.</p>
S	<p>This symbol, which if used must be the very first symbol in the PICTURE value, indicates that negative values are possible for this data item. Without an “S”, any negative values stored into this data item via a MOVE or arithmetic statement will have the negative sign stripped from it (in effect becoming the absolute value).</p>
V	<p>This symbol is used to define where an implied decimal-point (if any) is located in a numeric item. Just as there may only be a single decimal point in a number so may there be no more than one “V” in a PICTURE. Implied decimal points occupy no space in storage – they just specify how values are used. For example, if the value “1234” is in storage in a field defined as PIC 999V9, that value would be treated as 123.4 in any statements that referenced it.</p>

3. GNU COBOL supports all standard COBOL PICTURE editing symbols, namely “\$”, comma, asterisk (*), decimal-point, CR, DB, + (plus), - (minus), “B”, “0” (zero) and “/”, as follows:

Figure 5-7 - Numeric Editing PICTURE Symbols

Editing Symbol	Meaning and Usage																					
- (minus)	<p>This symbol must be used either at the very beginning of a PICTURE or at the very end. If “-” is used, none of “+”, “CR” or “DB” may be used. It is used to edit numeric values.</p> <p>Multiple consecutive “-” symbols are allowed only at the very beginning of the field. This is called a <i>floating minus sign</i>.</p> <p>Each “-” symbol will count as one character position in the size of the data item.</p> <p>If only a single “-” symbol is specified, that symbol will be “replaced” by a “-” if the value moved to the field is negative, or a SPACE otherwise.</p> <p>If a floating minus sign is used, think of the editing process as if it worked like this:</p> <ol style="list-style-type: none"> Determine what the edited value would be if each “-” were actually a “9”. Locate the digit in the edited result that corresponds to the right-most “-” and scan the edited value back to the left from that point until you come to a “0” that has nothing but “0” characters to the left of it. Replace that “0” with a “-” if the value moved to the field is negative or a SPACE otherwise. Replace all remaining “0” characters to the left of that position by SPACES. <p>Some examples (the symbol ␣ denotes a space):</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr style="background-color: black; color: white;"> <th>If this value...</th> <th>...is moved to a field with this PICTURE...</th> <th>... this value in storage will result:</th> </tr> </thead> <tbody> <tr> <td>17</td> <td>-999</td> <td>␣017</td> </tr> <tr> <td>-17</td> <td>-999</td> <td>-017</td> </tr> <tr> <td>265</td> <td>----99</td> <td>␣␣␣265</td> </tr> <tr> <td>-265</td> <td>----99</td> <td>␣␣␣-265</td> </tr> <tr> <td>51</td> <td>999-</td> <td>051␣</td> </tr> <tr> <td>-51</td> <td>999-</td> <td>051-</td> </tr> </tbody> </table>	If this value...	...is moved to a field with this PICTURE...	... this value in storage will result:	17	-999	␣017	-17	-999	-017	265	----99	␣␣␣265	-265	----99	␣␣␣-265	51	999-	051␣	-51	999-	051-
If this value...	...is moved to a field with this PICTURE...	... this value in storage will result:																				
17	-999	␣017																				
-17	-999	-017																				
265	----99	␣␣␣265																				
-265	----99	␣␣␣-265																				
51	999-	051␣																				
-51	999-	051-																				
\$ ¹²	<p>This symbol must be only be used at the very beginning of a PICTURE except that a “+” or “-” may appear to the left of it. It is used to edit numeric values.</p> <p>Multiple consecutive “\$” symbols are allowed. This is called a <i>floating currency symbol</i>.</p> <p>Each “\$” symbol will count as one character position in the size of the data item.</p> <p>If only a single “\$” symbol is specified, that symbol will be inserted into the edited value at that position unless there are so many significant digits to the field value that the position occupied by the “\$” is needed to represent a leading non-zero digit. In such cases, the “\$” will be treated as a “9”.</p> <p>If a floating currency sign is used, think of the editing process as if it worked like this:</p> <ol style="list-style-type: none"> Determine what the edited value would be if each “\$” were actually a “9”. Locate the digit in the edited result that corresponds to the right-most “\$” and scan the edited value back to the left from that point until you come to a “0” that has nothing but “0” characters to the left of it. Replace that “0” with a “\$”. Replace all remaining “0” characters to the left of that position by SPACES. <p>Some examples (the symbol ␣ denotes a space):</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr style="background-color: black; color: white;"> <th>If this value...</th> <th>...is moved to a field with this PICTURE...</th> <th>... this value in storage will result:</th> </tr> </thead> <tbody> <tr> <td>17</td> <td>\$999</td> <td>\$017</td> </tr> <tr> <td>265</td> <td>\$\$\$\$99</td> <td>␣␣␣\$265</td> </tr> </tbody> </table>	If this value...	...is moved to a field with this PICTURE...	... this value in storage will result:	17	\$999	\$017	265	\$\$\$\$99	␣␣␣\$265												
If this value...	...is moved to a field with this PICTURE...	... this value in storage will result:																				
17	\$999	\$017																				
265	\$\$\$\$99	␣␣␣\$265																				

¹² The default currency sign used is “\$”. Other countries use different currency signs. The **SPECIAL-NAMES** paragraph allows any symbol to be defined as a currency symbol. If the currency sign is defined to the character ‘#’, for example, then you would use the ‘#’ character as a **PICTURE** editing symbol.

Editing Symbol	Meaning and Usage												
* (asterisk) ¹³	<p>This symbol must be only be used at the very beginning of a PICTURE except that a “+” or “-” may appear to the left of it. It is used to edit numeric values.</p> <p>Multiple consecutive “*” symbols are not only allowed, but are the typical usage. This is called a <i>floating check protection symbol</i>.</p> <p>Each “*” symbol will count as one character position in the size of the data item.</p> <p>Think of the editing process as if it worked like this:</p> <ol style="list-style-type: none"> 1. Determine what the edited value would be if each “*” were actually a “9”. 2. Locate the digit in the edited result that corresponds to the right-most “*” and scan the edited value back to the left from that point until you come to a “0” that has nothing but “0” characters to the left of it. 3. Replace that “0” with a “*”. 4. Replace all remaining “0” characters to the left of that position by “*” also. <p>An example:</p> <table border="1" data-bbox="376 707 1323 797"> <thead> <tr> <th>If this value...</th> <th>...is moved to a field with this PICTURE...</th> <th>... this value in storage will result:</th> </tr> </thead> <tbody> <tr> <td>265</td> <td>*****99</td> <td>****265</td> </tr> </tbody> </table>	If this value...	...is moved to a field with this PICTURE...	... this value in storage will result:	265	*****99	****265						
If this value...	...is moved to a field with this PICTURE...	... this value in storage will result:											
265	*****99	****265											
, (comma) ¹³	<p>Each comma (,) in the PICTURE string represents a character position into which the character “,” will be inserted. This character position is counted in the size of the item. The “,” symbol is a “smart symbol” capable of masquerading as the <u>floating</u> symbol to its left and right should there be insufficient digits of precision to the numeric value being edited to require the insertion of a “,” character.</p> <p>For example (the symbol $\text{\textcircled{h}}$ denotes a space):</p> <table border="1" data-bbox="376 969 1323 1120"> <thead> <tr> <th>If this value...</th> <th>...is moved to a field with this PICTURE...</th> <th>... this value in storage will result:</th> </tr> </thead> <tbody> <tr> <td>17</td> <td>\$\$,\$\$\$,\$99</td> <td>$\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}17$</td> </tr> <tr> <td>265</td> <td>\$\$,\$\$\$,\$99</td> <td>$\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}265$</td> </tr> <tr> <td>1456</td> <td>\$\$,\$\$\$,\$99</td> <td>$\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}1,456$</td> </tr> </tbody> </table>	If this value...	...is moved to a field with this PICTURE...	... this value in storage will result:	17	\$\$,\$\$\$,\$99	$\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}17$	265	\$\$,\$\$\$,\$99	$\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}265$	1456	\$\$,\$\$\$,\$99	$\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}1,456$
If this value...	...is moved to a field with this PICTURE...	... this value in storage will result:											
17	\$\$,\$\$\$,\$99	$\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}17$											
265	\$\$,\$\$\$,\$99	$\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}265$											
1456	\$\$,\$\$\$,\$99	$\text{\textcircled{h}}\text{\textcircled{h}}\text{\textcircled{h}}1,456$											
. (period) ¹³	<p>This symbol inserts a decimal point into the edited value at the point where an implied decimal point exists in the value. It is used to edit numeric values. Note that the period specified at the end of every data item definition IS NOT treated as an editing symbol!</p> <p>An example:</p> <pre data-bbox="376 1272 925 1400"> 01 Edited-Value PIC 9(3).99. 01 Payment PIC 9(3)V99 VALUE 152.19. ... MOVE Payment TO Edited-Value. DISPLAY Edited-Value. </pre> <p>Will display 152.19</p>												
/ (slash)	<p>This symbol – usually used when editing dates for printing – inserts a “/” character into the edited value. The inserted “/” character will occupy a byte of storage in the edited result.</p> <p>An example:</p> <pre data-bbox="376 1568 766 1668"> 01 Edited-Date PIC 99/99/9999. ... MOVE 08182009 TO Edited-Date. DISPLAY Edited-Date. </pre> <p>The displayed value will be 08/18/2009.</p>												

¹³ If **DECIMAL-POINT IS COMMA** is specified in the **SPECIAL-NAMES** paragraph, the meanings and usages of the “.” and “,” characters will be reversed

Editing Symbol	Meaning and Usage																					
<p>+ (plus)</p>	<p>This symbol must be used either at the very beginning of a PICTURE or at the very end. If “+” is used, none of “-”, “CR” or “DB” may be used. It is used to edit numeric values.</p> <p>Multiple consecutive “+” symbols are allowed only at the very beginning of the field. This is called a <i>floating plus sign</i>.</p> <p>Each “+” symbol will count as one character position in the size of the data item. If only a single “+” symbol is specified, that symbol will be replaced by a “-” if the value moved to the field is negative, or a “+” otherwise.</p> <p>If a floating plus sign is used, think of the editing process as if it worked like this:</p> <ol style="list-style-type: none"> 1. Determine what the edited value would be if each “+” were actually a “9”. 2. Locate the digit in the edited result that corresponds to the right-most “+” and scan the edited value back to the left from that point until you come to a “0” that has nothing but “0” characters to the left of it. 3. Replace that “0” with a “-” if the value moved to the field is negative or a “+” otherwise. 4. Replace all remaining “0” characters to the left of that position by SPACES. <p>Some examples (the symbol ␣ denotes a space):</p> <table border="1" data-bbox="373 752 1321 994"> <thead> <tr> <th>If this value...</th> <th>...is moved to a field with this PICTURE...</th> <th>... this value in storage will result:</th> </tr> </thead> <tbody> <tr> <td>17</td> <td>+999</td> <td>+017</td> </tr> <tr> <td>-17</td> <td>+999</td> <td>-017</td> </tr> <tr> <td>265</td> <td>++++99</td> <td>␣␣␣+265</td> </tr> <tr> <td>-265</td> <td>++++99</td> <td>␣␣␣- 265</td> </tr> <tr> <td>51</td> <td>999+</td> <td>051+</td> </tr> <tr> <td>-51</td> <td>999-</td> <td>051-</td> </tr> </tbody> </table>	If this value...	...is moved to a field with this PICTURE...	... this value in storage will result:	17	+999	+017	-17	+999	-017	265	++++99	␣␣␣+265	-265	++++99	␣␣␣- 265	51	999+	051+	-51	999-	051-
If this value...	...is moved to a field with this PICTURE...	... this value in storage will result:																				
17	+999	+017																				
-17	+999	-017																				
265	++++99	␣␣␣+265																				
-265	++++99	␣␣␣- 265																				
51	999+	051+																				
-51	999-	051-																				
<p>0 (zero)</p>	<p>This symbol inserts a “0” character into the edited value. The inserted “0” character will occupy a byte of storage in the edited result.</p> <p>An example:</p> <pre> 01 Edited-Phone-Number PIC 9(3)B9(3)B9(4). ... MOVE 5185551212 TO Edited-Phone-Number. DISPLAY Edited-Phone-Number. </pre> <p>The displayed value will be 518 555 1212.</p>																					
<p>B</p>	<p>This symbol inserts a SPACE character into the edited value. The inserted SPACE character will occupy a byte of storage in the edited result.</p> <p>An example:</p> <pre> 01 Edited-Phone-Number PIC 9(3)B9(3)B9(4). ... MOVE 5185551212 TO Edited-Phone-Number. DISPLAY Edited-Phone-Number. </pre> <p>The displayed value will be 518 555 1212.</p>																					
<p>CR</p>	<p>This symbol must be used only at the very end of a PICTURE. If “CR” is used, none of “-”, “+” or “DB” may be used. It is used to edit numeric values.</p> <p>Multiple “CR” symbols are not allowed in one PICTURE clause.</p> <p>A “CR” symbol will count as two character positions in the size of the data item.</p> <p>If the value moved into the field is negative, the characters “CR” will be inserted into the edited value, otherwise two SPACES will be inserted.</p> <p>Some examples (the symbol ␣ denotes a space):</p> <table border="1" data-bbox="373 1823 1321 1937"> <thead> <tr> <th>This value...</th> <th>...is moved to a field with this PICTURE...</th> <th>...resulting in this value in storage:</th> </tr> </thead> <tbody> <tr> <td>17</td> <td>99CR</td> <td>17␣␣</td> </tr> <tr> <td>-17</td> <td>99CR</td> <td>17CR</td> </tr> </tbody> </table>	This value...	...is moved to a field with this PICTURE...	...resulting in this value in storage:	17	99CR	17␣␣	-17	99CR	17CR												
This value...	...is moved to a field with this PICTURE...	...resulting in this value in storage:																				
17	99CR	17␣␣																				
-17	99CR	17CR																				

Editing Symbol	Meaning and Usage									
DB	<p>This symbol must be used only at the very end of a PICTURE. If “DB” is used, none of “-”, “+” or “CR” may be used. It is used to edit numeric values.</p> <p>Multiple “DB” symbols are not allowed in one PICTURE clause.</p> <p>A “DB” symbol will count as two character positions in the size of the data item.</p> <p>If the value moved into the field is negative, the characters “DB” will be inserted into the edited value, otherwise two SPACES will be inserted.</p> <p>Some examples (the symbol ␣ denotes a space):</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: black; color: white;">This value...</th> <th style="background-color: black; color: white;">...is moved to a field with this PICTURE...</th> <th style="background-color: black; color: white;">...resulting in this value in storage:</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">17</td> <td style="text-align: center;">99DB</td> <td style="text-align: center;">17␣␣</td> </tr> <tr> <td style="text-align: center;">-17</td> <td style="text-align: center;">99DB</td> <td style="text-align: center;">17DB</td> </tr> </tbody> </table>	This value...	...is moved to a field with this PICTURE...	...resulting in this value in storage:	17	99DB	17␣␣	-17	99DB	17DB
This value...	...is moved to a field with this PICTURE...	...resulting in this value in storage:								
17	99DB	17␣␣								
-17	99DB	17DB								
Z	<p>This symbol must be only be used at the very beginning of a PICTURE except that a “+” or “-” may appear to the left of it. It is used to edit numeric values.</p> <p>Multiple consecutive “Z” symbols are not only allowed, but are the typical manner in which this editing symbol is used. This is called a <i>floating zero suppression</i>.</p> <p>Each “Z” symbol will count as one character position in the size of the data item.</p> <p>Think of the editing process as if it worked like this:</p> <ol style="list-style-type: none"> 1. Determine what the edited value would be if each “Z” were actually a “9”. 2. Locate the digit in the edited result that corresponds to the right-most “Z” and scan the edited value back to the left from that point until you come to a “0” that has nothing but “0” characters to the left of it. 3. Replace that “0” with a SPACE. 4. Replace all remaining “0” characters to the left of that position by SPACES. <p>Some examples (the symbol ␣ denotes a space):</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: black; color: white;">This value...</th> <th style="background-color: black; color: white;">...is moved to a field with this PICTURE...</th> <th style="background-color: black; color: white;">...resulting in this value in storage:</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">17</td> <td style="text-align: center;">Z999</td> <td style="text-align: center;">␣017</td> </tr> <tr> <td style="text-align: center;">265</td> <td style="text-align: center;">ZZZZ99</td> <td style="text-align: center;">␣␣␣265</td> </tr> </tbody> </table>	This value...	...is moved to a field with this PICTURE...	...resulting in this value in storage:	17	Z999	␣017	265	ZZZZ99	␣␣␣265
This value...	...is moved to a field with this PICTURE...	...resulting in this value in storage:								
17	Z999	␣017								
265	ZZZZ99	␣␣␣265								

No more than one editing symbol may be used in a floating manner in the same **PICTURE** clause.

4. Numeric data items containing editing symbols are referred to as numeric edited fields. Such data items may receive values in the various arithmetic statements but may not be used as sources of data in those same statements. The statements in question are **ADD**, **COMPUTE**, **DIVIDE**, **MULTIPLY** and **SUBTRACT**.

See Also...

The SPECIAL-NAMES Paragraph 4.1.4	The COMPUTE Statement 6.4.9
Storage Format of Data (USAGE) 5.2.1.11	The DIVIDE Statement 6.4.13
Class Tests 6.1.4.2.2	The MULTIPLY Statement 6.4.27
The ADD Statement 6.4.2	The SUBTRACT Statement 6.4.44

5.2.1.7. REDEFINES Clause

1. The **REDEFINES** clause causes *identifier-1* (the data item in which the REDEFINES clause is specified) to occupy the same physical storage space as *identifier-2*, so that storage may be defined in a different manner with a (probably) different structure. The following must all be true in order to use **REDEFINES**:

REDEFINES *identifier-2*

 - a. The level number of *identifier-2* must be the same as that of *identifier-1*.
 - b. The level number of *identifier-2* (and *identifier-1*) cannot be 66, 78 or 88.
 - c. If “n” represents the level number of *identifier-2* (and *identifier-1*), then no other data items with level number “n” may be defined between *identifier-1* and *identifier-2*.
 - d. The total allocated size of *identifier-1* must be the same as the total allocated size of *identifier-2*.

- e. No **OCCURS** clause may be defined on *identifier-2*. There may – however – be items defined with **OCCURS** clauses subordinate to *identifier-2*.
- f. No **VALUE** clause may be defined on *identifier-2*. No data items subordinate to *identifier-2* may have **VALUE** clauses, with the exception of level-88 condition names.

5.2.1.8. RENAMES Clause

The **RENAMES** clause regroups previously defined items by specifying alternative, possibly overlapping, groupings of elementary data items in a record.

RENAMES *identifier-3* [**THRU** | **THROUGH** *identifier-4*]

See Also...

Defining Level-66 **RENAMES** Data Items [5.2.4](#)

5.2.1.9. SIGN Clause

1. The **SIGN** clause, allowable only for **USAGE DISPLAY** numeric data items, specifies how an “S” symbol will be interpreted in a data item’s **PICTURE** clause. Without the **SEPARATE CHARACTER** option, the sign of the data item’s value will be encoded by transforming the last (**TRAILING**) or first (**LEADING**) digit as follows:

SIGN IS { **LEADING** | **TRAILING** } [**SEPARATE CHARACTER**]

Figure 5-8 - Sign-Encoding Characters

First/Last Digit	Encoded Value For POSITIVE	Encoded Value For NEGATIVE
0	0	p
1	1	q
2	2	r
3	3	s
4	4	t
5	5	u
6	6	v
7	7	w
8	8	x
9	9	y

If the **SEPARATE CHARACTER** clause is used, then an actual “+” or “-” sign will be inserted into the field’s value as the first (**LEADING**) or last (**TRAILING**) character.

2. When **SEPARATE CHARACTER** is specified, the “S” symbol in the data item’s **PICTURE** must be counted when determining the data item’s size.

See Also...

Defining a Data Item’s **PICTURE** [5.2.1.6](#)

5.2.1.10. SYNCHRONIZED Clause

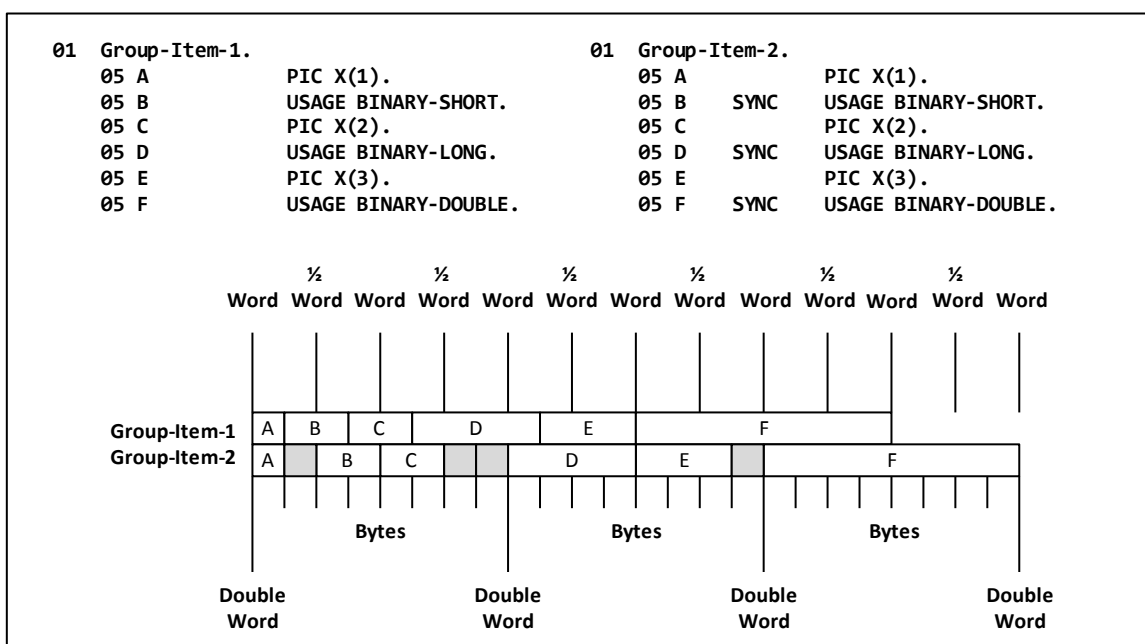
- The **SYNCHRONIZED** clause (which may be abbreviated as **SYNC**) optimizes the storage of binary numeric items to store them in such a manner as to make it as fast as possible for the CPU to fetch them. This synchronization is performed as follows:

SYNCHRONIZED [{ LEFT }] [{ RIGHT }]

- If the binary item occupies one byte of storage, no synchronization is performed.
- If the binary item occupies two bytes of storage, the binary item is allocated at the next half-word boundary.
- If the binary item occupies four bytes of storage, the binary item is allocated at the next word boundary.
- If the binary item occupies four bytes of storage, the binary item is allocated at the next word boundary.

Figure 5-9 provides an example of a group item’s storage allocation with and without using **SYNCHRONIZED**.

Figure 5-9 - Effect of the SYNCHRONIZED Clause



The grey blocks represent the unused “slack” bytes that are allocated in the **Group-Item-2** structure because of the **SYNC** clauses.

The **LEFT** and **RIGHT** options to the **SYNCHRONIZED** clause are recognized for syntactical compatibility with other COBOL implementations, but are otherwise non-functional.

5.2.1.11. USAGE Clause

- The following table summarizes the various possible **USAGE** specifications:

USAGE IS *data-item-usage*

Figure 5-10 - Summary of USAGE Specifications

USAGE	Range of Possible Values	Format (See note #2,#4)	Allows Negative Values? (See note #3)	Used w/ PICTURE?
BINARY	Defined by the quantity of “9”s in the PICTURE and the presence or absence of an “S” in the PICTURE	Compatible Binary Integer	If PICTURE contains “S”	Yes
BINARY-C-LONG [SIGNED]	Same as BINARY-DOUBLE SIGNED			
BINARY-C-LONG UNSIGNED	Typically 0 to 4,294,967,295	Native Binary Integer	No – see #3	No

USAGE	Range of Possible Values	Format (See note #2,#4)	Allows Negative Values? (See note #3)	Used w/ PICTURE?
FLOAT-LONG ¹⁸	Approximately -1.797693134862316×10 ³⁰⁸ to 1.797693134862316×10 ³⁰⁸	Native IEEE 754 Binary64 ¹⁸ Floating- point	Yes	No
FLOAT-SHORT ¹⁸	Approximately -3.4028235×10 ³⁸ to 3.4028235×10 ³⁸	Native IEEE 754 Binary32 ¹⁸	Yes	No
INDEX	0 to maximum address possible (32 or 64 bits)	Native Binary Integer	No	No
NATIONAL	USAGE NATIONAL , while syntactically recognized, is not supported by GNU COBOL			
PACKED-DECIMAL	Defined by the quantity of “9”s in the PICTURE and the presence or absence of an “S” in the PICTURE (see #1)	Signed Packed Decimal	If PICTURE contains “S”	No
POINTER	0 to maximum address possible (32 or 64 bits)	Native Binary Integer	No	No
PROGRAM-POINTER	0 to maximum address possible (32 or 64 bits)	Native Binary Integer	No	No
SIGNED-INT	Same as BINARY-LONG SIGNED			
SIGNED-LONG	Same as BINARY-DOUBLE SIGNED			
SIGNED-SHORT	Same as BINARY-SHORT SIGNED			
UNSIGNED-INT	Same as BINARY-LONG UNSIGNED			
UNSIGNED-LONG	Same as BINARY-DOUBLE UNSIGNED			
UNSIGNED-SHORT	Same as BINARY-SHORT UNSIGNED			

- Binary data (integer or floating-point) can be stored in either a “Big-Endian” or “Little-Endian” form.

Big-endian data allocation calls for the bytes that comprise a binary item to be allocated such that the least-significant byte is the right-most byte. For example, a four-byte binary item having a value of decimal 20 would be big-endian allocated as 00000014 (shown in hexadecimal notation).

Little-endian data allocation calls for the bytes that comprise a binary item to be allocated such that the least-significant byte is the left-most byte. For example, a four-byte binary item having a value of decimal 20 would be little-endian allocated as 14000000 (shown in hexadecimal notation).

All CPUs are capable of “understanding” big-endian format, which makes it the “most-compatible” form of binary storage across computer systems.

Some CPUs – such as the Intel/AMD i386/x64 architecture processors such as those used in most Windows PCs – prefer to process binary data stored in a little-endian format. Since that format is more efficient on those systems, it is referred to as the “native” binary format.

On a system supporting only one format of binary storage (generally, that would be big-endian), the terms “most-efficient” format and “native format” are synonymous.

- Data items that have the **UNSIGNED** attribute explicitly coded, or **DISPLAY/PACKED-DECIMAL/COMP-5/COMP-X** items that do not have an “S” symbol in their **PICTURE** clause cannot preserve negative values that may be stored into them. Storing a negative value into such a field will actually result in the sign being stripped, essentially saving the absolute value in the data item.
- Packed-decimal (i.e. **USAGE PACKED-DECIMAL, COMP-3** or **COMP-6**) data is stored as a series of bytes such that each byte contains two 4-bit fields, referred to as “nibbles” (since they comprise half a “byte”) with each nibble representing a “9” in the **PICTURE** and each holding a single decimal digit encoded as its binary value (0 = 0000, 1 = 0001, ..., 9 = 1001). The last byte of a **PACKED-DECIMAL** or **COMP-3** data item will always have its left nibble corresponding to the last “9” in the **PICTURE** and its right nibble reserved as a sign indicator. This sign indicator is always present regardless of whether or not the **PICTURE** included an “S” symbol. The first byte of the data item will contain an unused left nibble if the **PICTURE** had an even number of “9” symbols in it. The sign indicator will have a value of a hexadecimal A thru F. Traditional packed decimal encoding rules

¹⁸ The **USAGE** specifications **FLOAT-LONG** and **FLOAT-SHORT** use the IEEE 754 “Binary64” and “Binary32: formats, respectively. These are binary encodings of real decimal numbers, and as such cannot represent every possible value between the minimum and maximum values in the range for those **USAGES**. Wikipedia has an excellent article on the Binary64 and Binary32 encoding schemes – just search on “Binary32” or “Binary64”. GNU COBOL stores **FLOAT-LONG** and **FLOAT-SHORT** data items using Native byte ordering techniques (see #2).

call for hexadecimal values of C, A, F and E in the sign nibble to indicate a positive value and B or D to represent a negative value (hexadecimal digits 0-9 are undefined). Testing with a Windows MinGW/GNU COBOL implementation shows that – in fact – hex digit D represents a negative number and any other hexadecimal digit denoting a positive number. Therefore, a **PIC S9(3) COMP-3** packed-decimal field with a value of -15 would be stored internally as a hexadecimal 015D in GNU COBOL. If you attempt to store a negative number into a packed decimal field that has no “S” in its **PICTURE**, the absolute value of the negative number will actually be stored. A **USAGE of COMP-6** does not allow for negative values, therefore no sign nibble will be allocated. A **USAGE COMP-6** data item containing an odd number of “9” symbols in its **PICTURE** will leave its leftmost nibble unused.

5. A **USAGE** clause specified at the group item level will apply that **USAGE** to all subordinate data items, except those that themselves have a **USAGE** clause.

See Also...

GNU COBOL “config” Files [8.1.6](#)

5.2.1.12. VALUE Clause

1. The **VALUE** clause is ignored on **EXTERNAL** data items or on any data items defines as subordinate to an **EXTERNAL** data item.
2. The **VALUE** clause may not be used anywhere in the description of an 01 item serving as an FD or SD record description.
3. **VALUE** specifies an initial compilation-time value that will be assigned to the storage occupied by the data item in the program object code generated by the compiler. If the optional “**ALL**” clause is used, it may only be used with an alphanumeric literal value; the value will be repeated as needed to completely fill the data item. Here are some examples with and without **ALL**:

```
PIC X(5) VALUE "A"           *> will have the value "A",SPACE,SPACE,SPACE,SPACE
PIC X(5) VALUE ALL "A"      *> will have the value "A","A","A","A","A"
PIC 9(3) VALUE 1            *> will have the value 001
PIC 9(3) VALUE ALL "1"     *> will have the value 111
```

4. Giving a table an initial, compile-time value is one of the trickier aspects of COBOL data definition. There are basically three standard techniques and a fourth that people familiar with other COBOL implementations but new to GNU COBOL may find interesting. So, here are the three “standard” approaches:
 - a. Don’t bother worrying about it at compile-time. Use the **INITIALIZE** statement to initialize all data item occurrences in a table (at run-time) to their data-type-specific default values (numerics: 0, alphabetic and alphanumeric: **SPACES**).
 - b. Initialize small tables at compile time by including a **VALUE** clause on the group item that serves as a “parent” to the table, as follows:

```
05 SHIRT-SIZES                VALUE "S 14M 15L 16XL17".
  10 SHIRT-SIZE-TBL           OCCURS 4 TIMES.
    15 SST-SIZE               PIC X(2).
    15 SST-NECK               PIC 9(2).
```

- c. Initialize tables of almost any size at compilation time by utilizing the **REDEFINES** clause:

```
05 SHIRT-SIZE-VALUES.
  10 PIC X(4)                 VALUE "S 14".
  10 PIC X(4)                 VALUE "M 15".
  10 PIC X(4)                 VALUE "L 16".
  10 PIC X(4)                 VALUE "XL17".
05 SHIRT-SIZES                REDEFINES SHIRT-SIZE-VALUES.
  10 SHIRT-SIZE-TBL           OCCURS 4 TIMES.
    15 SST-SIZE               PIC X(2).
    15 SST-NECK               PIC 9(2).
```

Admittedly, the table shown in #3c is much more verbose than #3b. What is good about #3c, however, is that you can have as many **FILLER/VALUE** items as you need for a larger table (and those values can be as long as necessary!

Many COBOL compilers do not allow the use of **VALUE** and **OCCURS** on the same data item; additionally, they don’t allow a **VALUE** clause on a data item subordinate to an **OCCURS**. GNU COBOL, however, has neither of these restrictions!

Observe the following example, which illustrates the fourth manner in which tables may be initialized in GNU COBOL:

```
05 X          OCCURS 6 TIMES.  
 10 A        PIC X(1) VALUE '?'.  
 10 B        PIC X(1) VALUE '%'.  
 10 N        PIC 9(2) VALUE 10.
```

In this example, all six “A” items will be initialized to “?”, all six “B” items will be initialized to “%” and all six “N” items will be initialized to 10. It’s not clear exactly how many times this sort of initialization will be useful, but it’s there if you need it.

See Also...

The **INITIALIZE** Statement [6.2.22](#)

5.2.2. Defining SCREEN SECTION Data Items

Figure 5-11 - SCREEN SECTION Data Item Description Syntax

```

level-number [ identifier-1 | FILLER ]
[ AUTO | AUTO-SKIP | AUTOTERMINATE ]
[ BELL | BEEP ]
[ BACKGROUND-COLOR IS integer-1 | identifier-2 ]
[ BLANK LINE | SCREEN ]
[ BLANK WHEN ZERO ]
[ BLINK ]
[ COLUMN NUMBER IS [ PLUS | + ] integer-2 | identifier-3 ]
[ ERASE EOL | EOS ]
[ FOREGROUND-COLOR IS integer-3 | identifier-4 ]
[ FROM literal-1 | identifier-5
  [ TO identifier-6
    [ USING identifier-7 ] ] ]
[ FULL | LENGTH-CHECK ]
[ HIGHLIGHT | LOWLIGHT ]
[ JUSTIFIED RIGHT ]
[ LEFTLINE ]
[ LINE NUMBER IS [ PLUS | + ] integer-4 | identifier-8 ]
[ OCCURS integer-5 TIMES ]
[ OVERLINE ]
[ PICTURE picture-string ]
[ PROMPT [ CHARACTER IS literal-2 | identifier-9 ] ]
[ REQUIRED | EMPTY-CHECK ]
[ REVERSE-VIDEO ]
[ SECURE | NO-ECHO ]
[ SIGN IS LEADING | TRAILING [ SEPARATE CHARACTER ] ]
[ UNDERLINE ]
[ VALUE IS [ ALL ] literal-3 ]

```

The syntax skeleton shown here describes how data items are defined in the **SCREEN SECTION**.

These data items are used via special forms of the **ACCEPT** and **DISPLAY** verbs to create full-screen TUI (“Textual User Interface”) programs.

1. Data items defined in the **SCREEN SECTION** describe input, output or combination screen layouts to be used with **DISPLAY** or **ACCEPT** statements. These screen layouts may define the entire available screen area or any subset of it.
2. The term “available screen area” is a nebulous one in those environments where command-line shell sessions are invoked within a graphical user-interface environment (as will be the case on Windows, OSX and most Unix/Linux systems) – these environments allow command-line session windows to exist with a variable number of available screen rows and columns. When you are designing GNU COBOL screens, you need to do so with an awareness of the logical row/column geometry the program will be executing within.
3. Data items with level numbers 01 (Constants), 66, 78 and 88 may be used in the **SCREEN SECTION**; they have the same syntax, rules and usage as they do in the other **DATA DIVISION** sections.
4. Without **LINE** or **COLUMN** clauses, **SCREEN SECTION** fields will display on the console window beginning at whatever line/column coordinate is stated or implied by the **ACCEPT** or **DISPLAY** statement that presents the

screen item. After a field is presented to the console window, the next field will be presented immediately following that field.

- A **LINE** clause explicitly stated in the definition of a **SCREEN SECTION** data item will override any **LINE** clause included on the **ACCEPT** or **DISPLAY** statement that presents that data item to the screen. The same is true of **COLUMN** clauses.
- The Tab and Back-Tab (Shift-Tab) keys will position the cursor from field to field in the line/column sequence in which the fields occur on the screen at execution time, regardless of the sequence in which they were defined in the **SCREEN SECTION**.

See Also...

Defining Level-01 Constants 5.2.3	Defining Level-88 Condition Names 5.2.7
Defining Level-66 RENAMES Data Items 5.2.4	The ACCEPT Statement (Screen Data) 6.4.1.4
Defining Level 78 Constants 5.2.6	The DISPLAY Statement (Screen Data) 6.4.12.4

5.2.2.1. AUTO | AUTO-SKIP | AUTOTERMINATE Clause

- The **AUTO** clause (the three forms are all equivalent) will cause the cursor to automatically advance to the next input-enabled field if the field having the **AUTO** clause is completely filled.

[**AUTO** | **AUTO-SKIP** | **AUTOTERMINATE**]

5.2.2.2. BACKGROUND-COLOR Clause

- The **BACKGROUND-COLOR** clause is used to specify the screen background color of the screen data item or the default screen background color of subordinate items if **BACKGROUND-COLOR** is used on a group item. You specify colors by number (0-7), or by using the constant names provided in the "screenio.cpy" copybook (which is provided with all GNU COBOL source distributions).
- BACKGROUND-COLOR** values are inheritable from previous fields - they are not inherited from the prior field encountered but rather from parent data items (data items with numerically lower level numbers).
- The following is the GNU COBOL color palette:

[**BACKGROUND-COLOR IS** { *integer-1* } { *identifier-2* }]

Figure 5-12 - The GNU COBOL Color Palette (Windows Console)

Color Integer Value	"screenio.cpy" Constant Name	Normal or LOWLIGHT Appearance	HIGHLIGHT Appearance
0	COB-COLOR-BLACK		
1	COB-COLOR-BLUE		
2	COB-COLOR-GREEN		
3	COB-COLOR-CYAN		
4	COB-COLOR-RED		
5	COB-COLOR-MAGENTA		
6	COB-COLOR-YELLOW		
7	COB-COLOR-WHITE		

5.2.2.3. BEEP | BELL Clause

- Use the **BELL** or **BEEP** clauses (they are synonymous) to cause an audible tone to occur when the screen item is **DISPLAYED** ().

[**BELL** | **BEEP**]

5.2.2.4. BLANK LINE and BLANK SCREEN Clauses

- The **BLANK SCREEN** clause will blank-out the entire screen prior to displaying the new screen contents described by the screen data item whose description this clause is part of.
- The **BLANK LINE** clause will blank out the entire screen line upon which the screen

[**BLANK** { **LINE** } { **SCREEN** }]

data item whose description contains this clause prior to displaying this screen data item.

3. Blanked-out areas will have their foreground and background colors set to the attributes of the field containing the **BLANK** clause.
4. This clause is useful when one **SCREEN SECTION** item is being **DISPLAYed** over the top of a previously-**DISPLAYed** one.

5.2.2.5. BLANK WHEN ZERO Clause

1. The **BLANK WHEN ZERO** will cause that screen data item's value to be automatically transformed into SPACES if a value of 0 is ever put into the field via a FROM, USING or VALUE clause.

BLANK WHEN ZERO

5.2.2.6. BLINK Clause

1. The **BLINK** clause modifies the visual appearance of the displayed field by making the field contents blink. The manner in which the blinking is accomplished will vary, depending upon the "curses" package built into the GNU COBOL implementation you're using, as well as the visual presentation capabilities of the command window shell you're using. The Windows console, for example, does not support blinking, so the visual effect of **BLINK** in a native Windows or MinGW version of GNU COBOL is to elevate the **BACKGROUND-COLOR** intensity (normally low) to high intensity.
2. See [Figure 5-12](#) for the GNU COBOL color palette. The "HIGHLIGHT" column shows the effect the **BLINK** clause will have on **BACKGROUND-COLOR** when running within a Windows console window.

[**BLINK**]

5.2.2.7. COLUMN Clause

1. The **COLUMN** clause provides a means of explicitly stating in which column a field should be presented on the console window (it's line location will be determined by the **LINE** clause).
2. You may abbreviate **COLUMN** as **COL**.
3. The value of *integer-2* must be 1 or greater.
4. If *identifier-3* is used to specify either an absolute or relative column position, *identifier-3* must be defined as a **PIC 9** item without editing symbols. The value of *identifier-3* at the time the screen data item is presented must be 1 or greater.
5. Any numeric **USAGE** is allowed for *identifier-3* except for **COMPUTATIONAL-1** or **COMPUTATIONAL-2**. Note that either of these floating-point **USAGE** specifications will be accepted, but will produce unpredictable results.
6. Coordinates may be stated on an absolute basis (i.e. "**COLUMN 5**") or on a relative basis based upon the end of the previously-presented field (i.e. "**COLUMN PLUS 1**").
7. The symbol "+" may be used in lieu of the word **PLUS**, if desired; if "+" is used in combination with *integer-2*, however, there must be at least one space separating it from *integer-2*. Failure to include this space will cause the "+" sign to be simply treated as part of *integer-2* and will treat the **COLUMN** clause as an absolute column specification rather than a relative one.
8. If a screen data items description includes the **FROM**, **TO**, **USING** or **VALUE** clause but has no **COLUMN** clause, "**COLUMN PLUS 1**" will be assumed.

[**COLUMN NUMBER IS** [**PLUS** | +] { *integer-2* } *identifier-3*]

5.2.2.8. ERASE EOL and ERASE EOS Clauses

1. The **ERASE EOS** clause will blank-out screen contents from the location where the screen data item whose description contains this clause will be displayed, forward until the end of the screen prior to displaying this screen data item.
2. The **ERASE EOL** clause will blank-out screen contents from the location where the

[**ERASE** { **EOL** } **EOS**]

screen data item whose description contains this clause will be displayed, forward until the end of that screen line prior to displaying this screen data item.

- Erased- areas will have their foreground and background colors set to the attributes of the field containing the **ERASE** clause.
- This clause is useful when one **SCREEN SECTION** item is being **DISPLAYed** over the top of a previously-**DISPLAYed** one.

5.2.2.9. FOREGROUND-COLOR Clause

- The **FOREGROUND-COLOR** clause is used to specify the text color of the screen data item or the default text color of subordinate items if **FOREGROUND-COLOR** is used on a group item. You specify colors by number (0-7), or by using the constant names provided in the "screenio.cpy" copybook (which is provided with all GNU COBOL source distributions).

```
[ FOREGROUND-COLOR IS { integer-3 } ]
```

- FOREGROUND-COLOR** values are inheritable from previous fields - they are not inherited from the prior field encountered but rather from parent data items (data items with numerically lower level numbers).
- See [Figure 5-12](#).for the GNU COBOL color palette.

5.2.2.10. FROM, TO and USING Clauses

- The **FROM** clause is used to define a field whose contents should come from the specified literal or identifier.
- The **TO** clause is used to define a data-entry field with no initial value; when a value is entered, it will be saved to the specified identifier.
- The **USING** clause is a combination of "**FROM** *identifier-6*" and "**TO** *identifier-6*".

```
[ [ FROM { literal-1 } ] ]
[ [ TO ] ]
[ [ USING { identifier-6 } ] ]
```

5.2.2.11. FULL | LENGTH-CHECK Clause

- The **FULL** or **LENGTH-CHECK** clause forces the user to enter data into the field it is specified on (or into all subordinate input-capable fields if specified on a group item) sufficient to fill every character position of the field. In order to take effect, the user must move the cursor into the field having the **FULL/LENGTH-CHECK** clause in its definition. The **ACCEPT** statement will ignore the Enter key and any other cursor-moving keystrokes that would cause the cursor to move to another screen item unless the proper amount of data has been entered into the field. Function keys will still be allowed to terminate the **ACCEPT**, however. In order to be functional, this attribute must be supported by the underlying "curses" package your GNU COBOL package was built with. As of this time, the PDCurses package (used for native Windows or MinGW builds) does not support **FULL/LENGTH-CHECK**.

```
[ FULL | LENGTH-CHECK ]
```

See Also...

The ACCEPT Statement (Screen Data) [6.4.1.4](#)

5.2.2.12. HIGHLIGHT and LOWLIGHT Clauses

- The **HIGHLIGHT** and **LOWLIGHT** clauses control the intensity of text (**FOREGROUND-COLOR**). This is intended to provide a three-level intensity scheme (**LOWLIGHT** ... nothing (Normal) ... **HIGHLIGHT**). In environments such as a Windows console where only two levels of intensity are supported, **LOWLIGHT** is the same as leaving this clause off altogether.
- See [Figure 5-12](#).for the GNU COBOL color palette and the effect the HIGHTLIGHT clause has on it in 2-level intensity environments such as Windows.

```
[ { HIGHLIGHT } ]
[ { LOWLIGHT } ]
```

5.2.2.13. JUSTIFIED Clause

1. The **JUSTIFIED RIGHT** clause, valid only on an alphabetic (**PIC A**) or alphanumeric (**PIC X**) data item, will cause values shorter than the length of the data item to be right-justified and space-filled when they are transferred into the screen data item via the **FROM** or **USING** clause (the default behavior is to left-justify and space fill).
2. The word **JUSTIFIED** may be abbreviated as **JUST**.

JUSTIFIED RIGHT

5.2.2.14. LEFTLINE, OVERLINE and UNDERLINE Clauses

1. The **LEFTLINE**, **OVERLINE** and **UNDERLINE** clauses will introduce a horizontal line at the left, top or bottom edge of a screen field, respectively.
2. These clauses may be used in any combination in a single field's description.
3. These clauses are essentially non-functional when used within Windows command shell (cmd.exe) environments; those video attributes are not currently supported by the Windows console window API.
4. Whether or not these clauses operate on Cygwin or UNIX/Linux systems will depend upon the video attribute capabilities of the terminal output drivers being used.

[**LEFTLINE**]
 [**OVERLINE**]
 [**UNDERLINE**]

5.2.2.15. LINE Clause

1. The **LINE** clause provides a means of explicitly stating on which line a field should be presented on the console window (it's column location will be determined by the **COLUMN** clause).
2. The value of *integer-4* must be 1 or greater.
3. If *identifier-7* is used to specify either an absolute or relative column position, *identifier-7* must be defined as a **PIC 9** item without editing symbols. The value of *identifier-7* at the time the screen data item is presented must be 1 or greater.
4. Any numeric USAGE is allowed for *identifier-7* except for **COMPUTATIONAL-1** or **COMPUTATIONAL-2**. Note that either of these floating-point **USAGE** specifications will be accepted, but will produce unpredictable results.
5. Coordinates may be stated on an absolute basis (i.e. "**COLUMN 5**") or on a relative basis based upon the end of the previously-presented field (i.e. "**COLUMN PLUS 1**").
6. The symbol "+" may be used in lieu of the word **PLUS**, if desired; if "+" is used in combination with *integer-4*, however, there must be at least one space separating it from *integer-4*. Failure to include this space will cause the "+" sign to be simply treated as part of *integer-4* and will treat the **LINE** clause as an absolute line specification rather than a relative one.
7. If a screen data items description includes the **FROM**, **TO**, **USING** or **VALUE** clause but has no **LINE** clause, the "current screen line" will be assumed.

[**LINE NUMBER IS** [**PLUS** | +] { *integer-4* } *identifier-7*]

5.2.2.16. OCCURS Clause

1. An **OCCURS** clause can be used to repeat screen field definitions. It may be used on either elementary or group data items.
2. If an *identifier-1* was included in the description of the data item containing the **OCCURS** clause, references to *identifier-1* will need to be subscripted.

[**OCCURS** *integer-1* **TIMES**]

5.2.2.17. PICTURE Clause

1. The **PICTURE** clause specifies the type (A=Alphabetic, 9=Numeric, X=Alphanumeric) and size of a screen field.
2. If the screen data item whose description contains the **PICTURE** clause is an input field (meaning its definition includes either the **TO** or **USING** clause), the type specified by the **PICTURE** (A or 9) will be enforced on the user. For example, if the **PICTURE** is 9, only numeric characters (digits, decimal point, sign) will be accepted. If the **PICTURE** is A, only letters and spaces will be accepted.

[**PICTURE** *picture-string*]

3. If a screen data item does not have a **PICTURE** clause, its size will be inferred from the literal or identifier associated with the field via a **FROM**, **TO** or **USING** clause. If there is no such clause, then length will be inferred from the **VALUE** clause. If there is no **VALUE** clause, the screen data item will be treated as a group item (if data items that follow have a higher level number) or an elementary item of length 0 (if data items that follow have a smaller or equal level number).

5.2.2.18. PROMPT Clause

1. This clause defines the character that will be used as the fill-character for any input fields on the screen.
2. The default character, should no **CHARACTER** specification be coded, or should the **PROMPT** clause be absent altogether, is an underscore (“_”).
3. **PROMPT** characters will be automatically transformed into SPACES upon input.

```
[ PROMPT [ CHARACTER IS { literal-2
                               identifier-8 } ] ]
```

5.2.2.19. REQUIRED | EMPTY-CHECK Clause

1. The **REQUIRED** or **EMPTY-CHECK** clauses force the user to enter data into the field it is specified on (or into all subordinate input-capable fields if **REQUIRED/EMPTY-CHECK** is specified on a group item). In order to take effect, the user must move the cursor into the field having the **REQUIRED/EMPTY-CHECK** clause in its definition. The **ACCEPT** statement will ignore the Enter key and any other cursor-moving keystrokes that would cause the cursor to move to another screen item unless data has been entered into the field. Function keys will still be allowed to terminate the **ACCEPT**, however. In order to be functional, this attribute must be supported by the underlying “curses” package your GNU COBOL package was built with. As of this time, the PDCurses package (used for native Windows or MinGW builds) does not support **REQUIRED/EMPTY-CHECK**.

```
[ REQUIRED | EMPTY-CHECK ]
```

See Also...

The **ACCEPT** Statement (Screen Data) [6.4.1.4](#)

5.2.2.20. REVERSE-VIDEO Clause

1. The **REVERSE-VIDEO** attribute reverses the meaning of the specified or implied **BACKGROUND-COLOR** and **BACKGROUND-COLOR** attributes for the field (or all subordinate fields if used on a group item).

```
[ REVERSE-VIDEO ]
```

5.2.2.21. SECURE | NO-ECHO Clause

1. The **SECURE** or **NO-ECHO** clause (they are synonymous with each other) may only be used on a field allowing data entry (**USING** or **TO**). This attribute will cause all data entered into the field to appear as asterisks.

```
[ SECURE | NO-ECHO ]
```

5.2.2.22. SIGN Clause

1. The **SIGN** clause specifies how an “S” symbol (see section) within a **PICTURE** clause will be interpreted. Without the **SEPARATE CHARACTER** option, the sign of the screen data item’s value will be encoded by transforming the last (**TRAILING**) or first (**LEADING**) digit.

```
[ SIGN IS { LEADING
              TRAILING } [ SEPARATE CHARACTER ] ]
```

If the **SEPARATE CHARACTER** clause is used, then an actual “+” or “-” sign will be inserted into the field’s value as the first (**LEADING**) or last (**TRAILING**) character.

2. When **SEPARATE CHARACTER** is specified, the “S” symbol in the data item’s **PICTURE** must be counted when determining the data item’s size.

See Also...

Defining Signed Data Items (**SIGN**) [5.2.1.9](#)

5.2.2.23. VALUE Clause

1. The **VALUE** clause specifies an alphanumeric literal that will appear on the screen at the explicit or implicit line/column position of the screen data item.
2. A figurative constant may NOT be supplied as *literal-2*.
3. The inclusion of a **VALUE** clause into a screen data item's description overrides any **FROM**, **TO** or **USING** clause that may be present.
4. If there is no **PICTURE** clause supplied, the size of the screen data item will be the length of the *literal-2* value. If there is no **PICTURE** clause and the **ALL** option is specified, the **ALL** option will be ignored.
5. If there is a **PICTURE** clause specified along with the **VALUE** clause, then the **ALL** option, if any, will fill the field (up to the size specified by the **PICTURE**) with repeated instances of *literal-2* (including a possible trailing partial instance).

[**VALUE IS** [**ALL**] *literal-2*]

5.2.3. 01-Level Constant Descriptions

Figure 5-13 - 01-Level Constant Description Syntax

```
01 constant-name-1 CONSTANT [ IS GLOBAL ]
```

```

{
  AS {
    { BYTE-LENGTH }
    { LENGTH }
    literal-1
  } OF {
    { identifier-1 }
    { usage-name }
  }
} .
FROM compilation-variable-name-1

```

The 01-level constant is one of four types of compilation-time constants that can be declared within a program. The other three types are CDF >>**DEFINE** constants, CDF >>**SET** constants and 78-level constants.

This particular type of constant declaration provides the ability to determine the length of a data item or the storage size associated with a particular numeric **USAGE** type – something not possible with the other types of constants.

1. The optional **IS GLOBAL** clause will make the constant's value available to any nested subprograms.
2. Constants defined in this way become undefined once an **END PROGRAM** or **END FUNCTION** directive is encountered in the input source.
3. Data descriptions of this form do not actually allocate any storage – they merely define a name (*constant-name-1*) that may be used anywhere a numeric literal (**BYTE-LENGTH** or **LENGTH** options) or a literal of the same type as *literal-1* may be used.
4. The *constant-name-1* name may not be referenced on a CDF statement.
5. Care must be taken that *constant-name-1* does not duplicate any other data item name that has been defined in the program as references to that data item name will refer to the constant and not the data item. The GNU COBOL compiler will not issue a warning about this condition.
6. The value specified for *usage-name-1* may be any of the **USAGES** that do not use a **PICTURE** clause.
7. The **BYTE-LENGTH** clause will produce a numeric value for *constant-name-1* identical to that which would be returned by the **BYTE-LENGTH** intrinsic function executed against *identifier-1* or a data item declared with a **USAGE** of *usage-name*.
8. The **LENGTH** clause will produce a numeric value for *constant-name-1* identical to that which would be returned by the **LENGTH** intrinsic function executed against *identifier-1* or a data item declared with a **USAGE** of *usage-name*.
9. If used, *usage-name* may be any of **BINARY-C-LONG**, **BINARY-CHAR**, **BINARY-DOUBLE**, **BINARY-LONG**, **BINARY-SHORT**, **COMP-1** (or **COMPUTATIONAL-1**), **COMP-2** (or **COMPUTATIONAL-2**), **FLOAT-DECIMAL-16**, **FLOAT-DECIMAL-34**, **FLOAT-LONG**, **FLOAT-SHORT**, **POINTER**, or **PROGRAM-POINTER**.

Here is the listing of a GNU COBOL program that uses 01-level constants to DISPLAY the length (in bytes) of the various PICTURE-less USAGE types.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. USAGELengths.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Len-BINARY-C-LONG    CONSTANT AS LENGTH OF BINARY-C-LONG.
01 Len-BINARY-CHAR     CONSTANT AS LENGTH OF BINARY-CHAR.
01 Len-BINARY-DOUBLE   CONSTANT AS LENGTH OF BINARY-DOUBLE.
01 Len-BINARY-LONG     CONSTANT AS LENGTH OF BINARY-LONG.
01 Len-BINARY-SHORT   CONSTANT AS LENGTH OF BINARY-SHORT.
01 Len-COMP-1          CONSTANT AS LENGTH OF COMP-1.
01 Len-COMP-2          CONSTANT AS LENGTH OF COMP-2.
01 Len-FLOAT-DECIMAL-16 CONSTANT AS LENGTH OF FLOAT-DECIMAL-16.
01 Len-FLOAT-DECIMAL-34 CONSTANT AS LENGTH OF FLOAT-DECIMAL-34.
01 Len-FLOAT-LONG      CONSTANT AS LENGTH OF FLOAT-LONG.
01 Len-FLOAT-SHORT    CONSTANT AS LENGTH OF FLOAT-SHORT.
01 Len-POINTER         CONSTANT AS LENGTH OF POINTER.
01 Len-PROGRAM-POINTER CONSTANT AS LENGTH OF PROGRAM-POINTER.
PROCEDURE DIVISION.
000-Main.
    DISPLAY "On this system, with this build of GNU COBOL, the"
    DISPLAY "PICTURE-less USAGES have these lengths (in bytes):"
    DISPLAY " "
    DISPLAY "BINARY-C-LONG:    " Len-BINARY-C-LONG
    DISPLAY "BINARY-CHAR:     " Len-BINARY-CHAR
    DISPLAY "BINARY-DOUBLE:    " Len-BINARY-DOUBLE
    DISPLAY "BINARY-LONG:      " Len-BINARY-LONG
    DISPLAY "BINARY-SHORT:    " Len-BINARY-SHORT
    DISPLAY "COMP-1:          " Len-COMP-1
    DISPLAY "COMP-2:          " Len-COMP-2
    DISPLAY "FLOAT-DECIMAL-16: " Len-FLOAT-DECIMAL-16
    DISPLAY "FLOAT-DECIMAL-34: " Len-FLOAT-DECIMAL-34
    DISPLAY "FLOAT-LONG:      " Len-FLOAT-LONG
    DISPLAY "FLOAT-SHORT:    " Len-FLOAT-SHORT
    DISPLAY "POINTER:        " Len-POINTER
    DISPLAY "PROGRAM-POINTER: " Len-PROGRAM-POINTER
    STOP RUN

```

The output of this program, on my Windows 7 system with a 32-bit MinGW build of GNU COBOL is:

```

On this system, with this build of GNU COBOL, the
PICTURE-less USAGES have these lengths (in bytes):

```

```

BINARY-C-LONG:    4
BINARY-CHAR:     1
BINARY-DOUBLE:   8
BINARY-LONG:     4
BINARY-SHORT:    2
COMP-1:          4
COMP-2:          8
FLOAT-DECIMAL-16: 8
FLOAT-DECIMAL-34: 16
FLOAT-LONG:      8
FLOAT-SHORT:    4
POINTER:         4
PROGRAM-POINTER: 4

```

See Also...

Nested Subprograms 7.6	Defining Level 78 Constants 5.2.6
The CDF >>DEFINE Statement 2.2.1	The BYTE-LENGTH Intrinsic Function 6.1.14.6
The CDF >>SET Statement 2.2.3	The LENGTH Intrinsic Function 6.1.14.31

Storage Format of Data (**USAGE**) [5.2.1.11](#)

5.2.4. 66-Level Data Descriptions (RENAMES)

Figure 5-14 - 66-Level Data Description Syntax

```
66 identifier-1 RENAMES identifier-2 [ THRU identifier-3 ] .
```

A 66-level data item regroups previously defined items by specifying alternative, possibly overlapping, groupings of elementary data items.

1. You must use the level number 66 for data description entries that contain the **RENAMES** clause.
2. A level-66 data item cannot rename a level-66, level-01, level-77, or level-88 data item.
3. The *identifier-2* and *identifier-3* data items, along with all data items defined between those two data items in the program source, must all be contained within the same 01-level record description.
4. There may be multiple level-66 data items that rename data items contained within the same 01-level record description.
5. All **RENAMES** entries associated with one logical record must immediately follow that record's last data description entry.

5.2.5. 77-Level Data Descriptions

1. A 77-level data item is one described using the syntax covered in section where all of the following are true:
 - a. The *level-number* used is 77.
 - b. The data item is described in the **WORKING-STORAGE**, **LOCAL-STORAGE** or **LINKAGE SECTION**.
 - c. The data item is not named **FILLER**.
 - d. The data item is an elementary item.
 - e. The data item is not part of any group item.
 - f. The data item description does not contain the **OCCURS** or **RENAMES** clause.

See Also...

Defining Data Items [5.2](#)

5.2.6. 78-Level Constant Descriptions

Figure 5-15 - 78-Level Constant Description Syntax

```
78 identifier-1 VALUE IS literal-1 .
```

The 78-level constant is one of four types of compilation-time constants that can be declared within a program. The other three types are CDF >>**DEFINE** constants, CDF >>**SET** constants and 01-level constants.

1. Constants defined in this way become undefined once an **END PROGRAM** or **END FUNCTION** directive is encountered in the input source.

See Also...

The CDF >>**DEFINE** Statement [2.2.1](#)

The CDF >>**SET** Statement [2.2.3](#)

Defining Level-01 Constants [5.2.3](#)

5.2.7. 88-Level Condition Names

Figure 5-16 - 88-Level Condition Name Syntas

```

88 condition-name-1 { VALUE IS
                       VALUES ARE } { literal-1 [ THRU literal-2 ] ... } ...
      [ WHEN SET TO FALSE IS literal-3 ]

```

Condition names are Boolean (i.e. "TRUE" / "FALSE") data items that receive their TRUE and FALSE values based upon the values of other data items.

1. Condition names are always defined subordinate to another data item. That data item must be an elementary item.
2. Condition names do not occupy any storage.
3. The VALUE(s) specified for the condition name specify the specific values and/or ranges of values of the parent elementary data item that will cause the condition name to have a value of TRUE.
4. The optional **FALSE** clause defines an explicit value that will be assigned to the parent elementary data item should the **SET** statement ever be used to set the condition-name-1 to FALSE.

See Also...

Condition Names [6.1.4.2.1](#)

The **SET *condition-name*** Statement [6.4.39.6](#)

6. PROCEDURE DIVISION

The **PROCEDURE DIVISION** of any GNU COBOL program marks the point where all executable code is written.

6.1. General PROCEDURE DIVISION Components

6.1.1. General Format of the PROCEDURE DIVISION

Figure 6-1 - General PROCEDURE DIVISION Syntax

```

PROCEDURE DIVISION
  [ USING argument-1 ... ]
  [ RETURNING identifier-1 ] .

  [ DECLARATIVES
    [ declaratives-procedure ] ...
  ]
  END DECLARATIVES.

  [ [ section-name-1 SECTION. ]
    [ [ paragraph-name-1. ]
      [ procedural-sentence-1 ] ... ] ... ] ...

```

It is in the **PROCEDURE DIVISION** that all executable program code will be placed.

1. The **USING** clause defines arguments that may be passed to a GNU COBOL program serving as a subprogram. All identifiers specified on the **USING** clauses must be defined in the **LINKAGE SECTION**.
2. The **RETURNING** clause can be used as a means of specifying and documenting a value that a subprogram can pass back to the program that invoked it. Main programs that wish to “pass back” a return code value to the operating system when they exit do so simply by **MOVE**ing a value to the **RETURN-CODE** special register, and do not need (or use) a **RETURNING** clause on their **PROCEDURE DIVISION** header.
3. The first (optional) segment of any **PROCEDURE DIVISION** is a special area known as “**DECLARATIVES**”. In this area, you may define processing routines that are to be used as special “trap” routines executed only when certain events occur.
4. The various sections and paragraphs in which the procedural logic of your program will be coded will follow any “**DECLARATIVES**”. These sections and paragraphs are discussed in more detail in section 0.

See Also...

Special Registers 6.1.13	The MOVE Statement 6.2.26
Subprogram Argument Definitions 6.1.2	Sub-programming 7
Using DECLARATIVES 6.1.4	

6.1.2. General Format for Subprogram Arguments

Figure 6-2 - Syntax of a PROCEDURE DIVISION USING Argument

```

BY { REFERENCE
      VALUE
    } [ UNSIGNED ] [ SIZE IS { AUTO
                                DEFAULT
                                integer-1
                              } ] [ OPTIONAL ] identifier-1

```

1. The **BY REFERENCE** clause indicates that the program will be passed the address of the data item corresponding to a program argument; any changes this program makes to a **BY REFERENCE** argument will be passed back to the calling program.
2. **BY REFERENCE** is the assumed default for the first **USING** argument should no **BY** clause be specified for it. Subsequent arguments will assume the “**BY**” specification of the argument prior to them should they lack a **BY** clause of their own.

3. The **BY VALUE** clause indicates the program will be passed a copy of the data item from the calling program that corresponds to the argument. The contents of **BY VALUE** arguments can be changed by the subprograms receiving them, but those changes will not “find their way” back to the calling program.
4. If the calling program passes an argument **BY REFERENCE** or **BY CONTENT**, the subprogram should specify that argument as “**BY REFERENCE**” on its **PROCEDURE DIVISION** header. If the calling program passes an argument **BY VALUE**, the subprogram should specify that argument as “**BY VALUE**” on its **PROCEDURE DIVISION** header.
5. The various **SIZE** clauses specify the size (in bytes) of received **BY VALUE** arguments. The **SIZE IS AUTO** clause (the default) indicates that argument size will be determined automatically based upon the size of the item in the calling program. The remaining **SIZE** options allow you to force a specific size to be assumed.
6. The **UNSIGNED** clause will add “unsigned” to the C-language code generated when defining the argument in the function header of the C function corresponding to the GNU COBOL subprogramming. This is of value when a C program will be calling this subprogram.

See Also...

The **CALL** Statement [6.4.5](#)

Sub-programming 7

6.1.3. PROCEDURE DIVISION Sections and Paragraphs

The **PROCEDURE DIVISION** is the only one of the COBOL divisions that allows you to create your own sections and paragraphs. These are collectively referred to as *procedure names*. Procedure names are optional in the **PROCEDURE DIVISION** and – when used – are named entirely according to the needs and whims of the programmer.

When procedure names are defined, the entire collection of GNU COBOL statements that follow the procedure name are collectively referred to as a *procedure*. If there are no procedure names defined whatsoever, then the entire set of all statements defined within the **PROCEDURE DIVISION** constitute a single (unnamed) procedure.

Procedure names may be up to thirty one (31) characters long, and may consist of letters, numbers, dashes and underscores, with just one caveat. A procedure name may neither begin nor end with a dash (-) or underscore (_) character. This means that “17” is a perfectly valid procedure name.

There are two circumstances under which the use of certain GNU COBOL statements or options will require the specification of procedures. These situations are:

1. When **DECLARATIVES** are specified. These are discussed in section [6.1.4](#) (“General Format for DECLARATIVES Procedures”).
2. When any **PROCEDURE DIVISION** statement that references procedures is used. These statements are:
 - **ALTER**
 - **GO TO**
 - **MERGE** (with an **OUTPUT PROCEDURE**)
 - **PERFORM**
 - **SORT** (with an **INPUT PROCEDURE** and/or an **OUTPUT PROCEDURE**)

See Also...

User-defined Names [1.10](#)

The **ALTER** Statement [6.2.4](#)

The **GO TO** Statement [6.2.20](#)

The **MERGE** Statement [6.4.25](#)

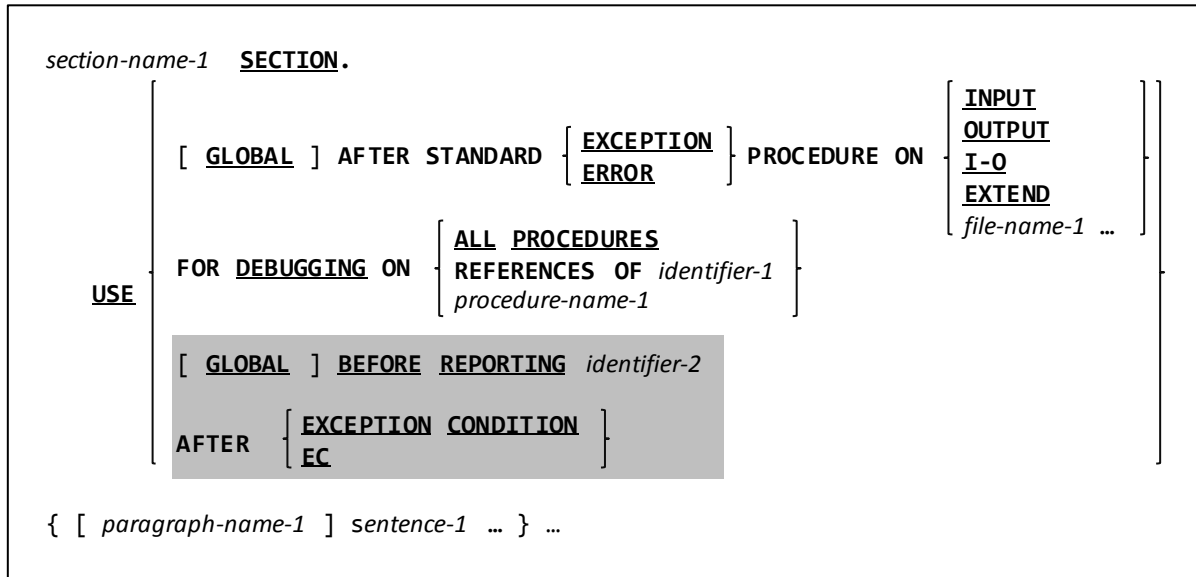
The **PERFORM** Statement (Procedural) [6.2.30.1](#)

The **SORT** Statement (File Sort) [6.4.40.1](#)

USE Statements and **DECLARATIVES** [6.1.4](#)

6.1.4. General Format for DECLARATIVES Procedures

Figure 6-3 - General DECLARATIVES Procedure Syntax



1. The **DECLARATIVES** area of the **PROCEDURE DIVISION** allows the programmer to define a series of “trap” procedures (referred to as *declarative procedures*) capable of intercepting certain events that may occur at program execution time. The syntax diagram above shows the format of a single such procedure.
2. **DECLARATIVES** may contain any number of declarative procedures, but no two declarative procedures should be designed to trap the same event.
3. The **USE BEFORE REPORTING** and **AFTER EXCEPTION CONDITION** clauses are currently syntactically recognized but otherwise unsupported.
4. The **USE FOR DEBUGGING** clause allows you to define a declarative procedure that will be invoked immediately before:
 - a. The specified identifier is referenced (**REFERENCES OF ...**), or ...
 - b. The named procedure is executed (*procedure-name-1*), or ...
 - c. Any procedure is executed (**ALL PROCEDURES**).

Any **USE FOR DEBUGGING** declarative procedures will be ignored at compilation time unless **WITH DEBUGGING MODE** is specified in the **SOURCE-COMPUTER** paragraph.

Any **USE FOR DEBUGGING** declarative procedures will be ignored at execution time unless the “**COB_SET_DEBUG**” environment variable has been set to a value of “Y”, “y” or “1”.

The typical use of a **USE FOR DEBUGGING** declarative procedure is to **DISPLAY** the **DEBUG-ITEM** special register, which will be implicitly and automatically created in your program for you if **WITH DEBUGGING MODE** is active.

5. The structure of **DEBUG-ITEM** will be as follows:

01	DEBUG-ITEM.		
05	DEBUG-LINE	PIC X(6).	The program line number of the statement that triggered the declaratives procedure.
05	FILLER	PIC X(1) VALUE SPACE.	
05	DEBUG-NAME	PIC X(31).	The procedure name or identifier name that triggered the declaratives procedure.
05	FILLER	PIC X(1) VALUE SPACE.	
05	DEBUG-SUB-1	PIC S9(4) SIGN LEADING SEPARATE.	The first subscript value (if any) for the reference of the identifier that triggered the declaratives procedure.
05	FILLER	PIC X(1) VALUE SPACE.	
05	DEBUG-SUB-2	PIC S9(4) SIGN LEADING SEPARATE.	The second subscript value (if any) for the reference of the identifier that triggered the declaratives procedure.
05	FILLER	PIC X(1) VALUE SPACE.	
05	DEBUG-SUB-3	PIC S9(4) SIGN LEADING SEPARATE.	The third subscript value (if any) for the reference of the identifier that triggered the declaratives procedure.
05	FILLER	PIC X(1) VALUE SPACE.	

05 DEBUG-CONTENTS PIC X(31).

A (brief) statement of the manner in which the procedure that triggered the declaratives procedure was executed or the first 31 characters of the value of the identifier whose reference triggered the declaratives procedure (the value after the statement was executed).

6. The **USE AFTER STANDARD ERROR PROCEDURE** clause defines a declarative procedure invoked any time a failure is encountered with the specified I/O type (or against the specified file(s)).
7. The **GLOBAL** option, if used, allows a declarative procedure to be used across all programs in the same compilation group.
8. Declarative procedures outlines (of any type) may not reference any other procedures defined outside the scope of **DECLARATIVES**.

See Also...

The SOURCE-COMPUTER Paragraph 4.1.1 Special Registers 6.1.13	Using DECLARATIVES 6.1.4 Execution-time Environment Variables 8.2.4
--------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------

6.1.5. Table References

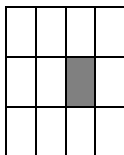
COBOL uses parenthesis to specify the subscripts used to reference table entries (tables in COBOL are what other programming languages refer to as arrays).

For example, observe the following data structure which simulates a 4 column by 3 row grid of characters:

```

01 GRID.
  05 GRID-ROW OCCURS 3 TIMES.
    10 GRID-COLUMN OCCURS 4 TIMES.
      15 GRID-CHARACTER          PIC X(1).
    
```

A reference to the GRID-CHARACTER shaded in the following diagram:



Would be coded as:

```
GRID-CHARACTER (2, 3)
```

Subscripts may be specified as numeric (integer) literals, **PIC 9** (integer) data items, data items created with any of the **PICTURE**-less integer **USAGE** specifications, **USAGE INDEX** data items or arithmetic expressions resulting in an integer value. The ability to use full arithmetic expressions as table (array) subscripts, while common in many languages, is rare in the COBOL universe, only having come into existence with the COBOL2002 standard.

See Also...

Arithmetic Expressions 6.1.4.1	Table Subscript versus Table Index 9.3
------------------------------------------------	--------------------------------------------------------

6.1.6. Qualification of Data Names

COBOL allows data names to be duplicated within a program, provided references to those data names may be made in such a manner as to make those references unique through a process known as *qualification*.

To see qualification at work, observe the following segments of two data records defined in a COBOL program:

```

01 EMPLOYEE.
  05 MAILING-ADDRESS.
    10 STREET          PIC X(35).
    10 CITY            PIC X(15).
    10 STATE          PIC X(2).
    10 ZIP-CODE.
      15 ZIP-CODE-5   PIC 9(5).
      15 FILLER       PIC X(4).
01 CUSTOMER.
  05 MAILING-ADDRESS.
    10 STREET          PIC X(35).
    10 CITY            PIC X(15).
    
```

```

10 STATE                PIC X(2).
10 ZIP-CODE.
   15 ZIP-CODE-5        PIC 9(5).
   15 FILLER            PIC X(4).

```

Now, let's deal with the problem of setting the CITY portion of an EMPLOYEE's MAILING-ADDRESS to "Philadelphia". Clearly, the following cannot work because the compiler will be unable to determine which of the two CITY fields you are referring to:

```
MOVE "Philadelphia" TO CITY.
```

We could qualify the reference to CITY as follows, in an attempt to correct the problem:

```
MOVE "Philadelphia" TO CITY OF MAILING-ADDRESS.
```

Unfortunately that too is insufficient because it is still insufficient to identify specifically which CITY is being referenced. To truly identify which specific CITY you want, you'd have to code the following:

```
MOVE "Philadelphia" TO CITY OF MAILING-ADDRESS OF EMPLOYEE.
```

Now there can be no confusion as to which CITY is being changed. Fortunately, you don't need to be quite so specific; COBOL allows intermediate qualification levels to be omitted. This allows you to specify:

```
MOVE "Philadelphia" TO CITY OF EMPLOYEE.
```

If you need to qualify a reference to a table, do so as follows:

```
identifier-1 OF identifier-2 ( subscript ... )
```

The reserved word "IN" may be used in lieu of "OF".

6.1.7. Reference Modifiers

Figure 6-4 - Reference Modifier Syntax

```

[ identifier-1 [ OF|IN identifier-2 ] [( subscript ... )] ] ( start : [ length ] )
[ intrinsic-function-reference ]

```

The COBOL '85 standard introduced the concept of a *reference modifier* to facilitate references to only a portion of a data item; GNU COBOL fully supports reference modification.

The *start* value indicates the starting character position being referenced (character position values start with 1, not 0 as is the case in some programming languages) and *length* specifies how many characters are wanted. If no *length* is specified, a value equivalent to the remaining character positions from *start* to the end will be assumed. Both *start* and *length* may be specified as integer numeric literals, integer numeric data items or arithmetic expressions with an integer value. The default *length* is 1.

Here are a few examples:

- | | |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CUSTOMER-LAST-NAME (1:3) | references the first three characters of CUSTOMER-LAST-NAME. |
| CUSTOMER-LAST-NAME (4:) | references all character positions of CUSTOMER-LAST-NAME from the fourth onward. |
| FUNCTION CURRENT-DATE (5:2) | references the current month. |
| Hex-Digits (Nibble + 1:1) | Assuming that "Nibble" is a numeric data item with a value in the range 0-15, and Hex-Digits is a PIC X(16) item with a value of "0123456789ABCDEF", this converts that numeric value to a hexadecimal digit. |
| Hex-Digits (Nibble + 1:) | Does the same as the above – if you leave out the length, 1 is assumed; YOU STILL NEED THE "." CHARACTER THOUGH. |
| Array-Element (6) (7:5) | References 5 characters in the 6 th occurrence of Array-Element, starting at character position 7. |

Reference modification may be used anywhere an identifier is legal, including serving as the receiving field of statements like **MOVE**, **STRING** and **ACCEPT**, to name a few.

See Also...

The CURRENT-DATE Intrinsic Function [6.1.14.12](#)

6.1.8. Expressions

GNU COBOL supports two basic types of Expressions

- ▶ *Arithmetic expressions*, which calculate a numeric result
- ▶ *Conditional Expressions*, which calculate a TRUE or FALSE value

Unlike other programming languages, which allow arithmetic values such as 0 and -1 to represent FALSE and TRUE, respectively, GNU COBOL treats logical TRUE/FALSE values as something different from 0/-1.

6.1.8.1. Arithmetic Expressions

Arithmetic expressions are formed using following operators. In complex expressions composed of multiple operators, a precedence of operation applies whereby those operations having a higher precedence are computed first before operations with a lower precedence.

Precedence	Operation	Discussion
1 st (Highest)	Figure 6-5 – Unary “Minus” (-) Operator Syntax $\left[\begin{array}{l} \text{numeric-literal-1} \\ - \text{identifier-1} \\ (\text{ arithmetic-expression-1 }) \end{array} \right]$	The unary “minus” (-) operator returns the arithmetic negation of its single argument, effectively returning as its value the product of its argument and -1.
	Figure 6-6 – Unary “Plus” (+) Operator Syntax $\left[\begin{array}{l} \text{numeric-literal-1} \\ + \text{identifier-1} \\ (\text{ arithmetic-expression-1 }) \end{array} \right]$	The unary “plus” (+) operator returns the value of its single argument, effectively returning as its value the product of its argument and +1.
2 nd	Figure 6-7 - Exponentiation Operator (** or ^) Syntax $\left[\begin{array}{l} \text{numeric-literal-1} \\ \text{identifier-1} \\ (\text{ arithmetic-expression-1 }) \end{array} \right] ** \left[\begin{array}{l} \text{numeric-literal-2} \\ \text{identifier-2} \\ (\text{ arithmetic-expression-2 }) \end{array} \right]$	The value of the left-hand argument raised to the power indicated by the right-hand argument is computed. Non-integer powers are allowed. GNU COBOL allows the “^” symbol to be used in lieu of the “**” symbol.
3 rd	Figure 6-8 - Multiplication Operator (*) Syntax $\left[\begin{array}{l} \text{numeric-literal-1} \\ \text{identifier-1} \\ (\text{ arithmetic-expression-1 }) \end{array} \right] * \left[\begin{array}{l} \text{numeric-literal-2} \\ \text{identifier-2} \\ (\text{ arithmetic-expression-2 }) \end{array} \right]$	The product of the left-hand argument and the right-hand argument is computed.
	Figure 6-9 - Division Operator (/) Syntax $\left[\begin{array}{l} \text{numeric-literal-1} \\ \text{identifier-1} \\ (\text{ arithmetic-expression-1 }) \end{array} \right] / \left[\begin{array}{l} \text{numeric-literal-2} \\ \text{identifier-2} \\ (\text{ arithmetic-expression-2 }) \end{array} \right]$	The value of the left-hand argument divided by the right-hand argument is computed. If the right-hand argument has a value of zero, expression evaluation will be prematurely terminated before a value is generated. This may cause program failure at run-time.

Precedence	Operation	Discussion
4 th (Lowest)	Figure 6-10 - Addition Operator (+) Syntax $\left[\begin{array}{l} \text{numeric-literal-1} \\ \text{identifier-1} \\ (\text{ arithmetic-expression-1 }) \end{array} \right] + \left[\begin{array}{l} \text{numeric-literal-2} \\ \text{identifier-2} \\ (\text{ arithmetic-expression-2 }) \end{array} \right]$	The sum of the left-hand argument and the right-hand argument is computed.
	Figure 6-11 - Subtraction Operator (-) Syntax $\left[\begin{array}{l} \text{numeric-literal-1} \\ \text{identifier-1} \\ (\text{ arithmetic-expression-1 }) \end{array} \right] - \left[\begin{array}{l} \text{numeric-literal-2} \\ \text{identifier-2} \\ (\text{ arithmetic-expression-2 }) \end{array} \right]$	The value of the right-hand argument subtracted from the left-hand argument is computed.

The syntactical rules of GNU COBOL, allowing a dash (-) character in data item names, can lead to some ambiguity. Observe this sample GNU COBOL code:

```
01 C      PIC 9 VALUE 5.
01 D      PIC 9 VALUE 2.
01 C-D    PIC 9 VALUE 7.
01 I      PIC 9 VALUE 0.
```

```
...
COMPUTE I=C-D+1
DISPLAY I
```

What should be displayed by the **DISPLAY** statement? The number “4”, which is the result of subtracting the value of **D** (the value 2) from the value of **C** (the value 5) and then adding 1 or the number “8”, which is the value of adding 1 to the value of data item **C-D**?

The right answer is “8” – the value of data item **C-D** plus 1!

The GNU COBOL compiler actually went through the following decision-making logic when generating code for the **COMPUTE** Statement

1. Is there a data item named “**C-D**” defined? If so, use its value for “**C-D**”
2. If there is no “**C-D**” data item, then check if there are “**C**” and “**D**” data items. If not, the **COMPUTE** statement is in error. If there are, however, then code will be generated to subtract the value of “**D**” from “**C**” and add 1 to the result.

Had there been at least one space to the left and/or the right of the “-”, there would have been no ambiguity – the compiler would have been forced to use the individual “**C**” and “**D**” data items.

It’s considered good COBOL programming practice to always code at least one space to both the left and right of every arithmetic operator as well as the “=” sign on a **COMPUTE**.

Here are some examples of how the precedence of operations affects the results of arithmetic expressions (all examples use numeric literals, to simplify the discussion).

Expression	Result	Notes
3 * 4 + 1	13	* has precedence over +
4 * 2 ^ 3 - 10	22	2 ³ is 8 (^ has precedence over *), times 4 is 32, minus 10 is 22.
(4 * 2) ^ 3 - 10	502	Parenthesis provide for a recursive application of the arithmetic expression rules, effectively allowing you to alter the precedence of operations. 4 times 2 is 8 (the use of parenthesis “trumps” the exponentiation operator, so the multiplication happens first); 8 ^ 3 is 512, minus 10 is 502.
5 / 2.5 + 7 * 2 - 1.15	15.35	Integer and non-integer operands may be freely intermixed

Of course, arithmetic expression operands may be numeric data items (any **USAGE** except **DISPLAY**, **POINTER** or **PROGRAM POINTER**) as well as numeric literals.

6.1.8.2. Conditional Expressions

Conditional expressions are expressions which identify the conditions under which a program may make a decision about processing to be performed. As such, conditional expressions produce a value of TRUE or FALSE.

There are seven types of conditional expressions, as follows, in increasing order of complexity.

6.1.8.2.1. Condition Names (Level-88 Items)

These are the simplest of all conditions. Observe the following code:

```

05 SHIRT-SIZE          PIC 99V9.
   88 LILLIPUTIAN     VALUE 0 THRU 12.5
   88 XS              VALUE 13 THRU 13.5.
   88 S              VALUE 14, 14.5.
   88 M              VALUE 15, 15.5.
   88 L              VALUE 16, 16.5.
   88 XL             VALUE 17, 17.5.
   88 XXL            VALUE 18, 18.5.
   88 BROBDINGNAGIAN VALUE 19 THRU 99.9.

```

The condition names “LILLIPUTIAN”, “XS”, “S”, “M”, “L”, “XL”, “XXL” and “BROBDINGNAGIAN” will have TRUE or FALSE values based upon the values within their parent data item (SHIRT-SIZE). So, a program wanting to test whether or not the current SHIRT-SIZE value can be classified as “XL” could have that decision coded as a combined condition (the most complex type of conditional expression), as either:

```

IF SHIRT-SIZE = 17 OR SHIRT-SIZE = 17.5
  - or -
IF SHIRT-SIZE = 17 OR 17.5

```

Or it could utilize the condition name XL as follows:

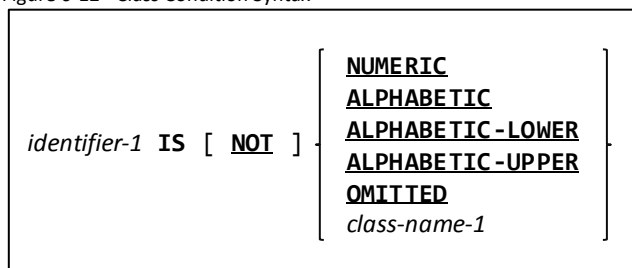
```
IF XL
```

See Also...

Defining Level-88 Condition Names [5.2.7](#)

6.1.8.2.2. Class Conditions

Figure 6-12 - Class Condition Syntax



Class conditions evaluate the type of data that is currently stored in a data item.

1. The **NUMERIC** class test considers only the characters “0”, “1”, ... , “9” to be numeric; only a data item containing nothing but digits will pass a **NUMERIC** class test. Spaces, decimal points, commas, currency signs, plus signs, minus signs and any other characters except the digit characters will all fail “**NUMERIC**” class tests.
2. The **ALPHABETIC** class test considers only upper-case letters, lower-case letters and **SPACES** to be alphabetic in nature.
3. The **ALPHABETIC-LOWER** and **ALPHABETIC-UPPER** class conditions consider only spaces and the respective type of letters to be acceptable in order to pass such a class test.
4. Note that what constitutes a “letter” (or upper/lower case too, for that manner) may be influenced through the use of **CHARACTER CLASSIFICATION** specifications in the **OBJECT-COMPUTER** paragraph.
5. Only data items whose **USAGE** is either explicitly or implicitly defined as **DISPLAY** may be used in **NUMERIC** or any of the **ALPHABETIC** class conditions.
6. Some COBOL implementations disallow the use of group items or **PIC A** items with **NUMERIC** class conditions and the use of **PIC 9** items with **ALPHABETIC** class conditions. GNU COBOL has no such restrictions.

- The **OMITTED** class condition is used when it is necessary for a subprogram to determine whether or not a particular argument was passed to it. In such class conditions, *identifier-1* must be a **LINKAGE SECTION** item defined on the **USING** clause of the subprograms **PROCEDURE DIVISION** header.
- The *class-name-1* option allows you to test for a user-defined class. Here's an example. First, assume the following **SPECIAL-NAMES** definition of the user-defined class "Hexadecimal":

```
SPECIAL-NAMES.
CLASS Hexadecimal IS '0' THRU '9', 'A' THRU 'F', 'a' THRU 'f'.
```

Now observe the following code, which will execute the **150-Process-Hex-Value** procedure if **Entered-Value** contains nothing but valid hexadecimal digits:

```
IF Entered-Value IS Hexadecimal
    PERFORM 150-Process-Hex-Value
END-IF
```

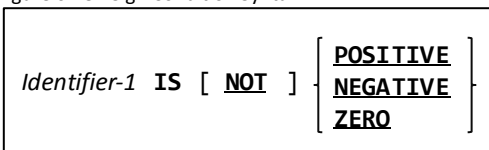
See Also...

The **OBJECT-COMPUTER** Paragraph [4.1.2](#)

The **CALL** Statement [6.4.5](#)

6.1.8.2.3. Sign Conditions

Figure 6-13 - Sign Condition Syntax

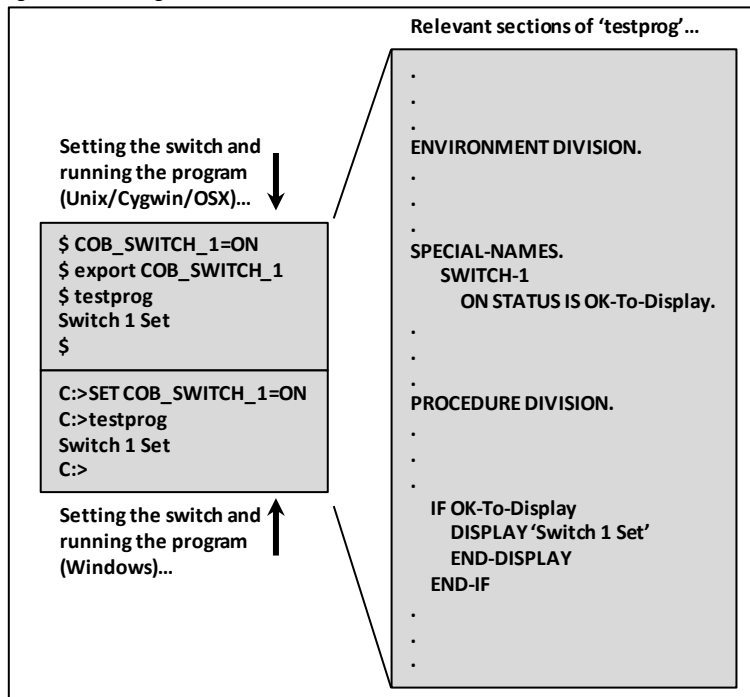


Sign conditions evaluate the numeric state of a **PIC 9** data item.

- Only data items defined with some sort of numeric **USAGE/PICTURE** can be used for this type of class condition.
- A **POSITIVE** or **NEGATIVE** class condition will be TRUE only if the value of *identifier-1* is strictly greater than or less than zero, respectively. A **ZERO** class condition can be passed only if the value of *identifier-1* is exactly zero.

6.1.8.2.4. Switch-Status Conditions

Figure 6-14 - Using Switch Conditions



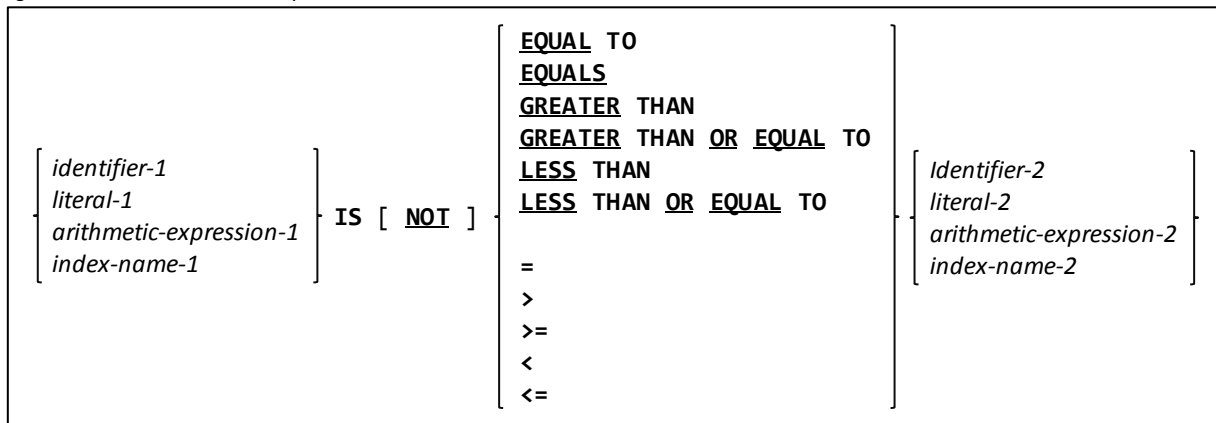
In the **SPECIAL-NAMES** paragraph, an external switch name can be associated with one or more condition names. These condition names may then be used to test the ON/OFF status of the external switch. An example is shown to the left.

See Also...

The **SPECIAL-NAMES** Paragraph [4.1.4](#)

6.1.8.2.5. Relation Conditions

Figure 6-15 - Relation Condition Syntax



These conditions evaluate how two different values “relate” to each other.

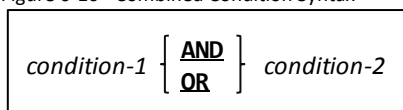
1. When comparing one numeric value to another, the **USAGE** and number of significant digits in either value are irrelevant as the comparison is performed using the actual algebraic values.
2. When comparing strings, the comparison is made based upon the program’s collating sequence (see section). When the two string arguments are of unequal length, the shorter is assumed to be padded (on the right) with a sufficient number of **SPACES** as to make the two strings of equal length. String comparisons take place on a corresponding character-by-character basis until a pair of characters is found that violates the condition being tested for based upon the relative position of where each character in the pair falls in the program’s **COLLATING SEQUENCE** (as defined in **SPECIAL-NAMES**).
3. There is no functional difference between using the wordy version (“IS EQUAL TO”, “IS LESS THAN”, ...) versus the symbolic version (“=”, “<”, ...) of the actual relation operators.

See Also...

The **SPECIAL-NAMES** Paragraph [4.1.4](#)

6.1.8.2.6. Combined Conditions

Figure 6-16 - Combined Condition Syntax



A combined condition is one that computes a TRUE/FALSE value from the TRUE/FALSE values of two other conditions (which could – themselves – be combined conditions).

1. If either condition has a value of TRUE, the result of **OR**ing the two together will result in a value of TRUE. Only when **OR**ing two FALSE conditions will a result of FALSE occur.
2. In order for **AND** to yield a value of TRUE, both conditions must have a value of TRUE. In all other circumstances, **AND** produces a FALSE value.
3. When chaining multiple, similar conditions together with the same operator (**OR/AND**), and left or right arguments having common operators and subjects, it is possible to abbreviate the program code. For example:

```
IF ACCOUNT-STATUS = 1 OR ACCOUNT-STATUS = 2 OR ACCOUNT-STATUS = 7
```

Could be abbreviated as:

```
IF ACCOUNT-STATUS = 1 OR 2 OR 7
```

4. Just as multiplication takes precedence over addition in arithmetic expressions, so does **AND** take precedence over **OR** in combined conditions. Use parenthesis to change this precedence, if necessary. For example:

```
FALSE OR FALSE AND TRUE    evaluates to TRUE
```

```
FALSE OR (FALSE AND TRUE)   evaluates to TRUE (since AND has precedence over OR, this is identical to the previous example)
```

```
(FALSE OR FALSE) AND TRUE   evaluates to FALSE
```

6.1.8.2.7. Negated Conditions

Figure 6-17 - Negated Condition Syntax

NOT <i>condition-1</i>

A condition may be negated by prefixing it with the **NOT** operator.

1. The **NOT** operator has the highest precedence of all logical operators, just as a unary minus sign (which “negates” a numeric value) is the highest precedence arithmetic operator.
2. Parenthesis must be used to explicitly signify the sequence in which conditions are evaluated and processed if the default precedence isn’t desired. For example:

NOT TRUE AND FALSE AND NOT FALSE	evaluates to FALSE AND FALSE AND TRUE which evaluates to FALSE
NOT (TRUE AND FALSE AND NOT FALSE)	evaluates to NOT (FALSE) which evaluates to TRUE
NOT TRUE AND (FALSE AND NOT FALSE)	evaluates to FALSE AND (FALSE AND TRUE) which evaluates to FALSE

6.1.9. Use of Periods (.)

All COBOL implementations distinguish between sentences and statements in the **PROCEDURE DIVISION**. A *statement* is a single executable COBOL instruction. For example, these are all statements:

```
MOVE SPACES TO Employee-Address
ADD 1 TO Record-Counter
DISPLAY "Record-Counter=" Record-Counter
```

Some COBOL statements have a “scope of applicability” associated with them where one or more other statements can be considered to be part of or related to the statement in question. An example of such a situation might be the following, where the interest on a loan is being calculated and displayed - 4% interest if the loan balance is under \$10000 and 4.5% otherwise:

```
IF Loan-Balance < 10000
  MULTIPLY Loan-Balance BY 0.04 GIVING Interest
ELSE
  MULTIPLY Loan-Balance BY 0.045 GIVING Interest
DISPLAY "Interest Amount = " Interest
```

In this example, the **IF** statement actually has a scope that can include two sets of associated statements – one set to be executed when the **IF** condition is TRUE and another if it is FALSE.

Unfortunately, there’s a problem with the above. A human being looking at that code will probably understand that the **DISPLAY** statement, because of its lack of indentation, is to be executed regardless of the TRUE/FALSE value of the **IF** condition. Unfortunately, the GNU COBOL compiler (or any other COBOL compiler for that matter) won’t see it that way because it really couldn’t care less what sort of indentation, if any, is used. In fact, any COBOL compiler would be just as happy to see the code written like this:

```
IF Loan-Balance < 10000 MULTIPLY Loan-balance BY 0.04 GIVING Interest ELSE MULTIPLY
Loan-Balance BY 0.045 GIVING Interest DISPLAY "Interest Amount = " Interest
```

So how then do we inform the compiler that the **DISPLAY** statement is outside the scope of the **IF**?

That’s where *sentences* come in.

A COBOL *sentence* is defined as any arbitrarily long sequence of statements, followed by a period (.) character. The period character is what terminates the scope of a set of statements. Therefore, our example needs to be coded like this:

```
IF Loan-Balance < 10000
  MULTIPLY Loan-Balance BY 0.04 GIVING Interest
ELSE
  MULTIPLY Loan-Balance BY 0.045 GIVING Interest.
DISPLAY "Interest Amount = " Interest
```

See the period at the end of the second **MULTIPLY** (I **highlighted** it)? That is what terminates the scope of the “**IF**”, thus making the **DISPLAY** something that will be executed regardless of how the “Loan-Balance < 10000” test evaluated.

6.1.10. Use of “VERB” / “END-VERB” Constructs

Prior to the 1985 COBOL standard, using a period character was the only way to signal the end of a statement’s scope. Unfortunately, this caused some problems. Take a look at this code:

```
IF A = 1
  IF B = 1
    DISPLAY “A & B = 1”
ELSE
  IF B = 1
    DISPLAY “A NOT = 1 BUT B = 1”
ELSE
  DISPLAY “NEITHER A NOR B = 1”.
```

This sort of problem led to the “band-aid” solution¹⁹ shown to the right being added to the COBOL language.

The problem with this code is that indentation – so critical for improving the human-readability of a program – provides an erroneous view of the logical flow. An **ELSE** is always associated with the most-recently encountered IF; this means the **highlighted ELSE** will be associated with the “**IF B = 1**” statement, not the “**IF A = 1**” statement.

```
IF A = 1
  IF B = 1
    DISPLAY “A & B = 1”
  ELSE
    NEXT SENTENCE
ELSE
  IF B = 1
    DISPLAY “A NOT = 1 BUT B = 1”
  ELSE
    DISPLAY “NEITHER A NOR B = 1”.
```

The **NEXT SENTENCE** statement informs the GNU COBOL compiler that if the “**B = 1**” condition is false, control should fall into the first statement that follows the next period.

With the 1985 standard for COBOL, a much more elegant solution was introduced. Those COBOL verbs (statements) that needed such a thing were allowed to use an “END-verb” construct to end their scope without disrupting the scope of any statements whose scope they might have been in. Any COBOL 85 compiler would have allowed the following solution to our problem:

```
IF A = 1
  IF B = 1
    DISPLAY “A & B = 1”
  END-IF
ELSE
  IF B = 1
    DISPLAY “A NOT = 1 BUT B = 1”
  ELSE
    DISPLAY “NEITHER A NOR B = 1”.
```

This new facility made the period almost obsolete, as our program segment would probably be coded like this today:

```
IF A = 1
  IF B = 1
    DISPLAY “A & B = 1”
  END-IF
ELSE
  IF B = 1
    DISPLAY “A NOT = 1 BUT B = 1”
  ELSE
    DISPLAY “NEITHER A NOR B = 1”
  END-IF
END-IF
```

COBOL (GNU COBOL included) still requires that each PROCEDURE DIVISION paragraph contain at least one *sentence* if there is any executable code in that paragraph, but a popular coding standard is now to simply code a single period right before the end of each paragraph. Check out the “GCic” sample program in section [10.4](#) and you’ll see how that would be done.

¹⁹ Yes, I realize you could have easily fixed the problem by changing the code to “**IF A = 1 AND B = 1**”, but that wouldn’t have allowed me to make my case here

The standard for the COBOL language shows the various “END-verb” specifications to be optional because using a period as a scope-terminator remains legal. Some statements have an “END-verb” scope-terminator defined for them that they don’t appear to need.²⁰

If you will be porting existing code over to GNU COBOL, you’ll find it an accommodating facility capable of conforming to language and coding standards that code is likely to use. If you are creating new GNU COBOL programs, however, I would strongly counsel you to use the “END-verb” structures religiously in those programs.

See Also...

The **NEXT SENTENCE** Statement [6.4.28](#)

6.1.11. Controlling Concurrent Access to Files

The manipulation of data files is one of the COBOL language’s great strengths. There are features built-in to the COBOL language to deal with the possibility that multiple programs may be attempting to access the same file concurrently. Multiple program concurrent access is dealt with in two ways – *file sharing* and *record locking*.

Not all GNU COBOL implementations support file sharing and record-locking options. Whether they do or not depends upon the operating system they were built for and the build options that were used when the specific GNU COBOL implementation was generated.

6.1.11.1. File Sharing

GNU COBOL controls concurrent-file access at the highest level through the concept of file sharing, enforced when a program attempts to **OPEN** a file. This is accomplished via a UNIX operating-system routine called “**fcntl()**”. That module is not currently supported by Windows²¹ and is not present in the MinGW Unix-emulation package. GNU COBOL builds created using a MinGW environment will be incapable of supporting file-sharing controls – files will always be shared in such environments. A GNU COBOL build created using the Cygwin environment on Windows would have access to “**fcntl()**” and therefore will support file sharing. Of course, actual Unix builds of GNU COBOL, as well as OSX builds²², should have no issues because “**fcntl()**” should be available.

Any limitations you impose on a successful **OPEN** will remain in place until your program either issues a **CLOSE** against the file or terminates.

There are three ways in which concurrent access to a file may be controlled at the file level:

Sharing Option on “OPEN”	Effect
ALL OTHER	When your program opens a file in this manner, no restrictions will be placed on other programs attempting to OPEN the file after your program did. This is the default sharing mode.
NO OTHER	When your program opens a file in this manner, your program announces that it is unwilling to allow <u>any</u> other program to have <u>any</u> access to the file as long as you are using that file; OPEN attempts made in other programs will fail with a file status of 37 (“PERMISSION DENIED”) until such time as you CLOSE the file.
READ ONLY	Opening a file in this manner indicates you are willing to allow other programs to OPEN the file for INPUT while you have it OPEN . If they attempt any other OPEN , their OPEN will fail with a file status of 37.

²⁰ **STRING** (section [6.2.43](#)) and **UNSTRING** (section [6.2.49](#)), for example – could it be there are plans in the works for a future standard to introduce an option to such statements that would need a scope-terminator?

²¹ Windows has other means of providing equivalent functionality to “**fcntl()**”, but the BDB package was not coded to utilize them. The use of other advanced file I/O packages that support both the UNIX and Windows concurrent-access routines (such as VBISAM) are currently under review by the author.

²² Apple Computer’s OSX operating system is based on an open-source version of UNIX (Darwin) and therefore includes support for “**fcntl()**”.

Of course, your program may fail if someone else got to the file first and **OPEN**ed it with a sharing option that imposed file-sharing limitations.

See Also...

FILE-STATUS Values [Figure 4-15](#)

The **CLOSE** Statement [6.4.7](#)

The **OPEN** Statement [6.4.29](#)

6.1.11.2. Record Locking

Record-locking is supported by advanced file-management software that provides a single point-of-control for access to files (usually **ORGANIZATION INDEXED** files). One such runtime package capable of doing this is the Berkely Database (BDB) package – a package frequently used in GNU COBOL builds to support **ORGANIZATION INDEXED** files. The various I/O statements are capable of imposing limitations on the access – by other concurrently-executing programs – to the file record they just accessed. These limitations are syntactically imposed by placing a lock on the record. Other records in the file remain available, assuming that file-sharing limitations imposed at **OPEN**-time didn't prevent access to the entire file.

Locks remain in-effect until a program holding the lock terminates or issues a **CLOSE** or **UNLOCK** against the file or executes a **COMMIT** or **ROLLBACK** statement.

The record locking options (not all options are available to all statements) are as shown in the following table.

Record Locking Option	Effect
WITH LOCK	Access to the record by other programs will be denied.
WITH KEPT LOCK	Normally, as a new record is accessed locks held for previous records are released. By using this option, not only is the newly-accessed record locked (as WITH LOCK would do), but prior record locks will be retained as well. A subsequent READ <u>without</u> the KEPT LOCK option will release all “kept” locks, as will the FREE statement.
WITH NO LOCK	The record will not be locked. This is the default locking option in effect for all statements.
IGNORING LOCK WITH IGNORE LOCK	This option is possible only when reading records – it informs GNU COBOL that any locks held by other programs should be ignored. The two options shown are synonymous.
WITH WAIT	This option is possible only when reading records – it informs GNU COBOL that the program is willing to wait for a lock held on the record being read to be released. Without this option, an attempt to read a locked record will be immediately aborted and a file status of 47 will be returned. With this option, the program will wait for a pre-configured time for the lock to be released. If the lock is released within the preconfigured wait time, the read will be successful. If the pre-configured wait time expires before the lock is released, the read attempt will be aborted and a 47 file status will be issued.

If the GNU COBOL build you are using was configured to use the Berkely Database (BDB) package for **INDEXED** file I/O, record locking will be available by using the execution-time environment variable **DB_HOME**.

See Also...

FILE-STATUS Values [Figure 4-15](#)

The **FREE** Statement [6.4.17](#)

The **CLOSE** Statement [6.4.7](#)

The **ROLLBACK** Statement [6.4.37](#)

The **COMMIT** Statement [6.4.8](#)

The **UNLOCK** Statement [6.4.48](#)

Execution-time Environment Variables [8.2.4](#)

6.1.12. Common Clauses On Executable Statements

6.1.12.1. AT END / NOT AT END

AT END clauses may be specified on **READ** and **RETURN** statements.

[**AT END** [imperative statement-1](#)]

[NOT AT END [imperative-statement-2](#)]

1. The optional **AT END** clause will – if present on a **READ** or **RETURN** statement – cause *imperative-statement-1* to be executed if the **READ** or **RETURN** attempt fails due to a File-Status of 10 (end-of-file).
2. An **AT END** clause **WILL NOT DETECT OTHER NON-ZERO FILE-STATUS VALUES**. See [Figure 4-15](#) for a list of possible File-Status values.
3. Use a **DECLARATIVES** routine (section) or an explicitly-declared file status field tested after the **READ** or **RETURN** to detect error conditions other than end-of-file.
4. An optional **NOT AT END** clause will cause *imperative-statement-2* to be executed if the **READ** or **RETURN** attempt is successful.

5. **See Also...**

Using **DECLARATIVES** [6.1.4](#)

The **READ** Statement [6.4.31](#)

The **RETURN** Statement [6.2.35](#)

6.1.12.2. CORRESPONDING Option

Three GNU COBOL verbs – **ADD** (section [6.4.2.3](#)), **MOVE** (section [6.4.26.2](#)) and **SUBTRACT** (section [6.4.44.3](#)) support the use of a “**CORRESPONDING**” option that allows multiple data items within one group item (*group-item-1* – the first named on the statement) to be paired with multiple corresponding data items (hence the name) in a second group item (*group-item-2* – the second named on the statement). The contents of *group-item-1* will remain unaffected by the statement while one or more data items within *group-item-2* will be changed.

In order for *data-item-1*, defined subordinate to group item *group-item-1* to be a “**CORRESPONDING**” match to *data-item-2* which is subordinate to *group-item-2*, each of the following must be true:

1. Both *data-item-1* and *data-item-2* must have the same name, and that name may not explicitly or implicitly be **FILLER**.
2. Both *data-item-1* and *data-item-2*...
 - a. ...must exist at the same relative structural “depth” of definition within *group-item-1* and *group-item-2*, respectively
 - b. ...and all “parent” data items defined within each group item must have identical (but non-“**FILLER**”) names.
3. When used with a **MOVE** verb...
 - a. ...one of *data-item-1* or *data-item-2* (but not both) is allowed to be a group item
 - b. ...and it must be valid to **MOVE** *data-item-1* TO *data-item-2*.
4. When used with **ADD** or **SUBTRACT** verbs, both *data-item-1* and *data-item-2* must be numeric, elementary, unedited items. Stated in different terms, neither *data-item-1* nor *data-item-2* may be group, alphabetic, alphanumeric or numeric-edited items.
5. Neither *data item-1* nor *data-item-2* may be a **REDEFINES** or **RENAMES** of another data item.
6. Neither *data item-1* nor *data-item-2* may have an **OCCURS** clause. Either may contain subordinate data items that have an **OCCURS** clause, however (assuming rule 3a applies)

Observe the following two group item structures...

```

03 X.
05 A          PIC 9(1).
05 G1.
   10 G2.
       15 B    PIC X(1).
05 C.
   10 FILLER  PIC X(1).
05 G3.
   10 G4.
       15 D    PIC X(1).
05 E          PIC X(1).
05 F          REDEFINES V1 PIC X(1).
05 G.
   10 G6      OCCURS 4 TIMES PIC X(1).
05 H          PIC X(4).
05 I          PIC 9(1).
05 J.
   10 K.
       15 M    PIC X(1).

```

```

01 Y.
02 A          PIC X(1).
02 G1.
   03 G2.
       04 B    PIC X(1).
02 C          PIC X(1).
02 G3.
   03 G5.
       04 D    PIC X(1).
   03 G6      PIC X(1).
02 E          PIC 9(1).
02 F          PIC X(1).
02 G          PIC X(4).
02 H          OCCURS 4 TIMES PIC X(1).
66 I          RENAMES E.
02 J.
   03 K.
       04 L.
           05 M.

```

The following are the valid **CORRESPONDING** matches, assuming the statement **MOVE CORRESPONDING X TO Y** is being used (there are no valid **CORRESPONDING** matches for **ADD CORRESPONDING** or **SUBTRACT CORRESPONDING** because every potential matchup violates rule #4): **A, B, C, G**

The following are the “**CORRESPONDING**” matchups that failed, and the reasons why they failed.

D Fails due to rule #2b	G3 Fails due to rule #3a	I Fails due to rule #5
E Fails due to rule #3b	G4 Fails due to rule #1	J Fails due to rule #3a
F Fails due to rule #5	G5 Fails due to rule #1	K Fails due to rule #3a
G1 Fails due to rule #3a	G6 Fails due to rule #6	L Fails due to rule #1
G2 Fails due to rule #3a	H Fails due to rule #6	M Fails due to rule #2a

See Also...

The ADD CORRESPONDING Statement	4	The SUBTRACT CORRESPONDING Statement	6.2.44.3
The MOVE CORRESPONDING Statement	6.2.26.2		

6.1.12.3. INVALID KEY / NOT INVALID KEY

INVALID KEY clauses may be specified on **DELETE**, **READ** (Random), **REWRITE**, **START** and **WRITE** statements.

```

[ ON INVALID KEY imperative statement-1 ]
[ NOT ON INVALID KEY imperative statement-2 ]

```

Specification of an **INVALID KEY** clause will allow your program to trap an I/O failure condition (with an I/O error code in the file’s **FILE-STATUS** field) that has occurred due to a record-not-found condition and handle it gracefully.

See Also...

Defining File Characteristics (SELECT)	4.2.1	The Random READ Statement	6.2.31.2
FILE-STATUS Values	Figure 4-15	The REWRITE Statement	6.4.36
The DELETE Statement	6.4.11	The START Statement	6.2.41
		The WRITE Statement	6.4.50

6.1.12.4. ON EXCEPTION / NOT ON EXCEPTION

EXCEPTION clauses may be specified on **ACCEPT**, **CALL** and **DISPLAY** statements.

```

[ ON EXCEPTION ERROR imperative statement-1 ]
[ NOT ON EXCEPTION ERROR imperative statement-2 ]

```

Specification of an **ON EXCEPTION** clause will allow your program to trap the failure condition that has occurred and handle it gracefully. If such a condition occurs at runtime without having one of these clauses specified, an error message will be generated (by the GNU COBOL runtime library) to the **SYSERR** device (pipe 2). The program may also be terminated, depending upon the type and severity of the error.

See Also...

The ACCEPT Statement (Command Line)	6.2.1.2
The ACCEPT Statement (Screen Data)	6.4.1.4
The CALL Statement	6.4.5
The DISPLAY Statement (Console/Stdout)	6.2.12.1

The DISPLAY Statement (Command Line)	6.2.12.2
The DISPLAY Statement (Environment)	6.2.12.3
The DISPLAY Statement (Screen Data)	6.4.12.4

6.1.12.5. ON OVERFLOW / NOT ON OVERFLOW

OVERFLOW clauses may be specified on **CALL**, **STRING** and **UNSTRING** statements.

```
[ ON OVERFLOW ERROR imperative statement-1
]
[ NOT ON OVERFLOW ERROR imperative statement-2
]
```

Specification of an **ON OVERFLOW** clause will allow your program to trap the failure condition that has occurred and handle it gracefully. If such a condition occurs at runtime without having one of these clauses specified, an error message will be generated (by the GNU COBOL runtime library) to the SYSERR device (pipe 2). The program may also be terminated, depending upon the type and severity of the error.

See Also...

The CALL Statement	6.4.5
The STRING Statement	6.2.43

The UNSTRING Statement	6.2.49
-------------------------------	------------------------

6.1.12.6. ON SIZE ERROR / NOT ON SIZE ERROR

SIZE ERROR clauses may be included on **ADD**, **COMPUTE**, **DIVIDE**, **MULTIPLY** and **SUBTRACT** statements.

```
[ ON SIZE ERROR imperative statement-1 ]
[ NOT ON SIZE ERROR imperative statement-2 ]
```

Specification of an **ON SIZE ERROR** clause will allow your program to trap the failure condition that has occurred and handle it gracefully. Field size overflow conditions occur silently, usually without any runtime messages being generated, even though such events rarely lend themselves to generating correct results. Division by zero errors, when no **ON SIZE ERROR** clause exists, will produce an error message (by the GNU COBOL runtime library) to the SYSERR device (pipe 2) and will also abort the program.

See Also...

The ADD Statement	6.4.2
The COMPUTE Statement	6.4.9
The DIVIDE Statement	6.4.13

The MULTIPLY Statement	6.4.27
The SUBTRACT Statement	6.4.44

6.1.12.7. Rounding Options

GNU COBOL provides for control over the final rounding process applied to the receiving fields on all arithmetic verbs. Each of the arithmetic statements (**ADD**, **COMPUTE**, **DIVIDE**, **MULTIPLY** and **SUBTRACT**) statements provide an optional **ROUNDED** clause to each receiving data item. The syntax of this clause is shown to the right.

The following rules apply to the rounding behavior induced by this clause.

```
ROUNDED  MODE IS [
                  AWAY-FROM-ZERO
                  NEAREST-AWAY-FROM-ZERO
                  NEAREST-EVEN
                  NEAREST-TOWARD-ZERO
                  PROHIBITED
                  TOWARD-GREATER
                  TOWARD-LESSER
                  TRUNCATION
                ]
```

1. Rounding only applies when the result being saved to the receiving field having a **ROUNDED** clause is a non-integer value
2. Absence of a **ROUNDED** clause is the same as specifying **ROUNDED MODE IS TRUNCATION**.

3. Use of a **ROUNDED** clause without a **MODE** specification is the same as specifying **ROUNDED MODE IS NEAREST-AWAY-FROM-ZERO**.
4. The behavior of the eight different rounding modes is defined in the following table.

Figure 6-18 - ROUNDED MODE Behavior

MODE	Behavior - Examples assume an integer receiving field – An ellipsis (...) indicates the last result value digit repeats	Result	Becomes	Result	Becomes
AWAY-FROM-ZERO	Rounding is to the nearest value of larger magnitude.	+2.499...	+3	-3.499...	-4
		-2.499...	-3	+3.500	+4
		+2.500	+3	-3.500	-4
		-2.500	-3	3.510	+4
		+3.499...	+4	-3.510	-4
NEAREST-AWAY-FROM-ZERO	Rounding is to the nearest value (larger or smaller). If two values are equally near, the value with the larger absolute value is selected.	+2.499...	+2	-3.499...	-3
		-2.499...	-2	+3.500	+4
		+2.500	+3	-3.500	-4
		-2.500	-3	3.510	+4
		+3.499...	+3	-3.510	-4
NEAREST-EVEN	Rounding is to the nearest value (larger or smaller). If two values are equally near, the value whose rightmost digit is even is selected. This mode is sometimes called "Banker's rounding".	+2.499...	+2	-3.499...	-3
		-2.499...	-2	+3.500	+4
		+2.500	+2	-3.500	-4
		-2.500	-2	3.510	+4
		+3.499...	+3	-3.510	-4
NEAREST-TOWARD-ZERO	Rounding is to the nearest value (larger or smaller). If two values are equally near, the value with the smaller absolute value is selected.	+2.499...	+2	-3.499...	-3
		-2.499...	-2	+3.500	+3
		+2.500	+2	-3.500	-3
		-2.500	-2	3.510	+4
		+3.499...	+3	-3.510	-4
PROHIBITED	No rounding is performed. If the value cannot be represented exactly in the desired format, the EC-SIZE-TRUNCATION condition (exception code 1005) is set to exist (and may be retrieved via the ACCEPT statement) and the results of the operation are undefined.	+2.499...	Undefined	-3.499...	Undefined
		-2.499...			
		+2.500			
		-2.500			
		+3.499...			
TOWARD-GREATER	Rounding is toward the nearest value whose algebraic value is larger.	+2.499...	+3	-3.499...	-3
		-2.499...	-2	+3.500	+4
		+2.500	+3	-3.500	-3
		-2.500	-2	3.510	+4
		+3.499...	+4	-3.510	-3
TOWARD-LESSER	Rounding is toward the nearest value whose algebraic value is smaller.	+2.499...	+2	-3.499...	-4
		-2.499...	-3	+3.500	+3
		+2.500	+2	-3.500	-4
		-2.500	-3	3.510	+3
		+3.499...	+3	-3.510	-4
TRUNCATION	Rounding is to the nearest value whose magnitude is smaller.	+2.499...	+2	-3.499...	-3
		-2.499...	-2	+3.500	+3
		+2.500	+2	-3.500	-3
		-2.500	-2	3.510	+3
		+3.499...	+3	-3.510	-3

See Also...

The ACCEPT Statement (Run-time Info) 6.2.1.7	The DIVIDE Statement 6.4.13
The ADD Statement 6.4.2	The MULTIPLY Statement 6.4.27
The COMPUTE Statement 6.4.9	The SUBTRACT Statement 6.4.44

6.1.13. Special Registers

GNU COBOL, like other COBOL dialects, includes a number of data items that are automatically available to a programmer without the need to actually define them in the **DATA DIVISION**. COBOL refers to such items as registers or special registers. The special registers available to a GNU COBOL program are as follows:

Figure 6-19 - Special Registers

Register Name	Implied COBOL PIC/USAGE ²³	Usage
COB-CRT-STATUS	PIC 9(4)	This is the default data item allocated for use by format 4 of the ACCEPT statement.
DEBUG-ITEM Subordinate items: DEBUG-LINE DEBUG-NAME DEBUG-SUB-1 DEBUG-SUB-2 DEBUG-SUB-3 DEBUG-CONTENTS	PIC X(88) (group item)	A group item in which debugging information generated by a USE FOR DEBUGGING section in the DECLARATIVES area will place information documenting why the USE FOR DEBUGGING procedure was invoked.
LINAGE-COUNTER	BINARY-LONG SIGNED	An occurrence of this register exists for each SELECTed file having a LINAGE clause. If there are multiple files whose FDs have a LINAGE clause, any explicit references to this register will require qualification (using " OF file-name "). The value of this register will be the current logical line number within the page body. DO NOT MODIFY THE CONTENTS OF THIS REGISTER.
NUMBER-OF-CALL-PARAMETERS	BINARY-LONG SIGNED	This register contains the number of arguments passed to a subroutine – the same value that would be returned by the C\$NARG built-in subroutine. Its value will be zero when referenced in a main program. This register, when referenced from within a user-defined function, returns a value of one (1) if the function has any number of arguments and a zero if it has no arguments.
RETURN-CODE	BINARY-LONG SIGNED	This register provides a numeric data item into which a subroutine may MOVE a value prior to transferring control back to the program that CALLed it, or into which a main program may MOVE a value before returning control to the operating system. Many built-in subroutines will return a value using this register. These values are – by convention – used to signify success (usually with a value of 0) or failure (usually with a non-zero value) of the process the program setting the RETURN-CODE value was attempting to perform. Chapter 7 discusses the role this special register plays with subprograms.
SORT-RETURN	BINARY-LONG SIGNED	This register is used to report the success/fail status of a RELEASE or RETURN statement. A value of 0 is reported on success. A value of 16 denotes failure. An " AT END " condition on a RETURN is <u>not</u> considered a failure.
WHEN-COMPILED	PIC X(16)	This register contains the date and time the program was compiled in the format " mm/dd/yyhh.mm.ss ". Note that only a two-digit year is provided.

²³ See sections 5.2.1.6 and [5.2.1.11](#) for a description of the PICTURE and USAGE specifications, respectively

See Also...

Describing the Structure of a File (FD/SD)	5.1
Using DECLARATIVES	6.1.4
The ACCEPT Statement (Screen Data)	6.4.1.4

The RELEASE Statement	6.2.33
The RETURN Statement	6.2.35
The C\$NARG Subroutine	8.3.1.9

6.1.14. Intrinsic Functions

GNU COBOL supports a variety of “intrinsic functions” that may be used anywhere in the **PROCEDURE DIVISION** where a literal is allowed. For example:

```
MOVE FUNCTION LENGTH(Employee-Last-Name) TO Employee-LN-Len.
```

Note how the word “**FUNCTION**” is part of the syntax when you use an intrinsic function. You can use intrinsic functions without having to include the reserved word **FUNCTION** via settings in the **REPOSITORY** paragraph of the **CONFIGURATION SECTION**. You may accomplish the same thing by specifying the “-fintrinsics” option to the GNU COBOL compiler when you compile your programs.

The following intrinsic functions, known to other “dialects” of COBOL, are defined to GNU COBOL as reserved words but are not otherwise implemented currently. Any attempts to use these functions will result in a compile-time error message.

BOOLEAN-OF-INTEGERS	FORMATTED-CURRENT-DATE	INTEGER-OF-FORMATTED-DATE
CHAR-NATIONAL	FORMATTED-DATE	NATIONAL-OF
DISPLAY-OF	FORMATTED-DATETIME	STANDARD-COMPARE
EXCEPTION-FILE-N	FORMATTED-TIME	TEST-FORMATTED-DATETIME
EXCEPTION-LOCATION-N	INTEGER-OF-BOOLEAN	

The supported intrinsic functions are listed in the following sections, along with their syntax and usage notes.

See Also...

The REPOSITORY Paragraph	4.1.3
---------------------------------	-----------------------

Compiler Switches Reference	8.1.2
-----------------------------	-----------------------

6.1.14.1. ABS(*number*)

Determines and returns the absolute value of the *number* (a numeric literal or data item) supplied as an argument.

6.1.14.2. ACOS(*cosine*)

The ACOS function determines and returns the trigonometric arc-cosine, or inverse cosine, of the *cosine* value (a numeric literal or data item) supplied as an argument.

6.1.14.3. ANNUITY(*interest-rate*, *number-of-periods*)

This function returns a numeric value approximating the ratio of an annuity paid at the specified *interest-rate* (numeric data items or literals) for each of the specified *number-of-periods* (numeric data items or literals).

The *interest-rate* is the rate of interest paid at each payment. If you only have an annual interest rate and you wish to compute annuity payments for monthly payments, divide the annual interest rate by 12 and use that value for *interest-rate* on this function.

Multiply this result times the desired principal amount to determine the amount of each period’s payment.

A note for the financially challenged: an annuity is basically a reverse loan; an accountant would take the result of this function multiplied by -1 to compute a loan payment you are making.

6.1.14.4. ASIN(*sine*)

The ASIN function determines and returns the trigonometric arc-sine, or inverse sine, of the *sine* value (a numeric literal or data item) supplied as an argument.

6.1.14.5. ATAN(*tangent*)

Use this function to determine and return the trigonometric arc-tangent, or inverse tangent, of the *tangent* value (a numeric literal or data item) supplied as an argument.

6.1.14.6. BYTE-LENGTH(*string*)

BYTE-LENGTH returns the length – in bytes – of the specified string (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal). This intrinsic function is identical to the **LENGTH-AN** function. Note that the value returned by this function is not necessarily the number of characters making up the *string*, but rather the number of actual bytes required to store *string*.

For example, if *string* is encoded using a double-byte character set such as UNICODE (where each character is represented by 16 bits of storage, not the 8-bits inherent to character sets like ASCII or EBCDIC), then calling this function with a *string* argument whose **PICTURE** is **X(4)** would return a value of 8 rather than the value 4.

6.1.14.7. CHAR(*integer*)

This function returns the character in the ordinal position specified by the *integer* argument (a numeric integer literal or data item) from the collating sequence being used by the program.

For example, if the program is using the (default) ASCII character set, **CHAR(34)** returns the 34th character in the ASCII character set – an exclamation-point (“!”). If you are using this function to convert a numeric value to its corresponding ASCII character, you must use an argument value one greater than the numeric value.

If an argument whose value is less than 1 or greater than 256 is specified, the character in the program collating sequence corresponding to a value of all zero bits is returned.

The following code is an alternative approach when you just wish to convert a number to its ASCII equivalent:

```
01 Char-Value.
   05 Numeric-Value          USAGE BINARY-CHAR.
.
.
.
   MOVE numeric-character-value TO Numeric-Value
   The Char-Value item now has the corresponding ASCII character value
```

6.1.14.8. COMBINED-DATETIME(*days, seconds*)

This function returns a 12-digit result, the first seven digits of which are the integer value of the *days* argument (a numeric data item or literal) and the last five of which are the integer value of the *seconds* argument (also a numeric data item or literal).

If a *days* value less than 1 or greater than 3067671 is specified, or if a *seconds* value less than 1 or greater than 86400 is specified, a value of 0 is returned and a runtime error will result.

6.1.14.9. CONCATENATE(*string-1* [, *string-2*] ...)

This function concatenates the *string-1*, *string-2*, ... (group items, **USAGE DISPLAY** elementary items and/or alphanumeric literals) together into a single string result.

If a numeric literal or **PICTURE 9** identifier is specified as an argument, decimal points, if any, will be removed and negative signs in **PICTURE S9** fields or numeric literals will be inserted as defined by the **SIGN** clause (or absence thereof) of the field. Numeric literals are processed as if **SIGN IS TRAILING SEPARATE** were in effect.

See Also...

Defining Signed Data Items (SIGN) 5.2.1.9

6.1.14.10. COS(*angle*)

The **COS** function determines and returns the trigonometric cosine of the *angle* (a numeric literal or data item) supplied as an argument. The *angle* is assumed to be a value expressed in radians.

6.1.14.11. CURRENCY-SYMBOL

The **CURRENCY-SYMBOL** function returns the currency symbol character currently in effect for the locale under which your program is running. On UNIX systems, your locale is established via the LANG environment variable. On Windows, the Control Panel's Regional and Language Options define the locale.

Changing the currency symbol via the **SPECIAL-NAMES** paragraph's **CURRENCY SYMBOL** setting will not affect the value returned by this function.

See Also...

The **SPECIAL-NAMES** Paragraph [4.1.4](#)

6.1.14.12. CURRENT-DATE

Returns the current date and time as the following 21-character structure:

```

01 CURRENT-DATE-AND-TIME.
   05 CDT-Year           PIC 9(4).
   05 CDT-Month          PIC 9(2). * > 01-12
   05 CDT-Day            PIC 9(2). * > 01-31
   05 CDT-Hour           PIC 9(2). * > 00-23
   05 CDT-Minutes        PIC 9(2). * > 00-59
   05 CDT-Seconds        PIC 9(2). * > 00-59
   05 CDT-Hundredths-Of-Secs PIC 9(2). * > 00-99
   05 CDT-GMT-Diff-Hours PIC S9(2)
                        SIGN LEADING SEPARATE.
   05 CDT-GMT-Diff-Minutes PIC 9(2). * > 00 or 30

```

Since the **CURRENT-DATE** function has no arguments, no parenthesis should be specified.

6.1.14.13. DATE-OF-INTEGGER(*integer*)

This function returns a calendar date in `yyyymmdd` format. The date is determined by adding the number of days specified as *integer* (a numeric integer data item or literal) to December 31, 1600. For example, `DATE-OF-INTEGGER(1)` returns 16010101.

A value less than 1 or greater than 3067671 (9999/12/31) will return a result of 0.

6.1.14.14. DATE-TO-YYYYMMDD(*yymmdd* [, *yy-cutoff*])

You can use this function to convert the six-digit date specified as *yymmdd* (a numeric integer data item or literal) to an eight-digit format (`yyyymmdd`). The optional *yy-cutoff* (a numeric integer data item or literal) argument is the year cutoff used to delineate centuries; if the year component of the date meets or exceeds this cutoff value, the result will be 19*yymmdd*; if the year component of the date is less than the cutoff value, the result will be 20*yymmdd*. The default cutoff value if no second argument is given will be 50.

6.1.14.15. DAY-OF-INTEGGER(*integer*)

This function returns a calendar date in `yyyyddd` (i.e. Julian) format. The date is determined by adding the number of days specified as *integer* (a numeric integer data item or literal) to December 31, 1600. For example, `DATE-OF-INTEGGER(1)` returns 1601001.

A value less than 1 or greater than 3067671 (9999/12/31) will return a result of 0.

6.1.14.16. DAY-TO-YYYYDDD(*yyddd* [, *yy-cutoff*])

You can use this function to convert the five-digit Julian date specified as *yyddd* (a numeric integer data item or literal) to a seven-digit Julian format (`yyyyddd`). The optional *yy-cutoff* argument (a numeric integer data item or literal) is the year cutoff used to delineate centuries; if the year component of the date meets or exceeds this cutoff value, the result will be 19*yyddd*; if the year component of the date is less than the cutoff, the result will be 20*yyddd*. The default cutoff value if no second argument is given will be 50.

6.1.14.17. E

This function returns the mathematical constant “E” (the base of natural logarithms). The maximum precision with which this value may be returned is 2.7182818284590452353602874713526625.

Since the E function has no arguments, no parenthesis should be specified.

6.1.14.18. EXCEPTION-FILE

This function returns I/O exception information from the most-recently executed input or output statement. The information is returned to a structure resembling the following:

```
01 INPUT-OUTPUT-EXCEPTION.
   05 IOE-FILE-STATUS          PIC 9(2).
   05 IOE-FILE-SELECT-NAME     PIC X(32).
```

See [Figure 4-15](#) for information about possible file-status values.

The name returned after the file status information will be the “SELECT” name of the file, and it will be returned only if the returned file status value is not 00.

Since the **EXCEPTION-FILE** function has no arguments, no parenthesis should be specified.

The documentation of the CBL_ERROR_PROC built-in subroutine illustrates the use of this function.

See Also...

The **CBL_ERROR_PROC** Subroutine [8.3.1.24](#)

6.1.14.19. EXCEPTION-LOCATION

This function returns exception information from the most-recently failing statement. The information is returned to a 1023 character string in one of the following formats, depending on the nature of the failure:

- ▶ *primary-entry-point-name; paragraph OF section; statement-number*
- ▶ *primary-entry-point-name; section; statement-number*
- ▶ *primary-entry-point-name; paragraph; statement-number*
- ▶ *primary-entry-point-name; statement-number*

Since the **EXCEPTION-LOCATION** function has no arguments, no parenthesis should be specified.

The program must be compiled with the “-debug”, “-ftraceall” or “-g” option for this function to return any meaningful information.

The documentation of the CBL_ERROR_PROC built-in subroutine illustrates the use of this function.

See Also...

The **CBL_ERROR_PROC** Subroutine [8.3.1.24](#)

6.1.14.20. EXCEPTION-STATEMENT

This function returns the most-recent COBOL statement that generated an exception condition.

Since the **EXCEPTION-STATEMENT** function has no arguments, no parenthesis should be specified.

The program must be compiled with the “-debug”, “-ftraceall” or “-g” option for this function to return any meaningful information.

The documentation of the CBL_ERROR_PROC built-in subroutine illustrates the use of this function.

See Also...

The **CBL_ERROR_PROC** Subroutine [8.3.1.24](#)

6.1.14.21. EXCEPTION-STATUS

This function returns the error type (as a text string) from the most-recent COBOL statement that generated an exception condition. [Figure 6-28](#) shows a list of possible error types.

Since the **EXCEPTION-STATUS** function has no arguments, no parenthesis should be specified.

The documentation of the `CBL_ERROR_PROC` built-in subroutine illustrates the use of this function.

See Also...

The `CBL_ERROR_PROC` Subroutine [8.3.1.24](#)

6.1.14.22. **EXP(number)**

Computes and returns the value of the mathematical constant “e” raised to the power specified by *number* (a numeric literal or data item).

6.1.14.23. **EXP10(number)**

Computes and returns the value of 10 raised to the power specified by *number* (a numeric literal or data item).

6.1.14.24. **FACTORIAL(number)**

This function computes and returns the factorial value of *number* (a numeric literal or data item).

6.1.14.25. **FRACTION-PART(number)**

This function returns that portion of *number* that occurs to the right of the decimal point. *Number* must be a numeric data item or a numeric literal. **FRACTION-PART(3.1415)**, for example, returns a value of 0.1415. This function is equivalent to the expression:

number – FUNCTION INTEGER-PART(*number*)

6.1.14.26. **HIGHEST-ALGEBRAIC(numeric-identifier)**

This function returns the highest (i.e. largest or farthest away from 0 in a positive direction if *numeric-identifier* is signed) value that could possibly be stored in the specified *numeric-identifier*.

6.1.14.27. **INTEGER(number)**

The **INTEGER** function returns the greatest integer value that is less than or equal to *number* (a numeric literal or data item).

6.1.14.28. **INTEGER-OF-DATE(date)**

This function converts *date* (a numeric integer data item or literal) – presumed to be a Gregorian calendar form standard date (YYYYMMDD) - to integer date form – that is, the number of days that have transpired since 1600/12/31.

6.1.14.29. **INTEGER-OF-DAY(date)**

This function converts *date* (a numeric integer data item or literal) – presumed to be a Julian calendar form standard date (YYYYDDD) to integer date form – that is, the number of days that have transpired since 1600/12/31.

6.1.14.30. **INTEGER-PART(number)**

Returns the integer portion of the value of *number* (a numeric literal or data item).

6.1.14.31. **LENGTH(string)**

Returns the length – in *characters* – of *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal). Note that the value returned by this function is not the number of bytes of storage occupied by *string*, but rather the number of actual *characters* making up the *string*. For example, if *string* is encoded using a double-byte character set such as UNICODE (where each character is represented by 16 bits of storage, not the 8-bits inherent to

character sets like ASCII or EBCDIC), then calling this function with a *string* argument whose **PICTURE** is **X(4)** would return a value of 4 rather than the value 8 (the actual number of bytes of storage occupied by that item).

6.1.14.32. LENGTH-AN(*string*)

Returns the length – in bytes of storage – of *string* (a group item, USAGE DISPLAY elementary item or alphanumeric literal). This intrinsic function is identical to the **BYTE-LENGTH** function. Note that the value returned by this function is not the number of actual characters making up the *string*, bytes of storage occupied by *string*, but rather the number of actual bytes required to store *string*. For example, if *string* is encoded using a double-byte character set such as UNICODE (where each character is represented by 16 bits of storage, not the 8-bits inherent to character sets like ASCII or EBCDIC), then calling this function with a *string* argument whose **PICTURE** is **X(4)** would return a value of 8 rather than the value 4.

6.1.14.33. LOCALE-COMPARE(*argument-1*, *argument-2* [, *locale*])

The **LOCALE-COMPARE** function returns a character indicating the result of comparing *argument-1* and *argument-2* using a culturally-preferred ordering defined by a *locale*.

Either argument may be an alphanumeric literal, a group item or an elementary item appropriate to storing alphabetic or alphanumeric data. If the lengths of the two arguments are unequal, the shorter will be assumed to be padded to the right with **SPACES**.

The two arguments will be compared, character by character, against each other until their relationship to each other can be determined. The comparison is made according to the cultural rules in effect for the specified locale name or for the current locale if no locale argument is specified²⁴. Once that relationship is determined, a one-character alphanumeric value will be returned as follows:

- “<” If *argument-1* is determined to be less than *argument-2*
- “=” If the two arguments are equal to each other
- “>” If *argument-1* is determined to be greater than *argument-2*

6.1.14.34. LOCALE-DATE(*date* [, *locale*])

Converts the eight-digit Gregorian date (a numeric integer data item or literal) from YYYYMMDD format to the format appropriate to the current locale. On a Windows system, this will be the “short date” format as set using Control Panel.

You may include an optional second argument to specify the *locale* name (group item or **PIC X** identifier) you’d like to use for date formatting. If used, this second argument **MUST** be an identifier. Locale names are specified using UNIX-standard names. The complete list of supported locale names is shown in [Figure 4-7](#).

6.1.14.35. LOCALE-TIME(*time* [, *locale*])

Converts the four- (HHMM) or six-digit (HHMMSS) *time* (a numeric integer data item or literal) to a format appropriate to the current locale. On a Windows system, this will be the “time” format as set using Control Panel.

You may include an optional *locale* name (a group item or **PIC X** identifier) you’d like to use for time formatting. If used, this second argument **MUST** be an identifier. Locale names are specified using UNIX-standard names. The complete list of supported locale names is shown in [Figure 4-7](#).

6.1.14.36. LOCALE-TIME-FROM-SECS(*seconds* [, *locale*])

Converts the number of *seconds* since midnight (a numeric integer data item or literal) to a format appropriate to the current locale. On a Windows system, this will be the “time” format as set using Control Panel.

You may include an optional *locale* name (a group item or **PIC X** identifier) you’d like to use for time formatting. If used, this second argument **MUST** be an identifier. Locale names are specified using UNIX-standard names. The complete list of supported locale names is shown in [Figure 4-7](#).

²⁴ Locale-based ordering is not necessarily a character-by-character comparison.

6.1.14.37. LOG(*number*)

Computes and returns the natural logarithm (base “e”) of *number* (a numeric literal or data item).

6.1.14.38. LOG10(*number*)

Computes and returns the base 10 logarithm of *number* (a numeric literal or data item).

6.1.14.39. LOWER-CASE(*string*)

This function returns the value of *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal), converted entirely to lower case. Note that what constitutes a “letter” (or upper/lower case too, for that manner) may be influenced through the use of a **CHARACTER CLASSIFICATION** specification in the **OBJECT-COMPUTER** paragraph..

See Also...

The **OBJECT-COMPUTER** Paragraph [4.1.2](#)

6.1.14.40. LOWEST-ALGEBRAIC(*numeric-identifier*)

This function returns the lowest (i.e. smallest or farthest away from 0 in a negative direction if *numeric-identifier* is signed) value that could possibly be stored in the specified *numeric-identifier*.

6.1.14.41. MAX(*number-1* [, *number-2*] ...)

This function returns the maximum value from the specified list *numbers* (these may be numeric data items or literals).

6.1.14.42. MEAN(*number-1* [, *number-2*] ...)

This function returns the statistical mean value of the specified list *numbers* (these may be numeric data items or literals).

6.1.14.43. MEDIAN(*number-1* [, *number-2*] ...)

This function returns the statistical median value of the specified list *numbers* (these may be numeric data items or literals).

6.1.14.44. MIDRANGE(*number-1* [, *number-2*] ...)

The **MIDRANGE** (middle range) function returns a numeric value that is the arithmetic mean (average) of the values of the minimum and maximum *numbers* (these may be numeric data items or literals).

6.1.14.45. MIN(*number-1* [, *number-2*] ...)

This function returns the minimum value from the specified list *numbers* (these may be numeric data items or literals).

6.1.14.46. MOD(*value*, *modulus*)

Returns *value* modulo *modulus*. Both arguments may be PIC 9 data items or numeric literals. Either (or both) may have a non-integer value.

The result is determined according to the following formula:

$$value - (modulus * FUNCTION INTEGER (value / modulus))$$

6.1.14.47. MODULE-CALLER-ID

Returns the primary entry-point name (section 3) of the GNU COBOL program that **CALL**ed this one, or the null string if the program is a main program.

The discussion of the **MODULE-TIME** function includes a sample program that also uses this function.

See Also...

The **MODULE-TIME** Intrinsic Function [6.1.14.53](#)

6.1.14.48. MODULE-DATE

Returns the date the GNU COBOL program was compiled, in the form YYYYMMDD.

The discussion of the **MODULE-TIME** function includes a sample program that also uses this function.

See Also...

The **MODULE-TIME** Intrinsic Function [6.1.14.53](#)

6.1.14.49. MODULE-FORMATTED-DATE

Returns the fully-formatted date and time when the program was compiled. The exact format of this returned string value may vary depending on the operating system, GNU COBOL build type and/or LOCALE settings.

The discussion of the **MODULE-TIME** function includes a sample program that also uses this function.

See Also...

The **MODULE-TIME** Intrinsic Function [6.1.14.53](#)

6.1.14.50. MODULE-ID

Returns the primary entry-point name (section 3) of this GNU COBOL program.

The discussion of the **MODULE-TIME** function includes a sample program that also uses this function.

See Also...

The **MODULE-TIME** Intrinsic Function [6.1.14.53](#)

6.1.14.51. MODULE-PATH

This function returns the full path to the executable version of this GNU COBOL program. The filename component of this value will be exactly as typed on the command line, down to the use of upper- and lowercase letters and presence (or absence) of any extension.

The discussion of the **MODULE-TIME** function includes a sample program that also uses this function.

See Also...

The **MODULE-TIME** Intrinsic Function [6.1.14.53](#)

6.1.14.52. MODULE-SOURCE

The filename of the source code of the program (as specified on the “cobc” command when the program was compiled) is returned by this function.

The discussion of the **MODULE-TIME** function includes a sample program that also uses this function.

See Also...

The **MODULE-TIME** Intrinsic Function [6.1.14.53](#)

6.1.14.53. MODULE-TIME

This function returns the time the GNU COBOL program was compiled, in the form HHMMSS.

The following sample main program uses all the MODULE- Functions

```

IDENTIFICATION DIVISION.
PROGRAM-ID. DEMOMODULE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    FUNCTION ALL INTRINSIC.
PROCEDURE DIVISION.
000-Main.
    DISPLAY "MODULE-CALLER-ID      = [" MODULE-CALLER-ID "]"
    DISPLAY "MODULE-DATE          = [" MODULE-DATE   "]"
    DISPLAY "MODULE-FORMATTED-DATE = [" MODULE-FORMATTED-DATE "]"
    DISPLAY "MODULE-ID            = [" MODULE-ID     "]"
    DISPLAY "MODULE-PATH          = [" MODULE-PATH   "]"
    DISPLAY "MODULE-SOURCE        = [" MODULE-SOURCE "]"
    DISPLAY "MODULE-TIME          = [" MODULE-TIME   "]"
    STOP RUN
.

```

The program produces this output when executed:

```

MODULE-CALLER-ID      = []
MODULE-DATE           = [20120614]
MODULE-FORMATTED-DATE = [Jun 14 2012 15:07:45]
MODULE-ID             = [DEMOMODULE]
MODULE-PATH           = [E:\Programs\Demos\DEMOMODULE.exe]
MODULE-SOURCE         = [DEMOMODULE.cb1]
MODULE-TIME           = [150745]

```

6.1.14.54. MONETARY-DECIMAL-POINT

This function returns the character used to separate the integer portion from the fractional part of a monetary currency value according to the rules currently in effect for the locale under which your program is running. On UNIX systems, your locale is established via the LANG environment variable. On Windows, the Control Panel's Regional and Language Options define the locale.

Note that using the **SPECIAL-NAMES** paragraph's **DECIMAL-POINT IS COMMA** setting will not affect the value returned by this function.

See Also...

The **SPECIAL-NAMES** Paragraph [4.1.4](#)

6.1.14.55. MONETARY-THOUSANDS-SEPARATOR

This function returns the character used to separate the thousands digit groupings of monetary currency values according to the rules currently in effect for the locale under which your program is running. On UNIX systems, your locale is established via the LANG environment variable. On Windows, the Control Panel's Regional and Language Options define the locale.

Note that using the **SPECIAL-NAMES** paragraph's **DECIMAL-POINT IS COMMA** setting will not affect the value returned by this function.

See Also...

The **SPECIAL-NAMES** Paragraph [4.1.4](#)

6.1.14.56. NUMERIC-DECIMAL-POINT

This function returns the character used to separate the integer portion of a non-integer numeric item from the fractional part according to the rules currently in effect for the locale under which your program is running. On UNIX systems, your locale is established via the LANG environment variable. On Windows, the Control Panel's Regional and Language Options define the locale.

Note that using the **SPECIAL-NAMES** paragraph's **DECIMAL-POINT IS COMMA** setting will not affect the value returned by this function.

*See Also...*The **SPECIAL-NAMES** Paragraph [4.1.4](#)

6.1.14.57. NUMERIC-THOUSANDS-SEPARATOR

This function returns the character used to separate the thousands digit groupings of numeric values according to the rules currently in effect for the locale under which your program is running. On UNIX systems, your locale is established via the LANG environment variable. On Windows, the Control Panel's Regional and Language Options define the locale.

Note that using the **SPECIAL-NAMES** paragraph's **DECIMAL-POINT IS COMMA** setting will not affect the value returned by this function.

*See Also...*The **SPECIAL-NAMES** Paragraph [4.1.4](#)

6.1.14.58. NUMVAL(*string*)

Format 1:

$$[\textit{space} \dots] \left[\left[\begin{array}{c} + \\ - \end{array} \right] \right] [\textit{space} \dots] [\textit{digit} \dots] [\cdot [\textit{digit} \dots]] [\textit{space} \dots]$$
Format 2:

$$[\textit{space} \dots] [\textit{digit} \dots] [\cdot [\textit{digit} \dots]] [\textit{space} \dots] \left[\left[\begin{array}{c} + \\ - \\ \text{DB} \\ \text{CR} \end{array} \right] \right] [\textit{space} \dots]$$

The **NUMVAL** function converts a *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal) to its corresponding numeric value.

The *string* must have either of the formats shown here, where *space* represents a SPACE character and *digit* represents one of the digit characters "0" through "9". In addition, there must be at least one *digit* characters in the *string*.

If *string* does not conform to either of the formats shown here, a value of zero will be returned.

6.1.14.59. NUMVAL-C(*string* [, *symbol*])

Format 1:

$$[\textit{space} \dots] \left[\left[\begin{array}{c} + \\ - \end{array} \right] \right] [\textit{space} \dots] [\textit{currency}] [\textit{space} \dots] [\textit{digit} \dots] [\cdot [\textit{digit} \dots]] [\textit{space} \dots]$$
Format 2:

$$[\textit{space} \dots] [\textit{currency}] [\textit{space} \dots] [\textit{digit} \dots] [\cdot [\textit{digit} \dots]] [\textit{space} \dots] \left[\left[\begin{array}{c} + \\ - \\ \text{DB} \\ \text{CR} \end{array} \right] \right] [\textit{space} \dots]$$

This function converts a *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal) representing a currency value to its corresponding numeric value.

The *string* must have either of the formats shown here, where *space* represents a SPACE character, *digit* represents one of the digit characters "0" through "9" and *currency* represents a currency symbol (a "\$", for example). In addition, there must be at least one *digit* characters in the *string*.

The optional *symbol* character represents the currency symbol (a single-character group item, **USAGE DISPLAY** elementary item or alphanumeric literal) that may be used as the *currency* character in *string*. If no symbol is specified, the value that would be returned by the **CURRENCY-SYMBOL** intrinsic function will be used.

*See Also...*The **CURRENCY-SYMBOL** Intrinsic Function [6.1.7.11](#)

6.1.14.60. NUMVAL-F(*string*)

$$[\text{space...}] \left[\left[\begin{array}{c} + \\ - \end{array} \right] \right] [\text{space...}] [\text{digit...}] [\cdot [\text{digit...}]] \text{E} [\text{space...}] \left[\left[\begin{array}{c} + \\ - \end{array} \right] \right] \text{digit...} [\text{space...}]$$

This function converts a *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal) representing a floating-point value to its corresponding numeric value.

The *string* must have the format shown here, where *space* represents a SPACE character and *digit* represents one of the digit characters "0" through "9". In addition, there must be at least one *digit* character in the *string* to the left of the "E" character.

6.1.14.61. ORD(*char*)

This function returns the ordinal position in the program character set (usually ASCII) corresponding to the 1st character of the *char* argument (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal). For example, assuming the program is using the standard ASCII collating sequence, **ORD("I")** returns 34 because "I" is the 34th ASCII character. If you are using this function to convert an ASCII character to its numeric value, you must subtract one from the result.

The following code is an alternative approach when you just wish to convert an ASCII character to its numeric equivalent:

```
01 Char-Value.
   05 Numeric-Value          USAGE BINARY-CHAR.
.
.
.
   MOVE "character" TO Char-Value
   The Numeric-Value item now has the corresponding numeric value
```

6.1.14.62. ORD-MAX(*char-1* [, *char-2*] ...)

This function returns the ordinal position in the argument list corresponding to the argument whose 1st character has the highest position in the program collating sequence (usually ASCII). For example, assuming the program is using the standard ASCII collating sequence, **ORD-MAX("Z", "z", "I")** returns 2 because the ASCII character "z" occurs after "Z" and "I" in the program collating sequence. Each *char* argument is a group item, **USAGE DISPLAY** elementary item or alphanumeric literal

6.1.14.63. ORD-MIN(*char-1* [, *char-2*] ...)

This function returns the ordinal position in the argument list corresponding to the argument whose 1st character has the lowest position in the program collating sequence (usually ASCII). For example, assuming the program is using the standard ASCII collating sequence, **ORD-MIN("Z", "z", "I")** returns 3 because the ASCII character "I" occurs before "Z" and "z" in the program's collating sequence. Each *char* argument is a group item, **USAGE DISPLAY** elementary item or alphanumeric literal

6.1.14.64. PI

This function returns the mathematical constant "PI". The maximum precision with which this value may be returned is 3.1415926535897932384626433832795029.

Since the **PI** function has no arguments, no parenthesis should be specified.

6.1.14.65. PRESENT-VALUE(*rate*, *value-1* [, *value-2*])

The **PRESENT-VALUE** function returns a value that approximates the present value of a series of future period-end amounts specified by the various *value* arguments at a discount rate specified by the *rate* argument. All arguments are **PICTURE 9** items and/or numeric literals.

The following formula summarizes the functions operation:
$$result = \sum_{n=1}^{\# \text{ of values}} \frac{value_n}{(1+rate)^n}$$

6.1.14.66. RANDOM [(*seed*)]

The **RANDOM** function returns a non-integer value in the range 0 to 1 (for example, 0.123456789).

If *seed* is specified, it must be zero or a positive integer (specified as a PIC 9 item and/or numeric literal). It is used as the seed value to generate a sequence of pseudo-random numbers.

If a subsequent reference specifies *seed*, a new sequence of pseudo-random numbers is started.

If the first executed reference to this function does not specify a *seed*, the seed will be supplied by the compiler.

In each case, subsequent references without specifying a *seed* return the next number in the current sequence.

6.1.14.67. RANGE(number-1 [, number-2] ...)

The **RANGE** function returns a value that is equal to the value of the maximum *number* in the argument list minus the value of the minimum *number* argument. All arguments are numeric data items and/or numeric literals.

6.1.14.68. REM(number, divisor)

This function returns a numeric value that is the remainder of *number* divided by *divisor*. Both arguments must be numeric data items or numeric literals.

The result is determined according to the following formula:

$$number - (divisor * FUNCTION INTEGER-PART (number / divisor))$$

6.1.14.69. REVERSE(string)

This function returns the byte-by-byte reversed value of the specified *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal).

6.1.14.70. SECONDS-FROM-FORMATTED-TIME(format, time)

This function decodes a string whose value represents a formatted time and returns the total number of seconds that string represents. The time string must contain hours, minutes and seconds. The time argument may be specified as a group item, **USAGE DISPLAY** elementary item or an alphanumeric literal.

The *format* argument is a string (a group item, **USAGE DISPLAY** elementary item or an alphanumeric literal) documenting the format of *time* using "hh", "mm" and "ss" to denote where the respective time information can be found. Any other characters found in *format* represent character positions that will be ignored. For example, a *format* of "hhmmss" indicates that *time* will be treated as a six-digit value where the first two characters are the number of hours, the next two represent minutes and the last two represent seconds. Similarly, a *format* of "hh:mm:ss" states that *time* will be an eight-character string where characters 3 and 6 will be ignored.

6.1.14.71. SECONDS-PAST-MIDNIGHT

This function returns the current time of day expressed as the total number of elapsed seconds since midnight.

6.1.14.72. SIGN(number)

The **SIGN** function returns a -1 if the value of *number* (a numeric literal or data item) is negative, a zero if the value of *number* is exactly zero and a 1 if the value of *number* is greater than 0.

6.1.14.73. SIN(angle)

Determines and returns the trigonometric sine of the specified *angle* (a numeric literal or data item). The *angle* is assumed to be a value expressed in radians.

6.1.14.74. SQRT(number)

The **SQRT** function returns a numeric value that approximates the square root of *number* (a numeric data item or literal with a non-negative value).

6.1.14.75. STANDARD-DEVIATION(*number-1* [, *number-2*] ...)

This function returns the statistical standard deviation of the specified list *numbers* (these may be numeric data items or literals).

6.1.14.76. STORED-CHAR-LENGTH(*string*)

Returns the length – in bytes – of the specified *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal) minus the total number of trailing spaces, if any.

6.1.14.77. SUBSTITUTE(*string*, *from-1*, *to-1* [, *from-n*, *to-n*])

This function parses the specified *string*, replacing all occurrences of the *from-n* strings with the corresponding *to-n* strings. The *from* strings must match exactly with regard to value and case. The *from* strings do not have to be the same length as the *to* strings. All arguments are group items, **USAGE DISPLAY** elementary items or alphanumeric literals.

A null *to* string will be treated as a single **SPACE**.

6.1.14.78. SUBSTITUTE-CASE(*string*, *from-1*, *to-1* [, *from-n*, *to-n*])

The **SUBSTITUTE-CASE** function operates the same as the **SUBSTITUTE** function, except that *from* string matching is performed without regard for case. All arguments are group items, **USAGE DISPLAY** elementary items or alphanumeric literals.

6.1.14.79. SUM(*number-1* [, *number-2*] ...)

The **SUM** function returns a value that is the sum of the *number* arguments (these may be numeric data items or literals).

6.1.14.80. TAN(*angle*)

Determines and returns the trigonometric tangent of the specified angle (a numeric literal or data item). The *angle* is assumed to be a value expressed in radians.

6.1.14.81. TEST-DATE-YYYYMMDD(*date*)

Determines if the supplied *date* (a numeric integer data item or literal) is a valid date of the form *yyyymmdd* and that the date is in the range 1601/01/01 to 9999/12/31. If it is, a 0 value is returned. If it isn't, a value of 1, 2 or 3 is returned signaling the problem lies with the year, month or day, respectively.

6.1.14.82. TEST-DAY-YYYYDDD(*date*)

Determines if the supplied date (a numeric integer data item or literal) is a valid date of the form *yyyddd* and that the date is in the range 1601001 to 9999365. If it is, a 0 value is returned. If it isn't, a value of 1 or 2 is returned signaling the problem lies with the year or day, respectively.

6.1.14.83. TEST-NUMVAL(*string*)

The **TEST-NUMVAL** function evaluates the specified *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal) for being appropriate for use as the string argument to a **NUMVAL** function, returning a **TRUE** value if it is appropriate and **FALSE** otherwise.

See Also...

The **NUMVAL** Intrinsic Function [6.1.14.58](#)

6.1.14.84. TEST-NUMVAL-C(*string* [, *symbol*])

The **TEST-NUMVAL-C** function evaluates the specified *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal) and symbol combination for being appropriate for use as the arguments to a **NUMVAL-C** function, returning a TRUE value if they are appropriate and FALSE otherwise.

See Also...

The **NUMVAL-C** Intrinsic Function [6.1.14.59](#)

6.1.14.85. TEST-NUMVAL-F(*string*)

This function evaluates the specified *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal) for being appropriate for use as the string argument to a **NUMVAL-F** function, returning a TRUE value if it is appropriate and FALSE otherwise.

See Also...

The **NUMVAL-F** Intrinsic Function [6.1.7.60](#)

6.1.14.86. TRIM(*string*[, LEADING|TRAILING])

This function removes leading or trailing spaces from the specified *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal). The second argument is specified as a keyword, not a quoted string or identifier. If no second argument is specified, both leading and trailing spaces will be removed.

6.1.14.87. UPPER-CASE(*string*)

This function returns the value of *string* (a group item, **USAGE DISPLAY** elementary item or alphanumeric literal), converted entirely to upper case. Note that what constitutes a “letter” (or upper/lower case too, for that manner) may be influenced through the use of **CHARACTER CLASSIFICATION** specifications in the **OBJECT-COMPUTER** paragraph.

See Also...

The **OBJECT-COMPUTER** Paragraph [4.1.2](#)

6.1.14.88. VARIANCE(*number-1* [, *number-2*] ...)

This function returns the statistical variance of the specified list *numbers* (these may be numeric data items or literals).

6.1.14.89. YEAR-TO-YYYY (*yy* [, *yy-cutoff*])

YEAR-TO-YYYY converts *yy* (a) - a two-digit year - to a four-digit format (*yyyy*). The optional *yy-cutoff* argument is the year cutoff used to delineate centuries; if *yy* meets or exceeds this cutoff value, the result will be 19*yy*; if *yy* is less than the cutoff, the result will be 20*yy*. The default cutoff value if no second argument is given will be 50. Both arguments must be numeric data items or literals.

6.2. GNU COBOL Statements

The remaining sections in this chapter present (in alphabetical order) the various verbs (statements) that make up the GNU COBOL procedural language.

6.2.1. ACCEPT

6.2.1.1. ACCEPT Format 1 – Read from Console

Figure 6-20 - ACCEPT (Read from Console) Syntax

```

ACCEPT identifier-1
  [ FROM mnemonic-name-1 ]
  [ END-ACCEPT ]

```

This format of the **ACCEPT** verb is used to read a value from the console window or the standard input device and store it into a data item (*identifier-1*).

1. *Mnemonic-name-1* must either be the built-in device name **CONSOLE**, **STDIN**, **SYSIN** or **SYSIPT** or a user-defined (**SPECIAL-NAMES**) mnemonic name attached to one of those four device names.
2. If no **FROM** clause is specified, **FROM CONSOLE** is assumed.
3. Input will be read either from the console window (**CONSOLE**) or from the system-standard input (pipe 0 = **STDIN**, **SYSIN** or **SYSIPT**) and will be saved in *identifier-1*.
4. If *identifier-1* is a numeric data item, the character value read from the console or standard-input device will be parsed according to the rules for "Format 1" input to the **NUMVAL** intrinsic function.

See Also...

The **SPECIAL-NAMES** Paragraph [4.1.4](#)

The **NUMVAL** Intrinsic Function [6.1.14.58](#)

6.2.1.2. ACCEPT Format 2 – Retrieve Command-Line Arguments

Figure 6-21 - ACCEPT (Command Line Arguments) Syntax

```

ACCEPT identifier-1
  FROM { COMMAND-LINE
          ARGUMENT-NUMBER
          ARGUMENT-VALUE [ exception-handler ] }
  [ END-ACCEPT ]

```

This format of the **ACCEPT** verb is used to retrieve information from the programs command-line.

1. When you accept from the **COMMAND-LINE** option, you will retrieve the entire set of arguments entered on the command line that executed the program, exactly as they were specified. Parsing that returned data into its meaningful information will be your responsibility.
2. By accepting from **ARGUMENT-NUMBER**, you will be asking the GNU COBOL run-time system to parse the arguments from the command-line and return the number of arguments found. Parsing will be conducted according to the operating system's rules, as follows:
 - ▶ Arguments will be separated by treating SPACES between characters as the delineators between arguments. The number of spaces separating two non-blank values is irrelevant.
 - ▶ Strings enclosed in double-quote characters (") will be treated as a single argument, regardless of how many spaces (if any) might be imbedded within those quotation characters.
 - ▶ On Windows systems, single-quote, or apostrophe characters (') will be treated just like any other data character and will NOT delineate strings.
3. By accepting from **ARGUMENT-VALUE**, you will be asking the GNU COBOL run-time system to parse the arguments from the command-line and return the "current" argument. You specify which argument number is "current" via the **DISPLAY ... UPON ARGUMENT-NUMBER** statement (section 0). Parsing or arguments will be conducted according to the rules set forth in #2 above.
4. Attempts to retrieve non-existent arguments can be handled via an optional *exception-handler*.

See Also...

Handling Exceptions (**ON EXCEPTION**) [6.1.12.4](#)

The **DISPLAY** Statement (Command Line) [6.2.12.2](#)

6.2.1.3. ACCEPT Format 3 – Retrieve Environment Variable Values

Figure 6-22 - ACCEPT (Environment Variable Values) Syntax

```

ACCEPT identifier-1
      FROM { ENVIRONMENT-VALUE
             { ENVIRONMENT { literal-1
                              { identifier-2 } } }
             [ exception-handler ]
      [ END-ACCEPT ]

```

This format of the **ACCEPT** verb is used to retrieve environment variable values.

1. By accepting from **ENVIRONMENT-VALUE**, you will be asking the GNU COBOL run-time system to retrieve the value of the environment variable whose name is currently in the **ENVIRONMENT-NAME** register. A value may be placed into the **ENVIRONMENT-NAME** register using the **DISPLAY** statement.
2. A simpler approach to retrieving an environment variables value is to use “**ACCEPT ... FROM ENVIRONMENT**”. Using that form, you specify the environment variable to be retrieved right on the **ACCEPT** command itself.
3. The optional *exception-handler* may be used to detect requests to retrieve the values of non-existent environment variables..

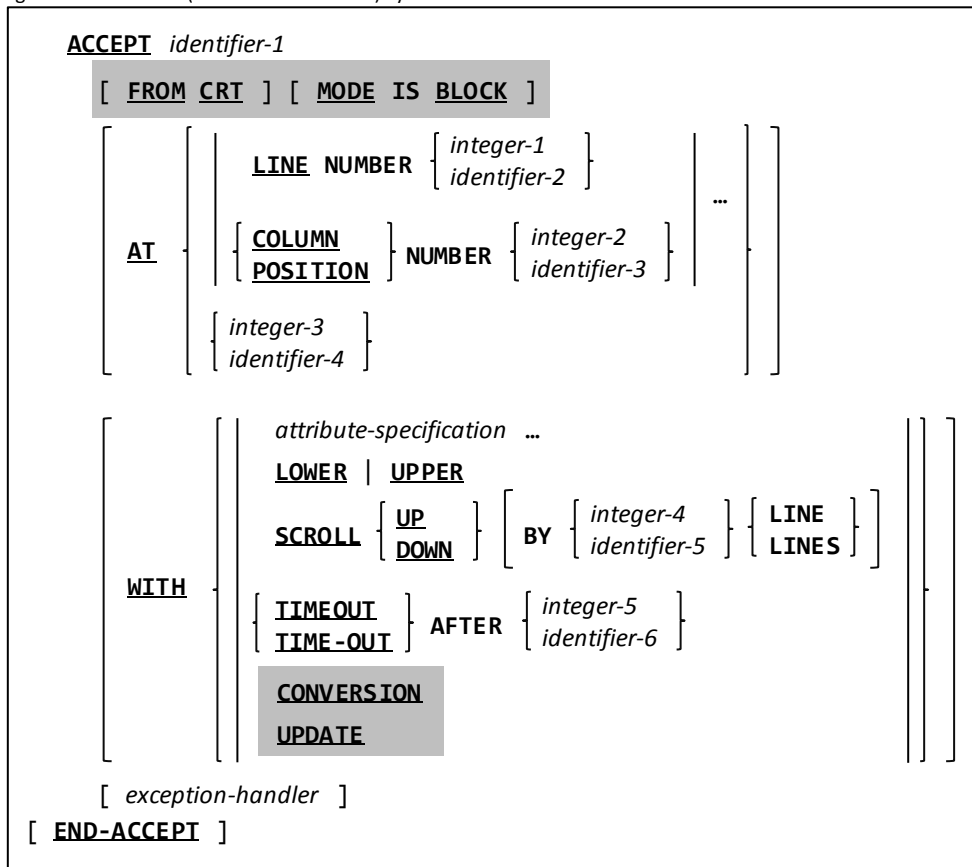
See Also...

Handling Exceptions (**ON EXCEPTION**) [6.1.12.4](#)

The **DISPLAY** Statement (Environment) [6.2.12.3](#)

6.2.1.4. ACCEPT Format 4 – Retrieve Full-Screen Data

Figure 6-23 - ACCEPT (Retrieve Screen Data) Syntax



This format of the **ACCEPT** verb is used to retrieve data from a formatted console window screen.

1. The following *attribute-specification* clauses are allowed on the **ACCEPT** statement – these are the same as those allowed for **SCREEN SECTION** data items.

AUTO AUTO-SKIP AUTOTERMINATE	FULL LENGTH-CHECK	REQUIRED EMPTY-CHECK
BACKGROUND-COLOR	HIGHLIGHT LOWLIGHT	REVERSE-VIDEO
BEEP BELL	LEFTLINE	SECURE NO-ECHO
BLINK	OVERLINE	UNDERLINE
BACKGROUND-COLOR	PROMPT CHARACTER	

2. If *identifier-1* is defined in the **SCREEN SECTION**, any **AT**, *attribute-specification* **LOWER**, **UPPER** or **SCROLL** clauses specified on the **ACCEPT** will be ignored.
3. The various **AT** clauses provide a means of positioning the cursor to a specific spot on the screen before the screen is read. The *literal-3* / *identifier-4* value must be a four- or six-digit value with the 1st half of the number indicating the line where the cursor should be positioned and the second half indicating the column. There is no distinction between using the word **COLUMN** or **POSITION**.
4. **WITH** options (including the various individual *attribute-specifications*) should be coded only once.
5. The **SCROLL** option will cause the entire contents of the screen to be scrolled **UP** or **DOWN** by the specified number of lines before any value is displayed on the screen. It is possible to specify a **SCROLL UP** clause as well as a **SCROLL DOWN** clause. If no **LINES** specification is made, "1 LINE" will be assumed.
6. The **TIMEOUT** option will cause the **ACCEPT** to wait no more than the specified number of seconds for input. The wait count may be specified as a positive integer or a numeric data item with a positive value. Once the timeout limit expires, **ACCEPT** will proceed as if the Enter key had been pressed with no data being entered. The keyword **TIME-OUT** may be used as a synonym for **TIMEOUT**.
7. While supported syntactically, the **CONVERSION** and **UPDATE** options are non-functional.
8. When a Format 4 **ACCEPT** statement with a **SCREEN SECTION** item specified as *identifier-1* is executed, an implied **DISPLAY** of *identifier-1* will occur before input is accepted. Coding an explicit "**DISPLAY identifier-1**" before an "**ACCEPT identifier-1**" is redundant and will incur the performance penalty of painting the screen contents twice.

9. The optional *exception-handler* may be used to handle screen I/O errors.
10. After this format of the **ACCEPT** statement is executed, the programs **CRT STATUS** code identifier (section [4.1.4](#)) will be populated with one of the following:

Figure 6-24 - Screen ACCEPT CRT STATUS Codes

Code	Meaning	Code	Meaning
0000	ENTER key pressed	2005	Esc ²⁷
1001 - 1064	F1 – F64	8000	No data is available on screen ACCEPT
2001,2002	PgUp,PgDn ²⁵	9000	Fatal screen I/O error
2003,2004,2006	Up Arrow,Down-Arrow,PrtSc (Print Screen) ²⁶		

This value will indicate what special key was pressed to terminate the **ACCEPT**.

The actual key pressed to generate a function key (Fn) will depend on the type of terminal device you're using (PC, Macintosh, VT100, etc.) and what type of enhanced display driver was configured with the version of GNU COBOL you're using. For example, on a GNU COBOL built for a Windows PC using MinGW and PDCurses, F1-F12 are the actual F-keys on the PC keyboard, F13-F24 are entered by shifting the F-keys, F25-F36 are entered by holding Ctrl while pressing an F-key and F37-F48 are entered by holding Alt while pressing an F-key. On the other hand, a GNU COBOL implementation built for Windows using Cygwin and NCurses treats the PCs F1-F12 keys as the actual F1-F12, while shifted F-keys will enter F11-F20. With Cygwin/NCurses, Ctrl- and Alt-modified F-keys aren't recognized. Neither are Shift-F11 or Shift-F12.

Numeric keypad keys are not recognizable on Windows MinGW/PDCurses builds of GNU COBOL, regardless of NumLock settings. Windows Cygwin/NCurses builds recognize numeric keypad inputs properly. Although not tested during the preparation of this documentation, I would expect native Windows builds using PDCurses to behave as MinGW builds do and native Unix builds using NCurses to behave as do Cygwin builds.

The **CRT STATUS** field the status code is saved into will be either **COB-CRT-STATUS**, if the CRT STATUS clause was not specified in the **SPECIAL-NAMES** paragraph, or the programmer-specified identifier if that clause was specified in **SPECIAL-NAMES**.

See Also...

Defining Screens [5.2.2](#)

Handling Exceptions (**ON EXCEPTION**) [6.1.12.4](#)

6.2.1.5. ACCEPT Format 5 – Retrieve Date/Time

Figure 6-25 - ACCEPT (Retrieve Date/Time) Syntax

```

ACCEPT identifier-1
      { DATE [ YYYYMMDD ]
        FROM { DAY [ YYYYDDD ]
                 DAY-OF-WEEK
                 TIME
              }
      [ END-ACCEPT ]
    
```

This format of the **ACCEPT** verb is used to retrieve the current system date, time or current day of the week and store it into a data item.

1. The data retrieved from the system, and the format in which it is structured, will vary according to the following chart:

Figure 6-26 - ACCEPT Options for DATE/TIME Retrieval

ACCEPT Option	Data Returned	<i>identifier-1</i> Format
DATE	Current date in Gregorian form (two-digit year)	01 CURRENT-DATE. 05 CD-YEAR PIC 9(2). 05 CD-MONTH PIC 9(2). 05 CD-DAY-OF-MONTH PIC 9(2).
DATE YYYYMMDD	Current date in Gregorian form (four-digit year)	01 CURRENT-DATE. 05 CD-YEAR PIC 9(4). 05 CD-MONTH PIC 9(2). 05 CD-DAY-OF-MONTH PIC 9(2).

²⁵ These keys are available only if the environment variable COB_SCREEN_EXCEPTIONS is set to any non-blank value at runtime.

²⁶ These keys are not detectable on Windows systems

²⁷ This key is available only if the environment variable COB_SCREEN_ESC is set to any non-blank value at runtime (this is in addition to setting COB_SCREEN_EXCEPTIONS)

ACCEPT Option	Data Returned	<i>identifier-1</i> Format
DAY	Current date in Julian form (two-digit year)	01 CURRENT-DATE. 05 CD-YEAR PIC 9(2). 05 CD-DAY-OF-YEAR PIC 9(3).
DAY YYYYDD	Current date in Julian form (four-digit year)	01 CURRENT-DATE. 05 CD-YEAR PIC 9(4). 05 CD-DAY-OF-YEAR PIC 9(3).
DAY-OF-WEEK	Current day of the week	01 CURRENT-DATE. 05 CD-DAY-OF-WEEK PIC 9(1). 88 MONDAY VALUE 1. 88 TUESDAY VALUE 2. 88 WEDNESDAY VALUE 3. 88 THURSDAY VALUE 4. 88 FRIDAY VALUE 5. 88 SATURDAY VALUE 6. 88 SUNDAY VALUE 7.
TIME	Current time	01 CURRENT-TIME. 05 CT-HOURS PIC 9(2). 05 CT-MINUTES PIC 9(2). 05 CT-SECONDS PIC 9(2). 06 CT-HUNDREDTHS-OF-SECS PIC 9(2).

6.2.1.6. ACCEPT Format 6 - Retrieve Screen Information

Figure 6-27 - ACCEPT (Retrieve Screen Information) Syntax

ACCEPT	<i>identifier-1</i>
FROM	{ LINES LINE-NUMBER COLUMNS COLS ESCAPE KEY }
[END-ACCEPT]	

This format of the **ACCEPT** verb is used to retrieve information about the console window or about the user's interactions with it.

1. The **LINES** and **COLUMNS** options will retrieve the respective components of the size of the console display. When the console is running in a windowed environment, this will be the sizing of the window in which the program is executing, in terms of horizontal (**COLUMNS**) or vertical (**LINES**) character counts – not pixels. When the system is not running a windowing environment, the physical console screen attributes will be returned. In environments such as a Windows console window, where the logical size of the window may far exceed that of the physical console window, the size returned will be that of the physical console window. If necessary, the screen will be initialized so that the screen window size may be determined. Values of 0 will be returned if GNU COBOL was not generated to include screen I/O.. Compare this result with that of the **CBL_GET_SCR_SIZE** built-in subroutine.
2. The **LINE NUMBER** option is a synonym for **LINES** and the word **COLUMNS** may be specified as **COLS**.
3. The **ESCAPE KEY** option may be used after a format 4 **ACCEPT** has been used to retrieve data off a formatted screen. The result returned will be the four-digit key id of the special key that was pressed to terminate the format 4 **ACCEPT** (a 0000 is returned for the Enter key). This value will be the same as that returned into the **CRT STATUS** field defined in the **SPECIAL-NAMES** paragraph or into the **COB-CRT-STATUS** identifier if no **CRT STATUS** was specified. Consult [Figure 6-23](#) for a list of possible values.

See Also...

The **SPECIAL-NAMES** Paragraph [4.1.4](#)

The **CBL_GET_SCR_SIZE** Subroutine [8.3.1.30](#)

6.2.1.7. ACCEPT Format 7 – Retrieve Run-Time Information

Figure 6-28 - ACCEPT (Retrieve Run-Time Information) Syntax

ACCEPT	<i>identifier-1</i>
FROM	{ EXCEPTION STATUS USER NAME }
[END-ACCEPT]	

This format of the **ACCEPT** verb is used to retrieve run-time information such as the most-recent error exception code and the current user's user name.

1. The specified identifier must be defined as a **PIC X(4)** item to receive **EXCEPTION STATUS**. When receiving **USER NAME**, the identifier should be large enough to receive the longest user name on your system. If insufficient space is allocated, the returned value will be truncated. If excess space is allocated, the returned value will be padded with **SPACES** (to the right).

2. The most-recently encountered runtime error status will be returned in the identifier ('0000' if no error has occurred) when issuing an **ACCEPT ... FROM EXCEPTION STATUS**.
3. The following table summarizes the current run-time error exception codes.

Figure 6-29 - Run-Time Exception Code Values

Exception Code Returned to ACCEPT	Error Type String Returned by the EXCEPTION-STATUS Function	Description
0101	EC-ARGUMENT-FUNCTION	Function argument error
0202	EC-BOUND-ODO	OCCURS ... DEPENDING ON data item out of bounds
0204	EC-BOUND-PTR	Data-pointer contains an address that is out of bounds
0205	EC-BOUND-REF-MOD	Reference modifier out of bounds
0207	EC-BOUND-SUBSCRIPT	Subscript out of bounds
0303	EC-DATA-INCOMPATIBLE	Incompatible data exception
0500	EC-I-O	input-output exception
0501	EC-I-O-AT-END	I-O status "1x"
0502	EC-I-O-EOP	An end of page condition occurred
0504	EC-I-O-FILE-SHARING	I-O status "6x"
0505	EC-I-O-IMP	I-O status "9x"
0506	EC-I-O-INVALID-KEY	I-O status "2x"
0508	EC-I-O-LOGIC-ERROR	I-O status "4x"
0509	EC-I-O-PERMANENT-ERROR	I-O status "3x"
050A	EC-I-O-RECORD-OPERATION	I-O status "5x"
0601	EC-IMP-ACCEPT	Implementation-defined accept condition
0602	EC-IMP-DISPLAY	Implementation-defined display condition
0A00	EC-OVERFLOW	Overflow condition
0A02	EC-OVERFLOW-STRING	STRING overflow condition
0A03	EC-OVERFLOW-UNSTRING	UNSTRING overflow condition
0B05	EC-PROGRAM-NOT-FOUND	Called program not found
0D03	EC-RANGE-INSPECT-SIZE	Size of replace item in inspect differs
1000	EC-SIZE	Size error exception
1004	EC-SIZE-OVERFLOW	Arithmetic overflow in calculation
1005	EC-SIZE-TRUNCATION	Significant digits truncated in store
1007	EC-SIZE-ZERO-DIVIDE	Division by zero
1202	EC-STORAGE-NOT-ALLOC	The data-pointer specified in a FREE statement does not identify currently allocated storage
1203	EC-STORAGE-NOT-AVAIL	The amount of storage requested by an ALLOCATE statement is not available

4. When using **ACCEPT ... FROM USER NAME**, the returned result is the userid that was used to login to the system with, and not any actual first and/or last name of the user in question (unless, of course, that is the information used as a logon id).

6.2.2. ADD

6.2.2.1. ADD Format 1 – ADD TO

Figure 6-30 - ADD (TO) Syntax

```

ADD { { literal-1
         identifier-1 } } ...
         TO { identifier-2 [ rounding-option ] } ...
         [ size-error-clause ]
[ END-ADD ]

```

This format of the **ADD** statement generates the arithmetic sum of all arguments that appear before the **TO** (*identifier-1* or *literal-1*) and then adds that sum to each of the identifiers listed after the **TO** (*identifier-2*).

1. *Identifier-1* and *identifier-2* must be numeric unedited data items while *literal-1* must be a numeric literal.
2. The value(s) specified before the “**TO**” keyword will be added together, and that sum will be added onto each of the identifiers specified after the “**TO**” keyword (*identifier-2*), in turn.
3. The optional “*rounding-option*” clause available to each *identifier-2* will control how non-integer results will be saved.
4. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-2* is insufficiently sized to hold the generated results.

See Also...

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

Rounding Options [6.1.12.7](#)

6.2.2.2. ADD Format 2 – ADD GIVING

Figure 6-31 - ADD (GIVING) Syntax

```

ADD { { literal-1
         identifier-1 } } ...
         [ TO identifier-2 ]
         GIVING { identifier-3 [ rounding-option ] } ...
         [ size-error-clause ]
[ END-ADD ]

```

This format of the **ADD** statement generates the arithmetic sum of all arguments that appear before the **TO** (*identifier-1* or *literal-1*), adds that sum to the contents of *identifier-2* (if any) and then replaces the contents of the identifiers listed after the **GIVING** (*identifier-3*) with that sum.

1. *Identifier-1* and *identifier-2* must be numeric unedited data items, *identifier-3* must be a numeric (edited or unedited) data item and *literal-1* must be a numeric literal.
2. The value(s) specified before the “**TO**” keyword will be added together, and that sum will be added to the value of *identifier-2* (if any). The contents of *identifier-2* are not altered. The resulting sum is then saved to each of the identifiers specified after the “**GIVING**” keyword (*identifier-3*), in turn. Unless also specified as one of the *identifier-1* items or as the *identifier-2* item, none of the *identifier-3* items will be involved in the calculation other than simply serving as the receiving field(s) of the operation.
3. The optional “*rounding-option*” clause available to each *identifier-3* will control how non-integer results will be saved.
4. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-2* is insufficiently sized to hold the generated results.

See Also...

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

Rounding Options [6.1.12.7](#)

6.2.2.3. ADD Format 3 – ADD CORRESPONDING

Figure 6-32 - ADD (CORRESPONDING) Syntax

```

ADD CORRESPONDING identifier-1
  TO identifier-2 [ rounding-option ]
  [ size-error-clause ]
[ END-ADD ]

```

This format of the **ADD** statement generates code equivalent to individual **ADD TO** statements for corresponding matches of data items found subordinate to the two identifiers.

1. When corresponding matches are established, the effect of an **ADD CORRESPONDING** on those matches will be as if a series of individual **ADD** Format 1 statements were done – one for each match.
2. The optional “*rounding-option*” clause available to each *identifier-2* will control how non-integer results will be saved.
3. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-2* is insufficiently sized to hold the generated results.

See Also...

The **CORRESPONDING** Clause [6.1.12.2](#)

Rounding Options [6.1.12.7](#)

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

6.2.3. ALLOCATE

Figure 6-33 - ALLOCATE Syntax

```

ALLOCATE { expression-1 CHARACTERS }
           [ identifier-1 ]
           [ INITIALIZED ]
           [ RETURNING identifier-2 ]

```

The **ALLOCATE** statement is used to dynamically allocate memory at run-time.

1. If used, *expression-1* must be an arithmetic expression with a non-zero positive integer value.
2. If used, *identifier-1* should be an 01-level item defined with the **BASED** attribute in **WORKING-STORAGE** or **LOCAL-STORAGE**. It can be an 01 item defined in the **LINKAGE SECTION** without the **BASED** option, but using such a data item is not recommended.
3. If used, *identifier-2* should be a **USAGE POINTER** data item.
4. The optional **RETURNING** clause will return the address of the allocated memory block into the specified **USAGE POINTER** item. When this option is used, GNU COBOL will retain knowledge of the originally-requested size of the allocated memory block in case a **FREE** statement is ever issued against that **USAGE POINTER** item.
5. When the “*identifier-1*” option is used in conjunction with **INITIALIZED**, the allocated memory block will be initialized according to the **PICTURE** and (if any) **VALUE** clauses present in the definition of *identifier-1* as if an **INITIALIZE *identifier-1* WITH FILLER ALL TO VALUE THEN TO DEFAULT** were executed once *identifier-1* was allocated.
6. When the “*expression-1* **CHARACTERS**” option is used, **INITIALIZED** will initialize the allocated memory block to binary zeros.
7. If the **INITIALIZED** clause is not used, the initial contents of allocated memory will be left to whatever rules of memory allocation are in effect for the operating system the program is running under.
8. There are two basic ways in which this statement is used. The simplest is:

```
ALLOCATE My-01-Item
```

With this form, a block of storage equal in size to the defined size of **My-01-Item** (which must have been defined with the **BASED** attribute) will be allocated. The address of that block of storage will become the base address of **My-01-Item** so that it and its subordinate data items become usable within the program.

A second (and equivalent) approach is:

```
ALLOCATE LENGTH OF My-01-Item CHARACTERS RETURNING The-Pointer.
SET ADDRESS OF My-01-Item TO The-Pointer.
```

9. Referencing a **BASED** data item either before its storage has been **ALLOCATED** or after its storage has been **FREED** will lead to unpredictable results²⁸.

See Also...

The DATA DIVISION	5	The FREE Statement	6.4.17
Dynamically Allocated Items (BASED)	5.2.1.2	The INITIALIZE Statement	6.2.22
Storage Format of Data (USAGE)	5.2.1.11		

²⁸ The COBOL standards like to use the term “unpredictable results” to indicate any sort of unexpected or undesirable behavior – the results in this case probably are predictable though – the program will probably abort from attempting to access an invalid address.

6.2.4. ALTER

Figure 6-34 - ALTER Syntax

```
ALTER procedure-name-1 TO PROCEED TO procedure-name-2
```

The **ALTER** verb was used in the early years of the COBOL language to edit a program, changing a

“**GO TO**” statement ***at run time*** to branch to a spot in the program different than where the **GO TO** statement was originally compiled for.

1. Support for the **ALTER** verb has been added to GNU COBOL for the purpose of enabling GNU COBOL to pass those National Institute of Standards and Technology (NIST) tests for the COBOL programming language that require support for the **ALTER** verb.
2. Use of this statement is **STRONGLY** discouraged because it's use makes it extremely difficult to know where a potentially ALTER-able **GO TO** statement is actually going to at run time.

6.2.5. CALL

Figure 6-35 - CALL Syntax

```

CALL [ { STDCALL
        STATIC
        mnemonic-name-1 } ] { literal-1
                               identifier-1 }

[ USING argument-1 ... ]
[ RETURNING | GIVING identifier-2 ]
[ overflow-clause | exception-clause ]
[ END-CALL ]

```

The **CALL** statement is used to transfer control to a sub-program, called a *subroutine*.

Chapter 7 deals with the specifics of using subprograms with GNU COBOL programs.

1. The expectation is that the subroutine will eventually return control back to the **CALLING** program, at which point the **CALLING** program will resume execution starting with the statement immediately following the **CALL**. Subprograms are not required to return to their **CALLERS**, however, and are free to halt program execution if they wish.
2. The *mnemonic-name-1* / **STATIC** / **STDCALL** option, if used, affects the linkage conventions that will be used to the subroutine being called, as follows:
 - a. The **STATIC** option will cause the linkage to the subroutine to be performed in such a way as to require the subroutine to be statically-linked with the calling program. Note that this enables static-linking to be used on a subroutine-by-subroutine selective basis.
 - b. The **STDCALL** option allows system-standard calling conventions (as opposed to GNU COBOL calling conventions) to be used when calling a subroutine. The definition of what constitutes “system standard” may vary from operating system to operating system. Use of this requires special knowledge about the linkage requirements of subroutines you are intending to **CALL**. Subroutines written in GNU COBOL do not need this option.
 - c. The *mnemonic-name* option allows a custom-defined calling convention to be used. Such mnemonic names are defined using the **CALL-CONVENTION** clause of the **SPECIAL-NAMES** paragraph. That clause associates a decimal integer value with *mnemonic-name-1* such that the individual bits set on or off in the binary number corresponding to the integer affect linkage to the subroutine as described in the following chart. Those rows of the chart that are greyed-out represent bit positions (switch settings) in the integer value that are currently accepted if (to provide compatibility to other COBOL implementations) coded, but are otherwise unsupported.

Bit Position	Decimal Value If 1	Meaning if 0	Meaning if 1
0 (right-most)	1	Subroutine arguments will be processed in right-to-left sequence	Subroutine arguments will be passed in left-to-right sequence
1	2	The calling program will flush processed arguments from the argument stack	The called program (subroutine) will flush processed arguments from the argument stack
2	4	The RETURN-CODE register will be updated in addition to any RETURNING/GIVING data item	The RETURN-CODE register will not be updated (but any RETURNING/GIVING data item still will)
3	8	If CALL “literal” is used, the subroutine will be located and linked in with the calling program at compile time or may be dynamically located and loaded at execution time, depending on compiler switch settings and operating system capabilities.	If CALL “literal” is used, the subroutine can only be located and linked with the calling program at compilation time.

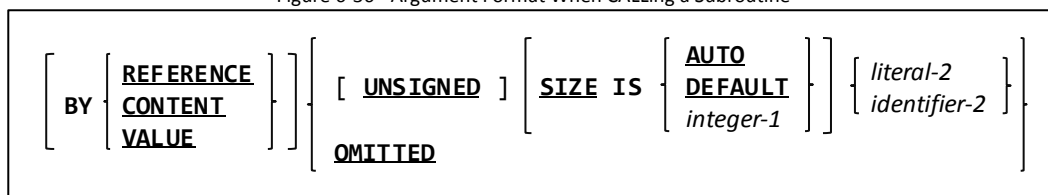
Bit Position	Decimal Value If 1	Meaning if 0	Meaning if 1
4	16	OS/2 "OPTLINK" conventions will <u>not</u> be used to CALL the subprogram.	OS/2 "OPTLINK" conventions will be used to CALL the subprogram.
5	32	Windows 16-bit "thunking" will not be in effect.	Windows 16-bit "thunking" will be used to CALL the subroutine as a DLL.
6	64	The STDCALL convention will not be used.	The STDCALL convention will be used. ²⁹

Using the "**STDCALL**" option on a **CALL** statement is equivalent to using a **CALL-CONVENTION "8"** (only bit 3 set)

Using the "**STATIC**" option on a **CALL** statement is equivalent to using a **CALL CONVENTION 64** (only bit 6 set)

3. The **RETURNING** and **GIVING** keywords may be used interchangeably.
4. The value of *literal-1* or *identifier-1* is the entry-point of the subprogram you wish to **CALL**.
5. When you **CALL** a subroutine using *identifier-1*, you are forcing the runtime system to call a dynamically-loadable module. The contents of *identifier-1* will be the entry-point name within that module. If this is the first **CALL** to any entry-point within the module, the contents of *identifier-1* must be the name of the module itself (making it the primary entry-point name within the module).
6. If the subprogram being called is a GNU COBOL program, and if that program had the **INITIAL** attribute specified on its **PROGRAM-ID** clause, all of the subprogram's **DATA DIVISION** data will be restored to its initial state each time the subprogram is executed³⁰. This [re]-initialization behavior will always apply to any data defined in the subprogram's **LOCAL-STORAGE SECTION** (if any), regardless of the use (or not) of **INITIAL**.
7. The **USING** clause defines a list of arguments that may be passed from the calling program to the subprogram. The syntax used to specify an argument is as follows:

Figure 6-36 - Argument Format When CALLing a Subroutine



8. The manner in which an argument is passed to the subroutine depends upon its **BY** clause, if any, specified for the arguments, as follows:
 - a. **BY REFERENCE** passes the address of the argument to the subroutine. If the subroutine changes the contents of that argument, the change will be "visible" to the calling program.
 - b. **BY CONTENT** passes the address of a copy of the argument to the subroutine. If the subroutine changes the value of such an argument, the original version of it back in the calling program remains unchanged.
 - c. **BY VALUE** passes the value of the argument as the argument. This feature exists to provide compatibility with C, C++ and other languages and would not normally be used when calling GNU COBOL subprograms.
 - d. If an argument lacks a **BY REFERENCE**, **BY CONTENT** or **BY VALUE** clause, the most-recently encountered "**BY**" specification on that **CALL** statement will be assumed (or **BY REFERENCE** if there have been no "**BY**" specifications specified yet).
 - e. No more than 36 arguments may be passed to a subroutine, unless the GNU COBOL compiler was built with a specifically different argument limit specified for it..

²⁹ The **STDCALL** calling convention is the one required to use the Microsoft Win32 API

³⁰ This is regardless of which entry-point within the subprogram is **CALLED**

9. The **RETURNING** clause allows you to specify a data item into which the subroutine should return a value. If you use this clause on the **CALL**, the subroutine should include a **RETURNING** clause on its **PROCEDURE DIVISION** header. Of course, a subroutine may pass a value back in any argument passed **BY REFERENCE**.
10. The optional *overflow-clause* or *exception-clause* (the two may be used interchangeably) may be used to define actions to be taken if the subroutine could not be located and/or loaded.
11. For additional information, see the documentation of the **CANCEL**, **ENTRY**, **EXIT PROGRAM** and **GOBACK** statements.

See Also...

The IDENTIFICATION DIVISION	3
The SPECIAL-NAMES Paragraph	4.1.4
The DATA DIVISION	5
Special Registers	6.1.13
Handling Exceptions (ON EXCEPTION)	6.1.12.4
Handling Overflow (ON OVERFLOW)	6.1.12.5
The CANCEL Statement	6.2.6

The ENTRY Statement	6.2.14
The EXIT PROGRAM Statement	6.2.16
The GOBACK Statement	6.2.19
Sub-programming	7
Compiling & Dynamic-Linking Programs	8.1.3.2
Compiling & Static-Linking Programs	8.1.3.3

6.2.6. CANCEL

Figure 6-37 - CANCEL Syntax

```
CANCEL { literal-1 } ...  
         { identifier-1 }
```

The **CANCEL** statement unloads the dynamically-loadable module containing the entry-point specified as *literal-1* or *identifier-1* from memory.

1. If the dynamically-loadable module unloaded by the **CANCEL** is subsequently re-executed, all **DATA DIVISION** storage for that dynamically-loadable module will once again be in its initial state.

See Also...

Sub-programming 7

Compiling & Dynamic-Linking Programs [8.1.3.2](#)

6.2.7. CLOSE

Figure 6-38 - CLOSE Syntax

```
CLOSE { file-name-1 [ [ REEL | UNIT [ FOR REMOVAL ] ] ]
                [ WITH LOCK ] ]
                [ WITH NO REWIND ] ] } ...
```

The **CLOSE** statement terminates the program's access to the specified file(s).

1. The **CLOSE** statement may only be executed against files that have been successfully **OPENed**.
2. The **REEL**, **UNIT**, **WITH LOCK** and **NO REWIND** clauses are recognized syntactically but are otherwise non-functional except for the fact that a successful **CLOSE ... NO REWIND** will generate a **FILE-STATUS** value of 07 rather than 00.
3. A successful **CLOSE** will write any remaining unwritten record buffers to the file (similar to an **UNLOCK**) and release any file locks for the file; regardless of **OPEN** mode. A closed file will then be no longer available for subsequent I/O statements until it is once again **OPENed**.
4. When a **LINE SEQUENTIAL** or **LINE ADVANCING** file is **CLOSEd**, a final delimiter sequence will be written to the file to signal the termination point of the final data record in the file. This will only be necessary if the final record written to the file was written with the **AFTER ADVANCING** option.

See Also...

Types of Files	1.3.3.5	The OPEN Statement	6.4.29
FILE-STATUS Values	Figure 4-15	The UNLOCK Statement	6.4.48
		The WRITE Statement	6.4.50

6.2.8. COMMIT

Figure 6-39 - COMMIT Syntax

COMMIT

The **COMMIT** statement performs an **UNLOCK** against every currently-**OPEN** file, but does NOT **CLOSE** any of the files.

1. See the **UNLOCK** statement for additional details.

See Also...

The **CLOSE** Statement [6.4.7](#)

The **UNLOCK** Statement [6.4.48](#)

6.2.9. COMPUTE

Figure 6-40 - COMPUTE Syntax

```

COMPUTE { identifier-1 [ rounding-option ] } ... =|EQUAL arithmetic-expression-1
[ size-error-clause ]
[ END-COMPUTE ]

```

The **COMPUTE** statement provides a means of easily performing complex arithmetic operations with a single statement, instead of using cumbersome and possibly confusing sequences of **ADD**, **SUBTRACT**, **MULTIPLY** and **DIVIDE** statements.

1. Each *identifier-1* must be a numeric or numeric-edited data item.
2. The word **EQUAL** and the equals-sign (=) may be used interchangeably.
3. The optional "*rounding-option*" clause available to each *identifier-1* will control how non-integer results will be saved.
4. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-1* is insufficiently sized to hold the generated results.

See Also...

Handling Size Errors (ON SIZE ERROR)	6.1.12.6	The DIVIDE Statement	6.4.13
Rounding Options	6.1.12.7	The MULTIPLY Statement	6.4.27
The ADD Statement	6.4.2	The SUBTRACT Statement	6.4.44

6.2.10. CONTINUE

Figure 6-41 - CONTINUE Syntax

CONTINUE

The **CONTINUE** statement is a no-operation statement, performing no action whatsoever.

- The **CONTINUE** statement is often used with **IF** statements as a place-holder for conditionally-executed code that is not yet needed or not yet designed. The following two sentences are equivalent. One uses **CONTINUE** statements to mark places where code may need to be inserted in the future.

“Minimalist” Coding (Specifying only what is necessary)	Coding With CONTINUE (Documenting where code might be needed someday)
<pre> IF A = 1 IF B = 1 DISPLAY 'A=1 & B=1' END-DISPLAY END-IF ELSE IF A = 2 IF B = 2 DISPLAY 'A=2 & B=2' END-DISPLAY END-IF END-IF END-IF </pre>	<pre> IF A = 1 IF B = 1 DISPLAY 'A=1 & B=1' END-DISPLAY ELSE CONTINUE END-IF ELSE IF A = 2 IF B = 2 DISPLAY 'A=2 & B=2' END-DISPLAY ELSE CONTINUE END-IF ELSE CONTINUE END-IF END-IF </pre>

Coding such as this is generally a matter of personal preference or site coding standards. There is no difference in the object code generated by the two, so there isn't a run-time efficiency issue (just one of "coding efficiency").

- Another **IF**-statement usage for **CONTINUE** is to avoid the use of **NOT** in the conditional expression coded on the **IF** statement. This too is a personal and/or site standards issue. Here's an example:

Without CONTINUE	With CONTINUE
<pre> IF Action-Flag NOT = 'I' AND 'U' DISPLAY 'Invalid Action-Flag' EXIT PARAGRAPH END-IF </pre>	<pre> IF Action-Flag = 'I' OR 'U' CONTINUE ELSE DISPLAY 'Invalid Action-Flag' EXIT PARAGRAPH END-IF </pre>

Because of the way COBOL (GNU COBOL included) handles the abbreviation of conditional expressions, the conditional expression in the left-hand box is actually a short-hand version of the (not-so-intuitive):

IF Action-Flag NOT = 'I' AND Action-Flag NOT = 'U'

Inexperienced COBOL programmers would have coded the "IF" (incorrectly) as "IF Action-Flag NOT = 'I' OR 'U'", because it's basically how one might say it if describing the logic; this is sure to cause run-time problems as it actually represents "IF Action-Flag NOT = 'I' OR Action-Flag NOT = 'U' – not the same thing at all!

This causes many programmers to consider the code in the right-hand box to be more readable, even though it is a little longer.

See Also...

The **IF** Statement [6.2.21](#)

6.2.11. DELETE

Figure 6-42 - DELETE Syntax

```

DELETE file-name-1 RECORD
  [ invalid-key-clause ]
[ END-DELETE ]

```

The **DELETE** statement logically deletes a record from an **ORGANIZATION RELATIVE** or **ORGANIZATION INDEXED** file.

1. The **ORGANIZATION** of *file-name-1* must be **RELATIVE** or **INDEXED**.
2. For **RELATIVE** or **INDEXED** 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 sequential-format **READ** statement. That **READ** will therefore identify the record to be deleted.
3. If *file-name-1* is a **RELATIVE** file whose **ACCESS MODE** is either **RANDOM** or **DYNAMIC**, the record to be deleted is the one whose relative record number is currently the value of the field specified as the files **RELATIVE KEY** in it's **SELECT** statement.
4. If *file-name-1* is an **INDEXED** file whose **ACCESS MODE** is **RANDOM** or **DYNAMIC**, the record to be deleted is the one whose primary key is currently the value of the field specified as the **RECORD KEY** in the file's **SELECT** statement.
5. An "invalid key" condition will exist, and can be dealt with via the *invalid-key-clause*, if the record specified to be deleted by the **RELATIVE KEY** or **RECORD KEY** value does not exist in an access mode **RANDOM** or **DYNAMIC** file. This is a condition that cannot exist for **ACCESS MODE SEQUENTIAL** files because of rule #2. **DELETE** failures on **ACCESS MODE SEQUENTIAL** files can only be "handled" via **DECLARATIVES** (section).
6. No *invalid-key-clause* may be specified for a file who's **ACCESS MODE IS SEQUENTIAL**.

See Also...

Types of Files	1.3.3.5
Defining File Characteristics (SELECT)	4.2.1
Handling Invalid Keys (INVALID KEY)	6.1.12.3

Using DECLARATIVES	6.1.4
The READ Statement	6.4.31

6.2.12. DISPLAY

6.2.12.1. DISPLAY Format 1 – “UPON *device*”

Figure 6-43 - DISPLAY (Upon Console) Syntax

```

DISPLAY { literal-1 } ...
           [ UPON mnemonic-name-1 ]
           [ WITH NO ADVANCING ]
           [ exception-handler ]
           [ END-DISPLAY ]

```

This format of the **DISPLAY** statement displays the specified identifier contents and/or literal values on the specified device.

1. If no **UPON** clause is specified, **UPON CONSOLE** will be assumed. If the **UPON** clause is specified, *mnemonic-name-1* must be one of the built-in device names or a mnemonic name assigned to one of those devices via the **SPECIAL-NAMES** paragraph of the **CONFIGURATION SECTION**.
2. The **NO ADVANCING** clause, if used, will suppress the normal carriage-return / line-feed sequence that normally is added to the end of any console display. You can see an example of this at work in the sample program on page [6-59](#).
3. The optional *exception-handler* may be used to deal with errors attempting to display to the output device.

See Also...

The **SPECIAL-NAMES** Paragraph [4.1.4](#)

Handling Exceptions (**ON EXCEPTION**) [6.1.12.4](#)

Built-in Device Names [Figure 4-8](#)

4.

6.2.12.2. DISPLAY Format 2 – Access Command-Line Arguments

Figure 6-44 - DISPLAY (Access Command-line Arguments) Syntax

```

DISPLAY { literal-1 } ...
           UPON { ARGUMENT-NUMBER }
                 { COMMAND-LINE }
           [ exception-handler ]
           [ END-DISPLAY ]

```

This form of the **DISPLAY** statement may be used to specify the command-line argument number to be retrieved by a subsequent **ACCEPT** or to specify a new value for the command-line arguments themselves.

1. By **DISPLAY**ing a numeric integer value **UPON ARGUMENT-NUMBER**, you will specify which argument (by its relative number) will be retrieved by a subsequent **ACCEPT ... FROM ARGUMENT VALUE** statement.
2. Executing a **DISPLAY ... UPON COMMAND-LINE** will influence subsequent **ACCEPT ... FROM COMMAND-LINE** statements (which will then return the **DISPLAY**ed value), but will not influence subsequent **ACCEPT ... FROM ARGUMENT-VALUE** statements – these will continue to return the original program execution parameters.
3. The optional *exception-handler* may be used to deal any error that occur at run-time.

See Also...

Handling Exceptions (**ON EXCEPTION**) [6.1.12.4](#)

The **ACCEPT** Statement (Command Line) [6.2.1.2](#)

6.2.12.3. DISPLAY Format 3 – Access or Set Environment Variables

Figure 6-45 - DISPLAY (Access / Set Environment Variables) Syntax

```

DISPLAY { literal-1 } ...
           { identifier-1 } ...
           UPON { ENVIRONMENT-VALUE }
                 { ENVIRONMENT-NAME }
           [ exception-handler ]
[ END-DISPLAY ]

```

This form of the **DISPLAY** statement can be used to create or modify environment variables.

1. To create or change an environment variable will require two **DISPLAY** statements. The following example sets the environment variable "MY_ENV_VAR" to a value of "Demonstration Value":

```

DISPLAY "MY_ENV_VAR" UPON ENVIRONMENT-NAME
DISPLAY "Demonstration Value" UPON ENVIRONMENT-VALUE

```

2. Environment variables created or changed from within GNU COBOL programs will be available to any sub-shell processes spawned by that program (i.e. **CALL** "SYSTEM") but will not be known to the shell or console window that started the GNU COBOL program.
3. Consider using **SET ENVIRONMENT** in lieu of **DISPLAY** to set environment variables as it is much simpler.
4. The optional *exception-handler* may be used to deal any errorsthat occur at run-time.

See Also...

Handling Invalid Keys (**INVALID KEY**) [6.1.12.3](#)

The **SET ENVIRONMENT** Statement [6.4.39.1](#)

6.2.12.4. DISPLAY Format 4 – Screen Data

Figure 6-46 - DISPLAY (Screen Data) Syntax

```

DISPLAY { identifier-1 [ at-clause ] [ upon-clause ] [ with-clause ] } ...
           [ exception-handler ]
[ END-DISPLAY ]

```

This format of the **DISPLAY** statement presents data onto a formatted screen.

1. If *identifier-1* is defined in the **SCREEN SECTION**, any *at-clause*, *upon-clause* and *with-clause* specified for that identifier will be ignored, and all field positioning and screen control will occur as a result of the **SCREEN SECTION** definition of *identifier-1*.
2. The purpose of the *at-clause* is to define where on the screen *identifier-1* should be displayed. Consult the documentation for format 4 of the **ACCEPT** statement (Screen Data) for additional information.

```

AT { LINE NUMBER { integer-1 }
      { identifier-1 }
      { COLUMN POSITION } NUMBER { integer-2 }
      { identifier-2 }
      { integer-3 }
      { identifier-3 }
      ...

```

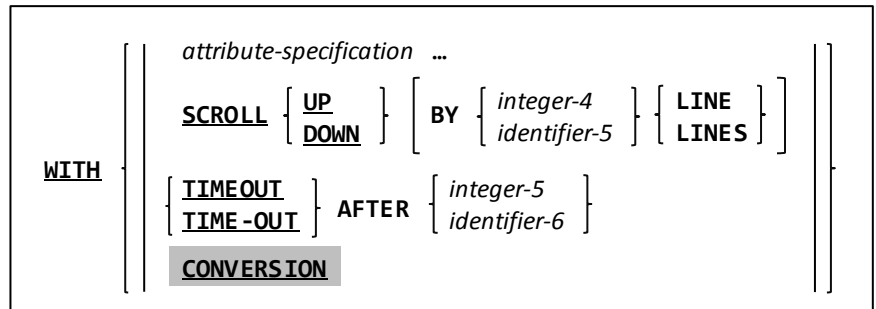
3. The **UPON** clause, while supported syntactically, is otherwise non-functional at this time.

```

[ UPON { CRT
          { CRT-UNDER } ]

```

4. The purpose of the *with-clause* is to define the visual attributes that should be applied to *identifier-1* when it is displayed on the screen. Consult the documentation for format 4 of the **ACCEPT** statement (Screen Data) for additional information.



The following *attribute-specification* clauses are allowed

on a **DISPLAY** statement *with-clause* – these are the same as those allowed for **SCREEN SECTION** data items.

BACKGROUND-COLOR	FOREGROUND-COLOR	UNDERLINE
BEEP BELL	HIGHLIGHT LOWLIGHT	ERASE EOL ERASE EOS
BLANK LINE BLANK SCREEN	OVERLINE	
BLINK	REVERSE-VIDEO	

4. The optional *exception-handler* may be used to deal any screen I/O errors that occur at run-time.

See Also...

Defining Screens [5.2.2](#)

The ACCEPT Statement (Screen Data) [6.4.1.4](#)

Handling Exceptions (**ON EXCEPTION**) [6.1.12.4](#)

6.2.13. DIVIDE

6.2.13.1. DIVIDE Format 1 – DIVIDE INTO

Figure 6-47 - DIVIDE INTO Syntax

```

DIVIDE { literal-1 } INTO { identifier-2 [ rounding-option ] } ...
      [ size-error-clause ]
[ END-DIVIDE ]

```

This format of **DIVIDE** will divide a specified value into one or more data items, replacing the value in each of those data items with the result of its old value divided by the *identifier-1* or *literal-1* value. Any remainder calculated as a result of the division is discarded.

1. *Identifier-1* and *identifier-2* must be numeric unedited data items and *literal-1* must be a numeric literal.
2. The optional “*rounding-option*” clause available to each *identifier-2* will control how non-integer results will be saved.
3. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-2* is insufficiently sized to hold the generated results; this clause will also detect attempts to divide by zero.

See Also...

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

Rounding Options [6.1.12.7](#)

6.2.13.2. DIVIDE Format 2 – DIVIDE INTO GIVING

Figure 6-48 - DIVIDE INTO GIVING Syntax

```

DIVIDE { literal-1 } INTO { literal-2 }
      GIVING { identifier-3 [ rounding-option ] } ...
      [ REMAINDER identifier-4 ]
      [ size-error-clause ]
[ END-DIVIDE ]

```

This format of **DIVIDE** will divide a specified value (*identifier-1* or *literal-1*) into another value (*identifier-2* or *literal-2*) and will then replace the contents of one or more receiving data items (*identifier-3* ...) with the results of that division.

Any remainder calculated as a result of the division is discarded unless a **REMAINDER** clause is present.

1. *Identifier-1* and *identifier-2* must be numeric unedited data items, *identifier-3* and *identifier-4* must be numeric (edited or unedited) data items and *literal-1* and *literal-2* must be numeric literals.
2. The optional “*rounding-option*” clause available to each *identifier-3* will control how non-integer results will be saved.
3. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-3* is insufficiently sized to hold the generated results; this clause will also detect attempts to divide by zero.

See Also...

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

Rounding Options [6.1.12.7](#)

6.2.13.3. DIVIDE Format 3 – DIVIDE BY GIVING

Figure 6-49 - DIVIDE BY GIVING Syntax

```

DIVIDE { literal-1 } BY { literal-2 }
         { identifier-1 }
         GIVING { identifier-3 [ rounding-option ] } ...
         [ REMAINDER identifier-4 ]
         [ size-error-clause ]
         [ END-DIVIDE ]

```

This format of **DIVIDE** will divide a specified value (*identifier-1* or *literal-1*) by another value (*identifier-2* or *literal-2*) and will then replace the contents of one or more receiving data items (*identifier-3* ...) with the results of that division.

Any remainder calculated as a result of the division is discarded unless a **REMAINDER** clause is present.

1. *Identifier-1* and *identifier-2* must be numeric unedited data items, *identifier-3* and *identifier-4* must be numeric (edited or unedited) data items and *literal-1* and *literal-2* must be numeric literals.
2. The optional “*rounding-option*” clause available to each *identifier-3* will control how non-integer results will be saved.
3. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-3* is insufficiently sized to hold the generated results; this clause will also detect attempts to divide by zero.

See Also...

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

Rounding Options [6.1.12.7](#)

6.2.14. ENTRY

Figure 6-50 - ENTRY Syntax

```
ENTRY literal-1 [ USING argument-1 ... ]
```

The **ENTRY** statement is used to define an alternate *entry-point* into a subroutine, along with the arguments that subroutine will be expecting.

1. You may not use an **ENTRY** statement in a nested subprogram.
2. The **USING** clause defines the arguments the subroutine entry-point supports. This list of arguments must match up against the **USING** clause of any **CALL** statements that will be invoking the subroutine using this entry-point.
3. Each *argument-n* specified on the **ENTRY** statement must be defined in the **LINKAGE SECTION** of the subprogram in which the **ENTRY** statement exists.
4. The *literal-1* value will specify the entry-point name of the subroutine. It must be specified exactly on **CALL** statements (with regard to the use of upper- and lower-case letters) as it is specified on the **ENTRY** statement.
5. Each *argument-n* entry must follow the syntax shown to the right. The usage of **REFERENCE**, **CONTENT** and **VALUE** on an argument should match the manner in which that argument is being passed on the **CALL** statement.

Figure 6-51 - ENTRY Statement Argument Syntax

```
[ BY { REFERENCE  
          CONTENT  
          VALUE } ] identifier-1
```

See Also...

The DATA DIVISION	5
The CALL Statement	6.4.5

Sub-programming	7
Details of Nested Subprograms	7.6

6.2.15. EVALUATE

Figure 6-52 - EVALUATE Syntax

```

EVALUATE selection-subject-1 [ ALSO selection-subject-2 ] ...
  { { WHEN selection-object-1 [ ALSO selection-object-2 ] } ... [ imperative-statement-1 ] } ...
  [ WHEN OTHER imperative-statement-2 ]
[ END-EVALUATE ]

```

The **EVALUATE** statement provides a means of defining processing that should take place under a multitude of conditions.

- There must be at least one **WHEN** clause specified on any **EVALUATE** statement. There may also be multiple **WHEN** clauses specified.
- There must be at least one *selection-subject* specified on the **EVALUATE** statement itself. The syntax of a *selection-subject* is shown to the right.
- Each *selection subject* will have its value matched against the corresponding *selection object* value on every **WHEN** clause.
- The first **WHEN** clause having each of its *selection-object(s)* successfully matched by the corresponding *selection-subject* on the **EVALUATE** statement will be the one whose *imperative-statement-1* (if any) is executed. If the successfully matched **WHEN** clause does not have its own *imperative-statement-1* then the next *imperative-statement-1* (on another **WHEN** clause) following the **WHEN** that was matched will be executed.
- If no **WHEN** clause has its *imperative-statement-1* executed, then the **WHEN OTHER** clause's *imperative-statement-2* will be executed (if **WHEN OTHER** was specified).
- Once *imperative-statement-1* or *imperative statement-2* is executed (or would have been executed if it existed), control will proceed with the statement following the **END-EVALUATE**.
- The syntax of a *selection-object* is shown to the right.
- The reserved words **THRU** and **THROUGH** may be used interchangeably.
- When using **THRU**, the values on both sides of the **THRU** must be the same class (both numeric, both alphanumeric, etc.).
- A *partial-expression* is one of the following:
 - A *class-condition* without a leading *identifier-1*
 - A *sign-condition* without a leading *identifier-1*
 - A *relation-condition* with nothing to the left of the relational operator
- In order for a *selection-subject* to match the corresponding *selection-object* on a **WHEN** clause, one of the following must be true:
 - The *selection-object* is **ANY**
 - The value of the *selection-subject* is equal to the value of the *selection object*
 - The value of the *selection-subject* falls within the range specified by the **THRU** clause of the *selection-object*
 - If the *selection-object* is a *partial-expression* (see #10, above), then the true/false result that would be obtained if the *partial-expression* is applied to the *selection-subject* must be true; this will be illustrated in an upcoming example

Syntax of a *selection-subject*

```

[ TRUE
  FALSE
  expression-1
  identifier-1
  literal-1 ]

```

Syntax of a *selection-object*

```

[ ANY
  TRUE
  FALSE
  partial-expression-1
  [ expression-2
    identifier-2
    literal-2 ] [ THRU | THROUGH [ expression-3
    identifier-3
    literal-3 ] ] ]

```

Here is a sample program that illustrates the **EVALUATE** statement.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. DEMOEVALUATE.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Test-Digit PIC 9(1).
88 Digit-Is-Odd VALUE 1, 3, 5, 7, 9.

```

Console output when run (user input is highlighted):

```

Enter a digit (0 Quits): 1
1 is PRIME and it's small too

```

```

      88 Digit-Is-Prime VALUE 1, 3, 5, 7.
PROCEDURE DIVISION.
P1. PERFORM UNTIL EXIT
    DISPLAY "Enter a digit (0 Quits): "
      WITH NO ADVANCING
    ACCEPT Test-Digit
    IF Test-Digit = 0
      EXIT PERFORM
    END-IF
    EVALUATE Digit-Is-Odd ALSO Digit-Is-Prime
      WHEN TRUE ALSO FALSE
        DISPLAY Test-Digit " is ODD"
          WITH NO ADVANCING
      WHEN TRUE ALSO TRUE
        DISPLAY Test-Digit " is PRIME"
          WITH NO ADVANCING
      WHEN FALSE ALSO ANY
        DISPLAY Test-Digit " is EVEN"
          WITH NO ADVANCING
    END-EVALUATE
    EVALUATE Test-Digit
      WHEN < 5
        DISPLAY " and it's small too"
      WHEN < 8
        DISPLAY " and it's medium too"
      WHEN OTHER
        DISPLAY " and it's large too"
    END-EVALUATE
    END-PERFORM
    DISPLAY "Bye!"
    STOP RUN
.
```

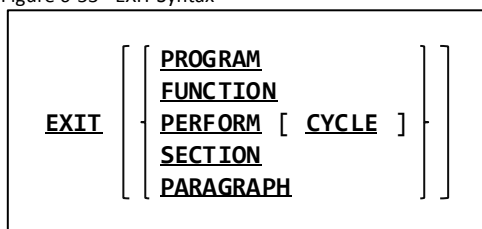
```

Enter a digit (0 Quits): 2
2 is EVEN and it's small too
Enter a digit (0 Quits): 3
3 is PRIME and it's small too
Enter a digit (0 Quits): 4
4 is EVEN and it's small too
Enter a digit (0 Quits): 5
5 is PRIME and it's medium too
Enter a digit (0 Quits): 6
6 is EVEN and it's medium too
Enter a digit (0 Quits): 7
7 is PRIME and it's medium too
Enter a digit (0 Quits): 8
8 is EVEN and it's large too
Enter a digit (0 Quits): 9
9 is ODD and it's large too
Enter a digit (0 Quits): 0
Bye!
```

*See Also...*Class Tests [6.1.4.2.2](#)Sign Tests [6.1.8.2.3](#)Relation Tests [6.1.8.2.5](#)

6.2.16. EXIT

Figure 6-53 - EXIT Syntax



The **EXIT** statement is a multi-purpose statement; it may provide a common end point for a series of procedures, exit an inline PERFORM, a paragraph or a section or it may mark the logical end of a subprogram.

1. When used without any of the optional clauses, the “EXIT” statement simply provides a common “GO TO” end point for a series of procedures. [Figure 6-57](#) illustrates this usage of the EXIT statement.
2. When an EXIT statement is used, it must be the only statement in the paragraph in which it occurs.
3. The EXIT statement takes no other run-time action.

Figure 6-54 - Using the EXIT Statement

```

01 Switches.
05 Input-File-Switch PIC X(1).
88 EOF-On-Input-File VALUE 'Y' FALSE 'N'.
.
.
.
SET EOF-On-Input-File TO FALSE.
PERFORM 100-Process-A-Transaction
THRU 199-Exit
UNTIL EOF-On-Input-File.
.
.
.
100-Process-A-Transaction.
READ Input-File AT END
SET EOF-On-Input-File TO TRUE
GO TO 199-Exit.
IF Input-Rec of Input-File = SPACES
GO TO 199-Exit. *> IGNORE BLANK RECORDS!
process the record just read
199-Exit.
EXIT.

```

4. An **EXIT PARAGRAPH** statement transfers control to a point immediately past the end of the current paragraph, while an **EXIT SECTION** statement causes control to pass to point immediately past the last paragraph in the current section. If the **EXIT PARAGRAPH** or **EXIT SECTION** resides in a paragraph within the scope of a procedural **PERFORM**, control will be returned back to the **PERFORM** for evaluation of any **TIMES**, **VARYING** and/or **UNTIL** clauses. If the **EXIT PARAGRAPH** or **EXIT SECTION** resides outside the scope of a procedural **PERFORM**, control simply transfers to the first executable statement in the next paragraph (**EXIT PARAGRAPH**) or section (**EXIT SECTION**).

[Figure 6-58](#) shows how the example shown in [Figure 6-57](#) could have been coded without a **GO TO** by utilizing an **EXIT PARAGRAPH** statement.

Figure 6-55 - Using EXIT PARAGRAPH

```

01 Switches.
05 Input-File-Switch PIC X(1).
88 EOF-On-Input-File VALUE 'Y' FALSE 'N'.
.
.
.
SET EOF-On-Input-File TO FALSE.
PERFORM 100-Process-A-Transaction
UNTIL EOF-On-Input-File.
.
.
.
100-Process-A-Transaction.
READ Input-File AT END
SET EOF-On-Input-File TO TRUE
EXIT PARAGRAPH.
IF Input-Rec of Input-File = SPACES
EXIT PARAGRAPH. *> IGNORE BLANK RECORDS!
process the record just read

```

5. The **EXIT PERFORM** and **EXIT PERFORM CYCLE** statements are intended to be used in conjunction with an inline **PERFORM** statement.
6. An **EXIT PERFORM CYCLE** will terminate the current iteration of the inline **PERFORM**, giving control to any **TIMES**, **VARYING** and/or **UNTIL** clauses for them to determine if another cycle needs to be performed.

7. An **EXIT PERFORM** will terminate the inline **PERFORM** outright, transferring control to the first statement following the **PERFORM**. [Figure 6-59](#) shows the final modification to the [Figure 6-57](#) example; by using Inline **PERFORM** and **EXIT PERFORM** statements we can really streamline processing.

Figure 6-56 - Using the EXIT PERFORM Statement

```

PERFORM UNTIL EXIT
  READ Input-File AT END
  EXIT PERFORM
END-READ
IF Input-Rec of Input-File = SPACES
  EXIT PERFORM CYCLE *-> IGNORE BLANK RECORDS!
END-IF
  process the record just read
END PERFORM

```

8. The **EXIT PROGRAM** and **EXIT FUNCTION** statements terminate the execution of a subroutine (i.e. a program that has been **CALLED** by another) or user-defined function, respectively. An **EXIT PROGRAM** statement returns control back to the statement following the **CALL** of the subprogram. An **EXIT FUNCTION** returns control back to the processing of the statement in the calling program that invoked the user-defined function.
9. If executed by a main program, neither the **EXIT PROGRAM** nor **EXIT FUNCTION** statements are non-functional. The **EXIT PROGRAM** statement is not legal anywhere within a user-defined function and **EXIT FUNCTION** cannot be used anywhere within a subroutine. Neither may be used within a **USE GLOBAL** routine in **DECLARATIVES**.
10. The COBOL2002 standard has made a common extension to the COBOL language - the **GOBACK** statement – now a standard language element; the **GOBACK** statement should be strongly considered as the preferred alternative to **EXIT PROGRAM** and **EXIT FUNCTION** for new subprograms.

See Also...

Using DECLARATIVES	6.1.4	The PERFORM Statement (Procedural)	6.2.30.1
The CALL Statement	6.4.5	The PERFORM Statement (Inline)	6.4.30.2
The GOBACK Statement	6.2.19	Sub-programming	7
The GO TO Statement	6.2.20	Subprograms Subroutines vs Functions	7.1

6.2.17. FREE

Figure 6-57 - FREE Syntax

```
FREE { [ ADDRESS OF ] identifier-1 } ...
```

The **FREE** statement releases memory previously allocated to the program by the **ALLOCATE** statement.

1. *Identifier-1* must be a **USAGE POINTER** data item or an 01-level data item with the **BASED** attribute.
2. If *identifier-1* is a **USAGE POINTER** data item and it contains a valid address, the **FREE** statement will release the memory block the pointer references. In addition, any **BASED** data items that the pointer was used to provide an address for will become un-based and therefore un-usable. If *identifier-1* did not contain a valid address, no action will be taken.
3. If *identifier-1* is a **BASED** data item and that data item is currently based (meaning it currently has memory allocated for it), its memory is released and *identifier-1* will become un-based and therefore un-usable. If *identifier-1* was not based, no action will be taken.
4. The **ADDRESS OF** clause adds no special function to the **FREE** statement.

See Also...


Dynamically Allocated Items (**BASED**) [5.2.1.2](#)

Storage Format of Data (**USAGE**) [5.2.1.11](#)

The **ALLOCATE** Statement [6.4.3](#)

6.2.18. GENERATE

Figure 6-58 - GENERATE Syntax



```
GENERATE { identifier-1  
          report-name-1 }
```

Although syntactically recognized by the GNU COBOL compiler, the **GENERATE** statement is non-functional because the RWCS (COBOL Report Writer Control System) is not currently supported by GNU COBOL.

6.2.19. GOBACK

Figure 6-59 - GOBACK Syntax

GOBACK

The **GOBACK** statement is used to logically terminate an executing program.

1. If executed within a subprogram (i.e. a subroutine or user-defined function), **GOBACK** behaves like an **EXIT PROGRAM** or **EXIT FUNCTION** statement, respectively.
2. If executed within a main program, **GOBACK** will act as a **STOP RUN** statement.

See Also...

The **EXIT FUNCTION** Statement [6.2.16](#)

The **EXIT PROGRAM** Statement [6.2.16](#)

The **STOP RUN** Statement [6.4.42](#)

Sub-programming 7

3.

6.2.20. GO TO

6.2.20.1. GO TO Format 1 – Simple GO TO

Figure 6-60 - Simple GO TO Syntax

```
GO TO procedure-name-1
```

This form of the **GO TO** statement unconditionally transfers control in a program to the specified *procedure-name-1*.

1. If *procedure-name-1* is a section, control will transfer to the first paragraph in that section.

6.2.20.2. GO TO Format 2 – GO TO DEPENDING ON

Figure 6-61 – GO TO DEPENDING ON Syntax

```
GO TO procedure-name-1 ...  
DEPENDING ON identifier-1
```

This form of the **GO TO** statement will transfer control to any one of a number of specified procedure names depending on the numeric value of the identifier specified on the statement.

1. The **PICTURE** and/or **USAGE** of the specified *identifier-1* must be such as to define it as a numeric, unedited, preferably unsigned integer data item.
2. If the value of *identifier-1* has the value 1, control will be transferred to the 1st specified procedure name. If the value is 2, control will transfer to the 2nd procedure name, and so on.
3. If the value of *identifier-1* is less than 1 or exceeds the total number of procedure names specified on the **GO TO** statement, control will simply fall thru into the next statement following the **GO TO**.
4. The following table shows how **GO TO DEPENDING ON** may be used in a real application situation, and compares it against the two alternatives – **IF** and **EVALUATE**.

Figure 6-62 - GOTO DEPENDING ON vs IF vs EVALUATE

GOTO DEPENDING ON	IF	EVALUATE
<pre>GO TO PROCESS-ACCT-TYPE-1 PROCESS-ACCT-TYPE-2 PROCESS-ACCT-TYPE-3 DEPENDING ON ACCT-TYPE. Code to handle invalid account type GO TO DONE-WITH-ACCT-TYPE. PROCESS-ACCT-TYPE-1. Code to handle account type 1 GO TO DONE-WITH-ACCT-TYPE. PROCESS-ACCT-TYPE-2. Code to handle account type 2 GO TO DONE-WITH-ACCT-TYPE. PROCESS-ACCT-TYPE-3. Code to handle account type 3 DONE-WITH-ACCT-TYPE.</pre>	<pre>IF ACCT-TYPE = 1 Code to handle account type 1 ELSE IF ACCT-TYPE = 2 Code to handle account type 2 ELSE IF ACCT-TYPE = 3 Code to handle account type 3 ELSE Code to handle invalid account type END-IF END-IF END-IF</pre>	<pre>EVALUATE ACCT-TYPE WHEN 1 Code to handle account type 1 WHEN 2 Code to handle account type 2 WHEN 3 Code to handle account type 3 WHEN OTHER Code to handle invalid account type END-EVALUATE.</pre>

There is no question that “modern programming philosophy” would prefer the **EVALUATE** approach. An interesting note is that the code generated by the **IF** and **EVALUATE** techniques is virtually identical.

See Also...

The **EVALUATE** Statement [6.2.15](#)

The **IF** Statement [6.2.21](#)

6.2.21. IF

Figure 6-63 - IF Syntax

```
IF conditional-expression THEN imperative-statement-1  
[ ELSE imperative-statement-2 ]  
[ END-IF ]
```

The **IF** statement is used to conditionally execute an imperative statement or to select one of two different imperative statements based upon the TRUE/FALSE value of a conditional expression.

1. If *conditional-expression* evaluates to true, *imperative-statement-1* will be executed regardless of whether or not an **ELSE** clause is present. Once *imperative-statement-1* has been executed, control falls into the first statement following the **END-IF** or to the first statement of the next sentence if there is no **END-IF** clause.
2. If the optional **ELSE** clause is present and *conditional-expression-1* evaluates to false, then (and only then) *imperative-statement-2* will be executed. Once *imperative-statement-2* has been executed, control falls into the first statement following the **END-IF** or to the first statement of the next sentence if there is no **END-IF** clause.
3. The **END-IF** statement isn't the only way the scope of an IF (or ELSE) can be terminated – the period character (.) can be used also to terminate the IF/ELSE by ending the sentence in which it is coded.

See Also...

Conditional Expressions [6.1.8.2](#)

Use of Periods (.) [6.1.5](#)

6.2.22. INITIALIZE

Figure 6-64 - INITIALIZE Syntax

```

INITIALIZE identifier-1 ... [ WITH FILLER ]
    [ { ALL
      category-name } TO VALUE ]
    [ THEN REPLACING { category-name DATA BY [ LENGTH OF ] { literal-1
      identifier-2 } } ... ]
    [ THEN TO DEFAULT ]

```

The **INITIALIZE** statement initializes each *identifier-1* with certain specific values, depending upon the options specified.

1. From the sequence of *identifier-1* data items specified on the **INITIALIZE** statement, a list of initializable fields, referred to as the *field list* in the remainder of this section, will include:
 - a. Every *identifier-1* that is an elementary item.
 - b. Every *identifier-1* that is a group item will have each elementary item defined anywhere within its full hierarchical structure included, excluding **FILLER** items.
 - c. If the optional **WITH FILLER** clause is included on the **INITIALIZE** statement, then rule #1.b above will include **FILLER** items.

Any *identifier-1* containing a **REDEFINES** in its definition will be included in the field list, but items defined subordinate to any *identifier-1* that contain **REDEFINES** in their descriptions (and any items subordinate to them as well) will be excluded.

2. A *category-name* may be any of the following:

ALPHABETIC	The PICTURE of any ALPHABETIC data item only contains A symbols
ALPHANUMERIC	The PICTURE of any ALPHANUMERIC data item contains only A , X and 9 symbols (but all A symbols is considered ALPHABETIC and all 9 symbols is considered NUMERIC)
ALPHANUMERIC-EDITED	The PICTURE of any ALPHANUMERIC-EDITED data item is that it is an ALPHANUMERIC data item that <u>also</u> contains B , 0 (zero) and/or slash (/) symbols
NUMERIC	A NUMERIC data item is one that is described with one of the pictureless USAGES (see Figure 5-10) or has a PICTURE composed of nothing but P , 9 , S and V symbols.
NUMERIC-EDITED	The PICTURE of any NUMERIC-EDITED data item is one that must have a PICTURE clause in it's definition, and that clause contains nothing but the symbol 9 and any editing symbol defined in Figure 5-7 .

3. The behavior of an **INITIALIZE** without a **VALUE** or **REPLACING** clause (either with or without a **DEFAULT** clause) will be to move zeros into every numeric or numeric-edited data item (as defined above) in the field list and, **SPACES** into all remaining fields in the initializable field list.
4. The behavior of an **INITIALIZE** with a **VALUE** and/or **REPLACING** clause will be as follows:
 - a. If there is an "**ALL TO VALUE**" clause present then all data items in the field list having an explicit **VALUE** clause coded in their description or having an implicit **VALUE** clause inherited from their parent group item will be initialized to that compile-time value.

If there is a "*category-name* **TO VALUE**" clause present then all data items in the field list that fall into the specified category (see the list above) and have either an explicit **VALUE** clause coded in their description or have an implicit **VALUE** clause inherited from their parent group item will be initialized to that compile-time value.

Any data items in the field list that get initialized by this rule will be excluded from the remaining rules.
 - b. If there is a "**REPLACING**" clause present, then all data items in the fields list that weren't initialized by rule #4.a and that fall into the specified category (see the list above) will be initialized to the value specified by

literal-1 or *identifier-2*. You may specify multiple “*category-name BY value*” clauses, but each must specify a unique *category-name*.

Any data items in the field list that get initialized by this rule will be excluded from the remaining rules.

- c. Finally, if there are any data items in the field list that weren’t initialized either by rule #4.a or #4.b and there is a DEFAULT clause present, those remaining data items will be initialized according to rule #3.

The following example may help your understanding of how the INITIALIZE statement works. The sample code makes use of the COBDUMP program documented in section [10.2](#) to dump the storage that is (or is not) being **INITIALIZED**.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DemoInitialize.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Item-1.
   05 I1-A VALUE ALL '*'.
      10 FILLER                               PIC X(1).
      10 I1-A-1                               PIC 9(1) VALUE 9.
   05 I1-B                                   USAGE BINARY-CHAR.
   05 I1-C                                   PIC A(1) VALUE 'C'.
   05 I1-D                                   PIC X/X VALUE 'ZZ'.
   05 I1-E                                   OCCURS 2 TIMES PIC 9.
PROCEDURE DIVISION.
000-Main.
   DISPLAY "MOVE HIGH-VALUES TO Item-1"
   PERFORM 100-Init-Item-1
   CALL "COBDUMP" USING Item-1
   DISPLAY " "

   DISPLAY "INITIALIZE Item-1"
   INITIALIZE Item-1
   CALL "COBDUMP" USING Item-1
   PERFORM 100-Init-Item-1
   DISPLAY " "

   DISPLAY "INITIALIZE Item-1 WITH FILLER"
   MOVE HIGH-VALUES TO Item-1
   INITIALIZE Item-1 WITH FILLER
   CALL "COBDUMP" USING Item-1
   PERFORM 100-Init-Item-1
   DISPLAY " "

   DISPLAY "INITIALIZE Item-1 ALL TO VALUE"
   MOVE HIGH-VALUES TO Item-1
   INITIALIZE Item-1 ALPHANUMERIC TO VALUE
   CALL "COBDUMP" USING Item-1
   PERFORM 100-Init-Item-1
   DISPLAY " "

   DISPLAY "INITIALIZE Item-1 REPLACING NUMERIC BY 1"
   MOVE HIGH-VALUES TO Item-1
   INITIALIZE Item-1 REPLACING NUMERIC BY 1
   CALL "COBDUMP" USING Item-1
   PERFORM 100-Init-Item-1
   DISPLAY " "

   STOP RUN
   .

100-Init-Item-1.
   MOVE HIGH-VALUES TO Item-1
   .
```


When executed, this program produces the following output:

```

MOVE HIGH-VALUES TO Item-1
<-Addr-> Byte <----- Hexadecimal -----> <---- Char ---->
=====
00404058   1 FF FF FF FF FF FF FF FF FF          .....

INITIALIZE Item-1
<-Addr-> Byte <----- Hexadecimal -----> <---- Char ---->
=====
00404058   1 FF 30 00 20 20 2F 20 30 30          .0. / 00

INITIALIZE Item-1 WITH FILLER
<-Addr-> Byte <----- Hexadecimal -----> <---- Char ---->
=====
00404058   1 20 30 00 20 20 2F 20 30 30          0. / 00

INITIALIZE Item-1 ALL TO VALUE
<-Addr-> Byte <----- Hexadecimal -----> <---- Char ---->
=====
00404058   1 2A 2A FF 43 5A 5A 20 FF FF          **.CZZ ..

INITIALIZE Item-1 REPLACING NUMERIC BY 1
<-Addr-> Byte <----- Hexadecimal -----> <---- Char ---->
=====
00404058   1 FF 31 01 FF FF FF FF 31 31          .1.....11

```

6.2.23. INITIATE

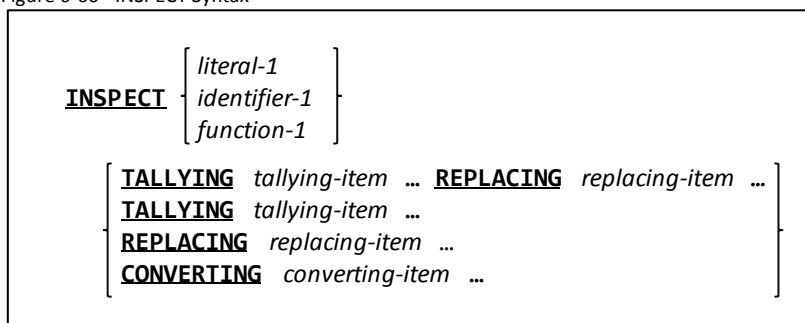
Figure 6-65 - INITIATE Syntax

```
INITIATE report-name-1 ...
```

Although syntactically recognized by the GNU COBOL compiler, the **INITIATE** statement is non-functional because the RWCS (COBOL Report Writer Control System) is not currently supported by GNU COBOL.

6.2.24. INSPECT

Figure 6-66 - INSPECT Syntax



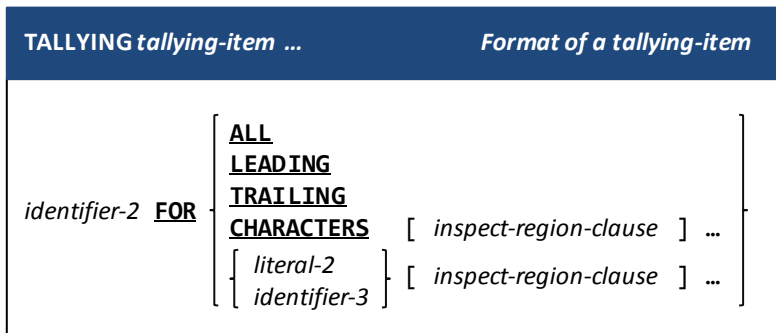
The **INSPECT** statement is used to perform various counting or data-alteration operations against strings.

1. *Identifier-1* and *literal-1* must be explicitly or implicitly defined as alphanumeric **USAGE DISPLAY** data. *Identifier-1* may be a group item. If *function-1* is specified, it must be an invocation of an intrinsic function that returns a string result. This is referred to as the **inspect target**.
2. A **TALLYING** clause will count the number of occurrences of a string of characters in the [inspect target](#).
3. A **REPLACING** clause will convert occurrences of strings in the [inspect target](#) to different (equally-sized) strings (for example, replacing all occurrences of "ABC" by "DEF"). The [inspect target](#) cannot be a literal or function result when using **REPLACING**.
4. A **CONVERTING** clause will perform any number of single character replacements in the [inspect target](#). The [inspect target](#) cannot be a literal or function result when using **CONVERTING**.
5. If both **TALLYING** and **REPLACING** are specified on the same **INSPECT** statement, the effect will be as if two **INSPECT** statements had been coded – the first performing the **TALLYING** and the second performing the **REPLACING**.

6.2.24.1. TALLYING Clause Syntax, Rules and Operation

The purpose of the **TALLYING** clause is to count how many occurrences of one or more strings appear within all or a subset of the [inspect target](#).

Each search string is specified using a single *tallying-item* after the **TALLYING** keyword. The syntax of a single *tallying item* is shown to the right.

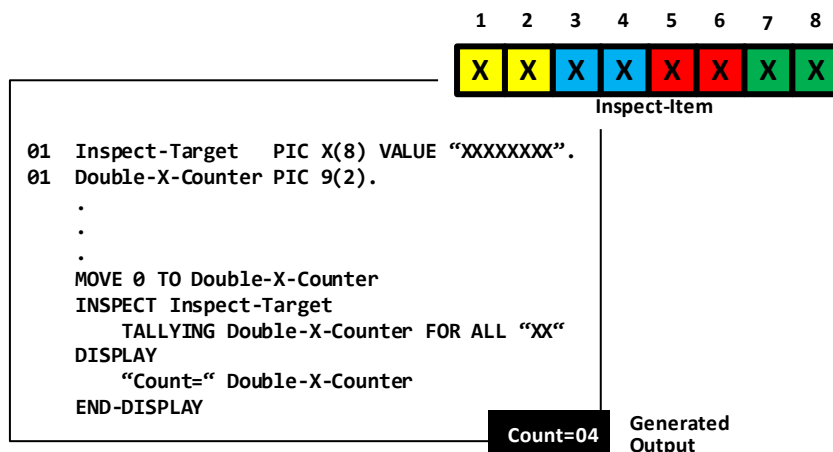


1. *Identifier-2* must be an unedited numeric item.
2. *Identifier-3* and *literal-2* must be explicitly or implicitly defined as alphanumeric **USAGE DISPLAY** data. *Identifier-3* may be a group item.
3. The *inspect-region-clause*(s) limit **TALLYING** processing to a specific subset of the [inspect target](#). If no *inspect-region-clause* is specified, the entire [inspect target](#) will be searched.
4. *Identifier-2* may be specified in multiple *tallying-items*.
5. *Identifier-2* will be incremented by 1 each time the target string being searched for is found within the specified range of the [inspect target](#). The target string will be:
 - a. Any single character if the **CHARACTERS** option is used; this form basically just counts total characters
 - b. **ALL**, all **LEADING** or all **TRAILING** occurrences of *Identifier-3* or *literal-2*.

- Once an occurrence of the target string is found and TALLYed, the INSPECT TALLYING process will resume from the end of the found occurrence. This prevents the possibility of counting overlapping occurrences.

The example shows an 8-character item whose value is "XXXXXXXX" used as the object of an INSPECT TALLYING that is looking for "XX" occurrences:

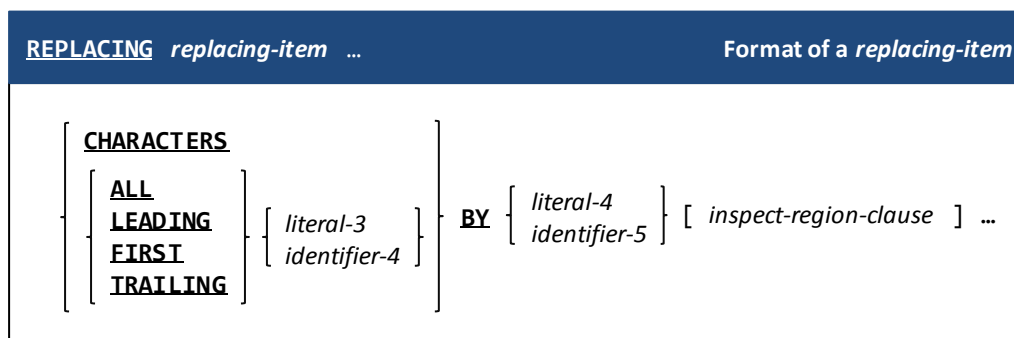
Figure 6-67 - An INSPECT TALLYING Example



Only four (4) "XX" occurrences were found. Character positions 2-3, 4-5 and 6-7 – even though they are "XX" occurrences – weren't counted because they overlapped other occurrences.

6.2.24.2. REPLACING Clause Syntax, Rules and Operation

The purpose of the REPLACING clause is to replace occurrences of a substring within the inspect target with a different substring of the same length. If you need to replace one or more substrings with others of a different length, consider using the SUBSTITUTE or SUBSTITUTE-CASE intrinsic function.



Each search and replace string is specified using a single replacing-item after the REPLACING keyword. The syntax of a single replacing item is shown above.

- Identifier-4 and literal-3 (known as the target string) must be explicitly or implicitly defined as alphanumeric USAGE DISPLAY data. Identifier-4 may be a group item.
- Identifier-5 and literal-4 (known as the replacement string) must be explicitly or implicitly defined as alphanumeric USAGE DISPLAY data. Identifier-5 may be a group item.
- Identifier-4 / literal-3 must be the same length as identifier-5 / literal-4.
- Target strings are identified as:
 - Any sequence of characters as long as the length of the replacement string if the CHARACTERS option is used
 - ALL, all LEADING, only the FIRST or all TRAILING occurrences of Identifier-4 or literal-3.
- The inspect-region-clause(s) limit REPLACING processing of any one specific replacing-item to a specific region of the inspect target. If no inspect-region-clause is specified, the entire inspect target will be processed. Different replacing-items may have different regions specified.
- REPLACING processing works as follows:
 - Processing begins with the first character of the inspect target an internal character pointer index to the first character position.
 - If the internal character pointer is pointing past the end of the inspect target, REPLACING processing is complete and the INSPECT statement will terminate.

- c. Each replacing-item is checked, in the sequence in which they are coded on the **INSPECT** statement, looking for one whose *inspect-region-clauses* allow its *target-string* to match the substring of the [inspect target](#) that begins with the current character of that [inspect target](#) currently being pointed to.
- d. If no replacing-items can match the [inspect target](#) from the current character position forward, the character pointer is advanced by one and processing returns to rule #6.b.
- e. If a match is found, that *replacing-item's* replacement-string will replace the target-string in the [inspect target](#) (starting at the current character position). If the *replacing-item's* coding specified the **FIRST** option, that *replacing-item* will be disabled for any further iterations during this execution of the **INSPECT** statement. The current character pointer into the [inspect target](#) will be set to the first character following the replaced string and processing returns to rule #6.b.

See Also...

The **SUBSTITUTE** Intrinsic Function [6.1.14.77](#)

The **SUBSTITUTE-CASE** Intrinsic Function [6.1.14.78](#)

6.2.24.3. CONVERTING Clause Syntax, Rules and Operation

The purpose of the **CONVERTING** clause is to perform a series of monocharacter substitutions against a data item.

Each search and replace character sequence is specified using a single *converting-item* after the **CONVERTING** keyword. The syntax of a single *converting item* is shown to the right.

CONVERTING *converting-item* Format of a *converting-item*

$$\left\{ \begin{array}{l} \textit{literal-5} \\ \textit{identifier-6} \end{array} \right\} \textbf{TO} \left\{ \begin{array}{l} \textit{literal-6} \\ \textit{identifier-7} \end{array} \right\} [\textit{inspect-region-clause}] \dots$$

1. *Identifier-6*, *identifier-7*, *literal-5* and *literal-6* must be explicitly or implicitly defined as alphanumeric **USAGE DISPLAY** data. *Identifier-6* and *identifier-7*, if used, may be group items.
2. *Identifier-6* / *literal-5* (the "from string") should be the same length as *identifier-7* / *literal-6* (the "to string"). If they aren't:
 - a. If the length of the *from string* exceeds the length of the *to string*, then the *to string* will be assumed to be padded to the right with spaces to make them the same length.
 - b. If the length of the *to string* exceeds the length of the *from string*, then the *to string* will be assumed to be truncated to the length of the *from string*.
3. Each character within the [inspect target](#) that lies within the range limits defined by the *inspect-region-clause(s)*, if any, will be searched for within the *from string*. If found, that [inspect target](#) character will be replaced by the *to string* character that corresponds (by relative position) to the character found in the *from string*.

6.2.24.4. INSPECT Region Clause, Rules and Operation

The purpose of an *inspect-region-clause* is to restrict the operation of a **TALLYING**, **REPLACING** or **CONVERTING** clause to a specific range of characters within the [inspect target](#).

If multiple *inspect-region-clauses* are specified, the effects of them as a group will serve to define the range.

Format of an *inspect-region-clause*

$$\left\{ \begin{array}{l} \textbf{BEFORE} \\ \textbf{AFTER} \end{array} \right\} \textbf{INITIAL} \left\{ \begin{array}{l} \textit{literal-7} \\ \textit{identifier-8} \end{array} \right\}$$

1. *Identifier-8* and *literal-7* must be explicitly or implicitly defined as alphanumeric **USAGE DISPLAY** data. *Identifier-8* may be a group item. They may be of any length.

The following example illustrates how a range clause works and how multiple range clauses can work together. It also illustrates how COBOL syntax allows potentially complicated operations to be coded in an easy-to-understand manner.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DemoINSPECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  Inspect-Target          PIC X(100) VALUE
    'THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG ' &
    'AND WAS BITTEN ON THE TAIL. THE FOX YELPED!'.
PROCEDURE DIVISION.
P1. DISPLAY "Before: " Inspect-Target
    INSPECT Inspect-Target
        REPLACING ALL "THE" BY "HIS"
        AFTER INITIAL "BITTEN"
        BEFORE INITIAL "."
    DISPLAY "After: " Inspect-Target
.
```

When executed, this code produces the following console output (the change made by the **INSPECT** is **highlighted**):

```
Before: THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG AND WAS BITTEN ON THE TAIL. THE FOX YELPED!
After:  THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG AND WAS BITTEN ON HIS TAIL. THE FOX YELPED!
```

6.2.25. MERGE

Figure 6-68 - MERGE Syntax

```

MERGE sort-file-1
  { ON { ASCENDING
        DESCENDING } KEY identifier-1 ... } ...
  [ WITH DUPLICATES IN ORDER ]
  [ COLLATING SEQUENCE IS alphabet-name-1 ]
  USING file-name-1 file-name-2 ...
  { GIVING file-name-3 ...
    OUTPUT PROCEDURE IS procedure-name-1 [ THRU|THROUGH procedure-name-2 ] }

```

The **MERGE** statement merges two or more files that have each been pre-sorted on a set of specified identical keys.

1. The *sort-file-1* named on the **MERGE** statement must be defined using a sort description (**SD**) in the **FILE SECTION** of the **DATA DIVISION**. This file is referred to in the remainder of this discussion as the “*merge work file*”.
2. *file-name-1*, *file-name-2* and *file-name-3* (if specified) must reference **ORGANIZATION LINE SEQUENTIAL** or **ORGANIZATION RECORD BINARY SEQUENTIAL** files. These files must be defined using a file description (**FD**) in the **FILE SECTION** of the **DATA DIVISION**.
3. The *identifier-1* ... field(s) must be defined as field(s) within a record of *sort-file-1*.
4. The **WITH DUPLICATES IN ORDER** clause is supported for compatibility purposes with other versions of COBOL, but is non-functional in GNU COBOL.

While any COBOL implementation’s **SORT** or **MERGE** facilities guarantee that records with duplicate key values will be in proper sequence with regard to other records with different key values, they generally make no promises as to the resulting relative sequence of records having duplicate key values with one another.

Some COBOL implementations provide this optional clause to force their **SORT** and **MERGE** facilities to retain duplicate key-value records in their original input sequence, relative to one another.

GNU COBOL always behaves as if the **WITH DUPLICATES IN ORDER** clause is specified, even if it isn’t.

5. The record descriptions of *file-name-1*, *file-name-2*, *file-name-3* (if any) and *sort-file-1* are assumed to be identical in layout and size. While the actual data names used for fields in these files’ records may differ, the structure of records, **PICTURE** of fields, size of fields and **USAGE** of data should match field-by-field across all files.

A common programming technique when using the **MERGE** statement is to define the records of all files involved on the **MERGE** as simple elementary items of the form “**01** *record-name* **PIC X(n)**.” where *n* is the record size. The only file where records are actually described in detail would then be *sort-file-1*.

6. The following rules apply to the files named on the **USING** clause:
 - a. None of them may be **OPEN** at the time the **MERGE** is executed.
 - b. Each of those files is assumed to be already sorted according to the specifications set forth on the **MERGE** statement’s **KEY** clause.
 - c. No two of those files may be referenced on a **SAME RECORD AREA**, **SAME SORT AREA** or **SAME SORT-MERGE AREA** statement specified in the **I-O-CONTROL** paragraph.
7. As the **MERGE** begins execution, the first record in each of the **USING** files is read automatically.
8. As the **MERGE** statement executes, the current record from each of the **USING** files is examined and compared to each other according to the rules set forth by the **KEY** clause. The record that should be “next” in sequence (according to **KEY**) will be written to the merge work file and the **USING** file from which that record came will be read so that its next record is available. As end-of-file conditions are reached on **USING** files, those files will be excluded from further **MERGE** processing – processing continues with the remaining **USING** files until all **USING** files have been completely processed.

9. Once the merge work file has been populated, the merged data will be written to *file-name-3* if the **GIVING** clause was specified, or will be processed by utilizing an **OUTPUT PROCEDURE**.
10. When **GIVING** is specified, none of the *file-name-3* ... files can be **OPEN** at the time the **MERGE** is executed.
11. When an **OUTPUT PROCEDURE** is used, the procedure(s) specified on the **OUTPUT PROCEDURE** clause will be invoked as if by a procedural **PERFORM** statement with no **VARYING** or **UNTIL** options specified. Merged records may be read from the merge work file – one at a time – within the **OUTPUT PROCEDURE** using the **RETURN** statement.
12. A **GO TO** statement that transfers control out of the **OUTPUT PROCEDURE** will terminate the **MERGE** but allows the program to continue executing from the point where the **GO TO** transferred control to. Once an **OUTPUT PROCEDURE** has been aborted using a **GO TO** it cannot be resumed, and the contents of the merge work file are lost. You may, however, re-execute the **MERGE** statement itself. **USING A "GO TO" TO PREMATURELY TERMINATE A MERGE, OR RE-STARTING A PREVIOUSLY-CANCELLED MERGE IS NOT CONSIDERED GOOD PROGRAMMING STYLE AND SHOULD BE AVOIDED.**
13. An **OUTPUT PROCEDURE** is terminated in the same way a procedural **PERFORM** would be. Usually, this action will be taken once the **RETURN** statement indicates that all records in the merge work file have been processed, but termination could occur at any time if required. Once the **OUTPUT PROCEDURE** terminates, the output phase – and the **MERGE** statement itself - is complete.
14. Neither a Format-1 **SORT** nor another **MERGE** may be executed within the scope of the procedures comprising the **OUTPUT PROCEDURE** unless those statements utilize a different sort or merge work file.

See Also...

The I-O-CONTROL Paragraph	4.2.2	The OPEN Statement	6.4.29
Describing the Structure of a File (FD/SD)	5.1	The PERFORM Statement (Procedural)	6.2.30.1
Defining a Data Item's PICTURE	5.2.1.6	The RETURN Statement	6.2.35
Storage Format of Data (USAGE)	5.2.1.11	The SORT Statement (File Sort)	6.4.40.1
The GO TO Statement	6.2.20		

6.2.26. MOVE

6.2.26.1. MOVE Format 1 – Simple MOVE

Figure 6-69 - Simple MOVE Syntax

```
MOVE { literal-1
         identifier-1 } TO identifier-2 ...
```

This statement moves a specific value to one or more receiving data items.

1. The **MOVE** statement will replace the contents of one or more receiving data items (*identifier-2* ...) with a new value – the one specified by *literal-1* or *identifier-1*.
2. Only numeric data can be moved to a numeric *identifier-2*. A **MOVE** involving numeric data will perform any necessary format conversions that might be necessary.
3. If *identifier-1* is specified as the source for a **MOVE**, its contents will not be changed³¹.

6.2.26.2. MOVE Format 2 – MOVE CORRESPONDING

Figure 6-70 - MOVE CORRESPONDING Syntax

```
MOVE CORRESPONDING identifier-1 TO identifier-2 ...
```

This statement moves similarly-named items from one group item to another.

1. The word **CORRESPONDING** may be abbreviated as **CORR**.
2. Both *identifier-1* and *identifier-2* must be group items.
3. When corresponding matches are established, the effect of a **MOVE CORRESPONDING** on those matches will be as if a series of individual **MOVE**s were done – one for each match.

See Also...

The **CORRESPONDING** Clause [6.1.12.2](#)

³¹ Here's an instance where COBOL's strong dependence on the English language can get the inexperienced programmer into trouble – it probably would have been better for generations of beginning COBOL programmers if this verb had been named "COPY" rather than **MOVE**, as the process of **MOVE**ing data from one place to another only affects the data items named after the "TO".

6.2.27. MULTIPLY

6.2.27.1. MULTIPLY Format 1 – MULTIPLY BY

Figure 6-71 - MULTIPLY BY Syntax

```

MULTIPLY { literal-1
             identifier-1 } BY { identifier-2 [ rounding-option ] } ...
      [ size-error-clause ]
[ END-MULTIPLY ]

```

1. *Identifier-1* and *identifier-2* must be numeric unedited data items, each *identifier-3* must be a numeric (edited or unedited) data item and *literal-1* and *literal-2* must be numeric literals.

2. The product of *identifier-1* or *literal-1* and each *identifier-2*, in turn, will be computed and moved to each of the *identifier-2* data items, replacing its old contents.
3. The value of *identifier-1* is not altered.
4. The optional “*rounding-option*” clause available to each *identifier-2* will control how non-integer results will be saved.
5. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-2* is insufficiently sized to hold the generated results; this clause will also detect attempts to divide by zero.

See Also...

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

Rounding Options [6.1.12.7](#)

6.2.27.2. MULTIPLY Format 2 – MULTIPLY GIVING

Figure 6-72 - MULTIPLY GIVING Syntax

```

MULTIPLY { literal-1
             identifier-1 } BY { literal-2
             identifier-2 }
      GIVING { identifier-3 [ rounding-option ] } ...
      [ size-error-clause ]
[ END-MULTIPLY ]

```

1. *Identifier-1* and *identifier-2* must be numeric unedited data items, *identifier-3* should be a numeric or numeric-edited data item and *literal-1* must be a numeric literal.

2. The product of *identifier-1* or *literal-1* and *identifier-2* or *literal-2* will be computed and moved to each of the *identifier-2* data item, replacing the old contents.
3. The optional “*rounding-option*” clause available to each *identifier-3* will control how non-integer results will be saved.
4. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-3* is insufficiently sized to hold the generated results; this clause will also detect attempts to divide by zero.

See Also...

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

Rounding Options [6.1.12.7](#)

6.2.28. NEXT SENTENCE

Figure 6-73 - NEXT SENTENCE Syntax

NEXT SENTENCE

The **NEXT SENTENCE** statement is a means of “breaking out” of a series of nested “**IF**” statements.

1. The **NEXT SENTENCE** statement is valid only when used within the scope of an “**IF**” statement.
2. As its name implies, this statement causes control to transfer to the next sentence in the program.
3. The **NEXT SENTENCE** statement is needed for COBOL programs that are coded according to pre-1985 standards. Programs coded for 1985 (and beyond) standards don’t need it.
4. New GNU COBOL programs should be coded to use the **END-IF** scope terminator for **IF** statements, which invalidates the use of **NEXT SENTENCE** in favor of the **CONTINUE** statement.

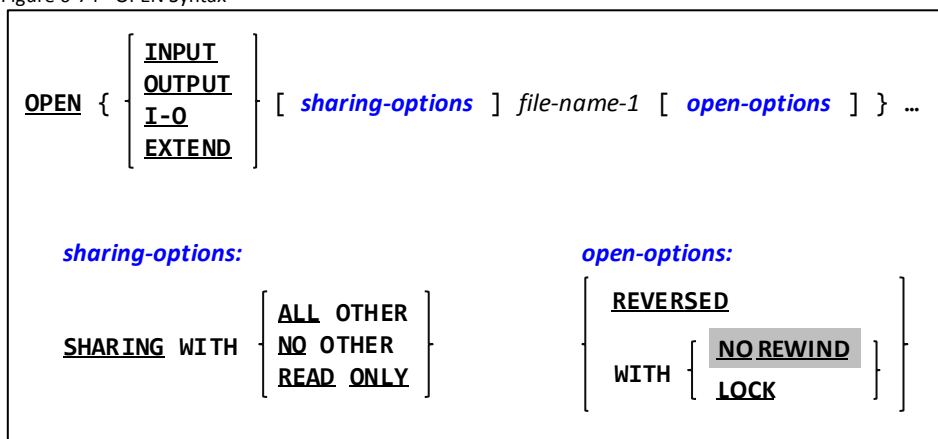
See Also...

Use of Periods (.) [6.1.5](#)

The **CONTINUE** Statement [6.4.10](#)

6.2.29. OPEN

Figure 6-74 - OPEN Syntax



The OPEN statement makes one or more files described in your program available for use.

- Any file defined in a GNU COBOL program must be successfully **OPEN**ed before it or any of its record descriptions may be referenced on a **CLOSE**, **DELETE**, **READ**, **REWRITE**, **START**, **UNLOCK** or **WRITE** statement. Additionally, a file must be successfully **OPEN**ed for any of its record data names (or data elements subordinate to those records) to be referenced on any statement other than a **MERGE** or **SORT**.
- Any attempt to **OPEN** a file that is already **OPEN** will fail with a **FILE STATUS** of 41 ("File Already OPEN"). This is a fatal error that will terminate the program.
- Any **OPEN** failure (including "File Already OPEN") may be trapped using **DECLARATIVES** or an error procedure established using the **CBL_ERROR_PROC** built-in subroutine. When either of these trap routines exit, however, the GNU COBOL runtime system will terminate the program. Ultimately, you cannot recover from an **OPEN** failure.
- The **INPUT**, **OUTPUT**, **I-O** and **EXTEND** options inform GNU COBOL of the manner in which you wish to use the file, as follows:

OPEN Mode	Effect
INPUT	You may only read the existing contents of the file - only the CLOSE , READ , START and UNLOCK statements will be allowed.
OUTPUT	You may only write new content (which will completely replace any previous file contents) to the file - only the CLOSE , UNLOCK and WRITE statements will be allowed.
I-O	You may perform any operation you wish against the file - all file I/O statements will be allowed.
EXTEND	You may only write new content (which will be appended after any previously existing file content) to the file - only the CLOSE , UNLOCK and WRITE statements will be allowed.

- The **SHARING** clause informs GNU COBOL how you are willing to co-exist with any other GNU COBOL programs that may attempt to **OPEN** the same file after your program does.
- The **WITH NO REWIND** option on the OPEN statement is supported syntactically but is otherwise non-functional. Note that the **CLOSE** statement (section 6.2.7) also has this option, which is supported by GNU COBOL.

Devices that would be capable of supporting a **WITH NO REWIND** clause (tape drives) are pretty rare in the environments in which GNU COBOL is intended to operate, and only such a device will be responsive to the **WITH NO REWIND** option.
- The **WITH LOCK** option will be functional only if your GNU COBOL build can support it. GNU COBOL built for MinGW or native Windows will not, because the Unix "fcntl()" primitive doesn't exist in those environments. GNU COBOL built for Cygwin or Unix will.
- The **REVERSED** option will be syntactically accepted, but a compilation specifying either the **"-wobsolete"** or **"-Wall"** options will yield a warning message that **REVERSED** is an obsolete feature.

See Also...

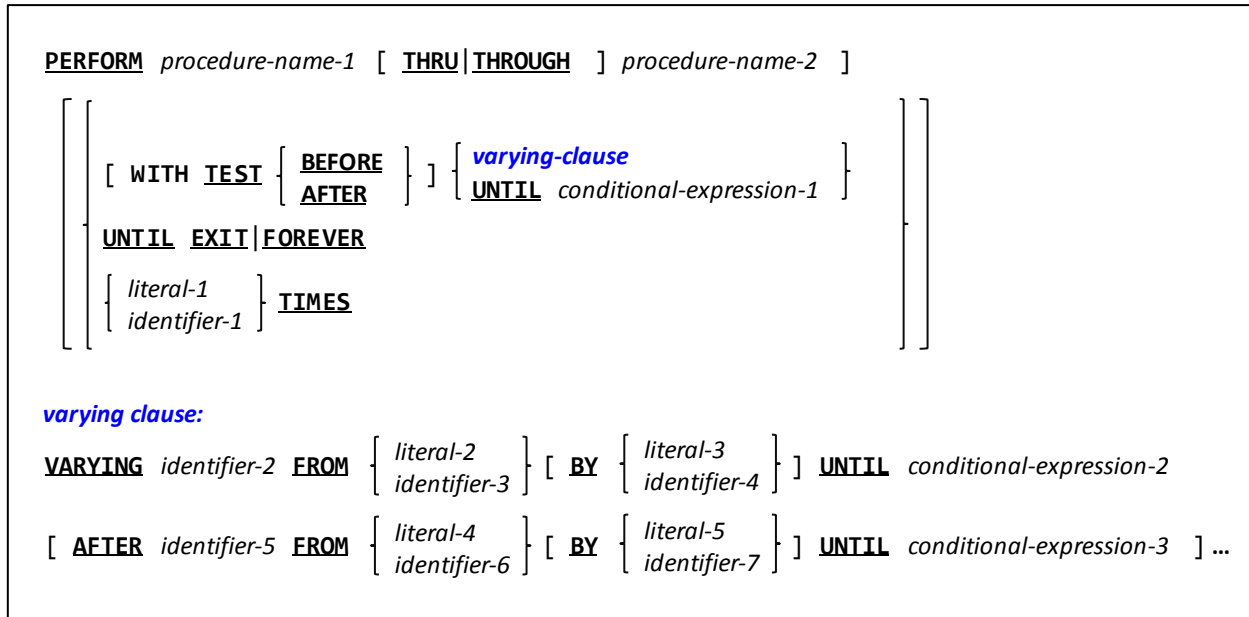
FILE-STATUS Values	Figure 4-15
File Sharing	6.1.9.1
Record Locking	6.1.9.2
Using DECLARATIVES	6.1.4
The CLOSE Statement	6.4.7
The DELETE Statement	6.4.11
The MERGE Statement	6.4.25

The READ Statement	6.4.31
The REWRITE Statement	6.4.36
The SORT Statement (File Sort)	6.4.40.1
The START Statement	6.2.41
The UNLOCK Statement	6.4.48
The WRITE Statement	6.4.50
The CBL_ERROR_PROC Subroutine	8.3.1.24

6.2.30. PERFORM

6.2.30.1. PERFORM Format 1 – Procedural

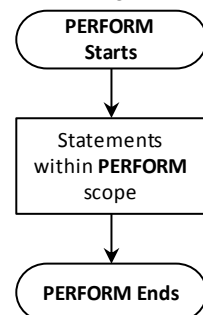
Figure 6-75 - Procedural PERFORM Syntax



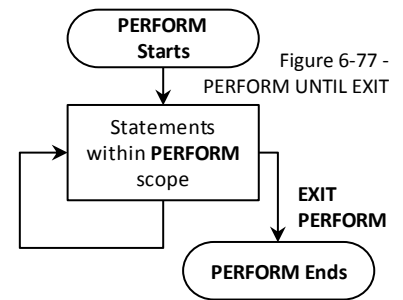
This format of the **PERFORM** statement is used to transfer control to one or more procedures and to return control when execution of the specified procedure(s) is complete. This invocation of the procedure(s) can be done a single time, multiple times, repeatedly until a condition becomes TRUE or forever (with – presumably – some way of breaking out of the control of the **PERFORM** or of halting program execution within the procedure(s)).

1. The words **THROUGH** and **THRU** may be used interchangeably. Both *procedure-name-1* and *procedure-name-2* must be **PROCEDURE DIVISION** sections or paragraphs defined in the same program as the **PERFORM** statement. If *procedure-name-2* is specified, it must follow *procedure-name-1* in the program's source code. The *scope* of the **PERFORM** is defined as being the statements within *procedure-name-1*, the statements within *procedure-name-2* and all statements in all procedures defined between them.
2. All *identifier-n* entries shown must be elementary unedited numeric data items. All *literal-n* entries shown must be numeric literals (or references to functions that return a numeric value).
3. Without the **UNTIL**, **TIMES**, **VARYING** or **FOREVER** clauses, the code within the scope of the **PERFORM** will be executed (once) and control will return to the statement following the **PERFORM**. See [Figure 6-76](#).

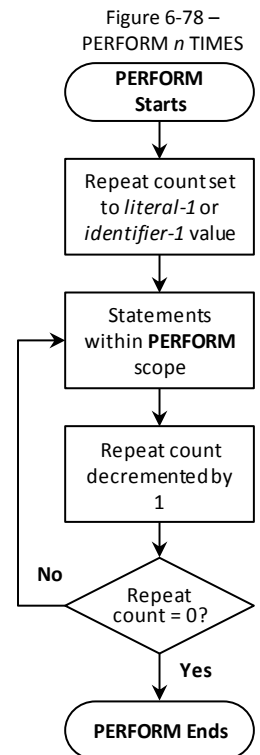
Figure 6-76 - Simple PERFORM



4. The **UNTIL EXIT** option will repeatedly execute the code within the scope of the **PERFORM** with no conditions defined on the **PERFORM** statement itself for termination of the repetition. It will be up to the programmer to include an **EXIT PERFORM** within the scope of the **PERFORM** that will break out of the loop.
5. The **FOREVER** option has the same effect as **UNTIL EXIT**.



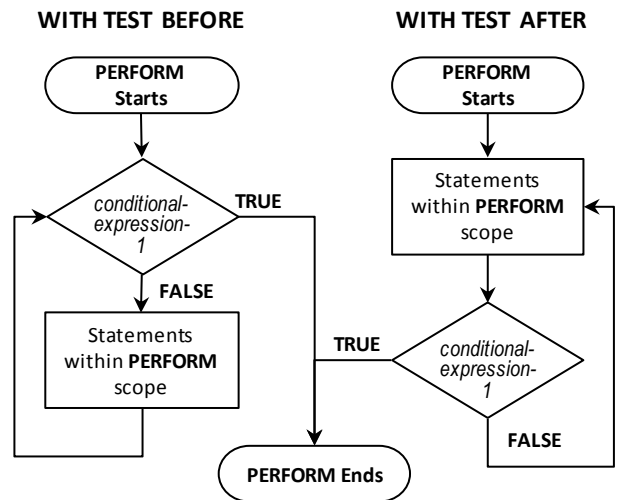
6. The **TIMES** option will repeat the execution of the code within the scope of the **PERFORM** a fixed number of times. When the **PERFORM** statement is executed, the repeat count will be set to the value of *literal-1* or the value within *identifier-1* at the time the **PERFORM** begins execution. Once that number of repetitions has concluded, control will fall into the next statement following the **PERFORM**³².



³² Changing the contents of *identifier-1* within the scope of the **PERFORM** will have no effect on the repetition count, as that was determined the moment the **PERFORM** began executing.

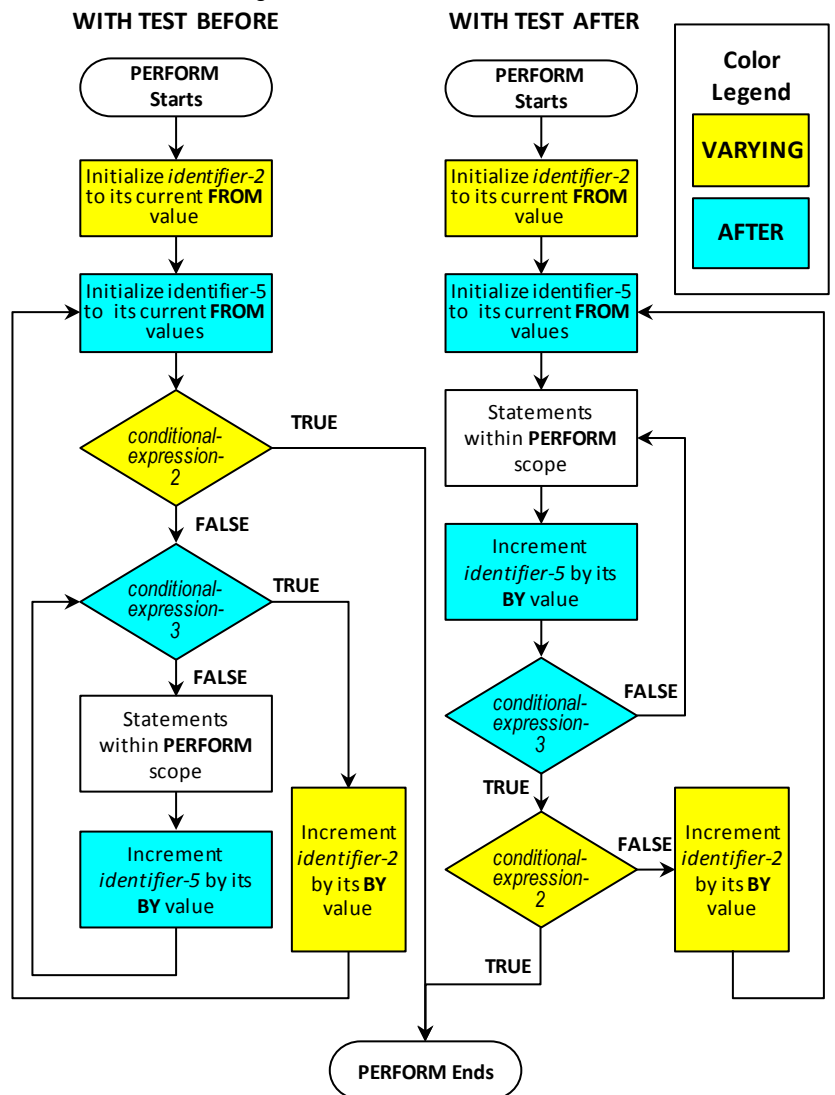
7. The “**UNTIL** *conditional-expression-1*” option will repeat the code within the scope of the **PERFORM** until the specified conditional expression evaluates to a TRUE value.
8. The optional **WITH TEST** clause will control whether **UNTIL** testing occurs **BEFORE** the scope of the **PERFORM** is executed on each iteration or **AFTER**. The default, if no **WITH TEST** clause is specified, is **BEFORE**.

Figure 6-79 - PERFORM UNTIL



9. The **VARYING** clause allows for the definition of a data item (*identifier-2*) that will have a unique numeric value for each iteration of the execution of the statements within the scope of the **PERFORM**.
10. If a **VARYING** clause has been used, you may also use any number of additional **AFTER** clauses to create a secondary loop situation where each **AFTER** will create an additional series of iterations, will define an additional data item to be incremented during each iteration and will define an additional conditional expression to define the termination of that series of iterations. Functionally, this is basically a way of nesting a **PERFORM VARYING** within another **PERFORM VARYING** without the need to code multiple statements.
11. The flowchart in [Figure 6-80](#) shows how **PERFORM VARYING** (with an **AFTER** clause too!) works in both **TEST BEFORE** and **TEST AFTER** modes.

Figure 6-80 - PERFORM VARYING AFTER



Observe the following code which defines a two-dimensional (3 row by 4 column) table and a pair of numeric data items to be used to subscript references to each element of the table:

```

Ø1 PERFORM-DEMO.
  Ø5 PD-ROW          OCCURS 3 TIMES.
    1Ø PD-COL        OCCURS 4 TIMES
      15 PD          PIC X(1).
Ø1 PD-Col-No        PIC 9 COMP.
Ø1 PD-Row-No        PIC 9 COMP.
    
```

PD (1, 1)	PD (1, 2)	PD (1, 3)	PD (1, 4)
PD (2, 1)	PD (2, 2)	PD (2, 3)	PD (2, 4)
PD (3, 1)	PD (3, 2)	PD (3, 3)	PD (3, 4)

Let's say we want to PERFORM a routine (100-Visit-Each-PD) which will – in turn – access each PD data item in the sequence shown to the right. Here's the PERFORM code:

```

PERFORM 100-Visit-Each-PD WITH TEST AFTER
  VARYING PD-Row-No FROM 1 BY 1 UNTIL PD-Row-No = 3
  AFTER PD-Col-No FROM 1 BY 1 UNTIL PD-Col-No = 4.
    
```

1	2	3	4
5	6	7	8
9	10	11	12

1	4	7	10
2	5	8	11
3	6	9	12

But, perhaps you needed to "visit" each PD in the sequence shown to the left. If so, then here's the PERFORM you need:

```

PERFORM 100-Visit-Each-PD WITH TEST AFTER
  VARYING PD-Col-No FROM 1 BY 1 UNTIL PD-Col-No = 4
  VARYING PD-Row-No FROM 1 BY 1 UNTIL PD-Row-No = 3.
    
```

As a general rule of thumb, if you use WITH TEST AFTER on a PERFORM, the termination conditions specified on VARYING and AFTER clauses should test the identifier being varied for being EQUAL TO the maximum value it should receive. If you use WITH TEST BEFORE, the termination conditions specified on VARYING and AFTER clauses should test the identifier being varied for being GREATER THAN the maximum value it should receive.

Thus, the two PERFORM examples shown above could have been coded this way:

```

PERFORM 100-Visit-Each-PD WITH TEST BEFORE
  VARYING PD-Row-No FROM 1 BY 1 UNTIL PD-Row-No > 3
  AFTER PD-Col-No FROM 1 BY 1 UNTIL PD-Col-No > 4.
    
```

- and -

```

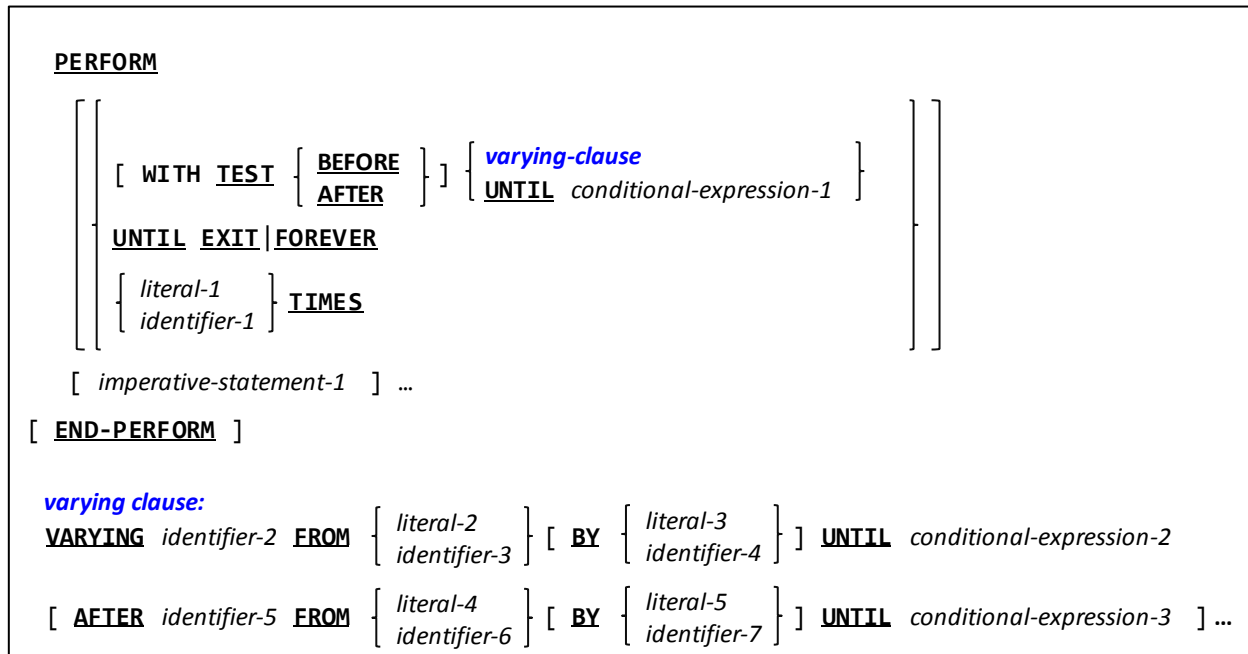
PERFORM 100-Visit-Each-PD WITH TEST BEFORE
  VARYING PD-Col-No FROM 1 BY 1 UNTIL PD-Col-No > 4
  VARYING PD-Row-No FROM 1 BY 1 UNTIL PD-Row-No > 3.
    
```

See Also...

Conditional Expressions [6.1.8.2](#)

6.2.30.2. PERFORM Format 2 – Inline

Figure 6-81 - Inline PERFORM Syntax



This format of the **PERFORM** statement is identical in operation to format 1, except for the fact that the statements that comprise the scope of the **PERFORM** are now specified in-line with the **PERFORM** code rather than in procedures located elsewhere within the program.

1. The various optional clauses have the same use and effect as in format 1 of the **PERFORM** statement.
2. The distinguishing characteristic of this format versus format 1 is that – with this version of the **PERFORM** statement – the code being executed is specified in-line (*imperative-statement-1 ...*) rather than in one or more separate procedures.

6.2.31. READ

6.2.31.1. READ Format 1 – Sequential READ

Figure 6-82 – READ (Sequential) Syntax

```

READ file-name-1 [ { NEXT
                     { PREVIOUS } ] RECORD
  [ INTO identifier-1 ]
  [ { IGNORING LOCK
    { WITH LOCK
    { WITH KEPT LOCK
    { WITH NO LOCK
    { WITH IGNORE LOCK
    { WITH WAIT } } } } ]
  [ at-end-clause ]

[ END-READ ]

```

This form of the READ statement retrieves the next (or previous) record from a file.

1. *File-name-1* must currently be **OPEN** for **INPUT** or **I-O**.
2. If the **ACCESS MODE** of *file-name-1* is **RANDOM**, this format of the **READ** statement cannot be used.
3. If the **ACCESS MODE** is **SEQUENTIAL**, this is the only format of READ that is available.
4. If the **ACCESS MODE** is **DYNAMIC**, this format of the **READ** statement may be used as well as format 2. The following minimalist **READ** statement...

```
READ file-name-1
```

...is perfectly legal according to both **READ** formats. For that reason, when **ACCESS MODE DYNAMIC** has been specified and you want to tell the GNU COBOL compiler that a statement such as the one above should be treated as a sequential **READ**, you must add either **NEXT** or **PRIOR** to the statement (otherwise it will be treated as a random **READ**).

5. The keywords **NEXT** and **PREVIOUS** specify in what direction of travel the reading process will take through the file. If neither **NEXT** nor **PREVIOUS** clause is specified, **NEXT** is assumed.
6. The **PREVIOUS** option is available only for **ORGANIZATION INDEXED** files.
7. A successful sequential **READ** will retrieve the next available record from *file-name-1*, in either a “next” or “previous” direction from the most-recently-READ record, depending upon the use of the **NEXT** or **PREVIOUS** option. The newly-retrieved record data will be saved into the 01-level record structure(s) that immediately follow the file’s **FD** or **SD**. If the optional **INTO** clause is present, a copy of the just-retrieved record will be automatically **MOVED** to *identifier-1*.
8. The optional LOCK options may be used to control access to the file by other programs while this program is running.
9. The optional *at-end-clause* may be used to detect situations where all records in a file have been processed (known as an end-of-file condition). Without using one of these clauses, a program would need to test the returned **FILE STATUS** value after each **READ**.

See Also...

Types of Files	1.3.3.5	Describing the Structure of a File (FD/SD)	5.1
Defining File Characteristics (SELECT)	4.2.1	Record Locking	6.1.11.2
FILE-STATUS Values	Figure 4-15	Handling End-of-File Conditions (AT END)	6.1.12.1
		The OPEN Statement	6.4.29

6.2.31.2. READ Format 2 – Random Read

Figure 6-83 - READ (Random) Syntax

```

READ file-name-1 RECORD

  [ INTO identifier-1 ]

  [ [ IGNORING LOCK
    WITH LOCK
    WITH KEPT LOCK
    WITH NO LOCK
    WITH IGNORE LOCK
    WITH WAIT ] ]

  [ KEY IS identifier-2 ]

  [ invalid-key-clause ]

[ END-READ ]

```

This form of the **READ** statement retrieves an arbitrary record from a **ORGANIZATION RELATIVE** or **ORGANIZATION INDEXED** file.

1. *File-name-1* must currently be **OPEN** for **INPUT** or **I-O**.
2. If the **ACCESS MODE** of *file-name-1* is **SEQUENTIAL**, this format of the **READ** statement cannot be used.
3. If the **ACCESS MODE** is **RANDOM**, this is the only format of **READ** that is available.
4. If the **ACCESS MODE** is **DYNAMIC**, this format of the **READ** statement may be used as well as format 1. The following minimalist **READ** statement...

```
READ file-name-1
```

...is perfectly legal according to both **READ** formats. For that reason, when **ACCESS MODE DYNAMIC** has been specified for a file, a **READ** statement such as the above will be automatically treated as a random READ.

5. The optional **KEY** clause tells the compiler how a record is to be located in the file.

If the **KEY** clause is absent:

If the file is an **ORGANIZATION RELATIVE** file, the contents of the field declared as the file's **RELATIVE KEY** will be used to identify a record. If the file is an **ORGANIZATION INDEXED** file, the contents of the field declared as the file's **RECORD KEY** (section will be used to identify a record.

If the **KEY** clause is specified:

If the file is an **ORGANIZATION RELATIVE** file, the contents of *identifier-2* will be used as the relative record number of the record to be accessed. *Identifier-2* does not have to be the **RELATIVE KEY** field of the file (although it could be if you wish). If the file is an **ORGANIZATION INDEXED** file, *identifier-2* must be the **PRIMARY RECORD KEY** or one of the file's **ALTERNATE RECORD KEY** fields (if any) – the current contents of that field will identify the record to be accessed. If an alternate record key is used, and that key allows duplicate values, the record accessed will be the 1st one having that key value.

6. The record identified by rule #5 will be retrieved from *file-name-1*. The newly-retrieved record data will be saved into the 01-level record structure(s) that immediately follow the file's **FD**. If the optional **INTO** clause is present, a copy of the just-retrieved record will be automatically **MOVED** to *identifier-1*.
7. The optional **LOCK** options may be used to control access to the file by other programs while this program is running.
8. The optional *invalid-key-clause* may be used to detect situations where the desired record cannot be read from the file (most likely because no record exists with the specified **RELATIVE KEY** or **RECORD KEY**). Without using one of these clauses, a program would need to test the returned **FILE STATUS** value after each **READ**.

See Also...

Types of Files	1.3.3.5	Describing the Structure of a File (FD/SD)	5.1
Defining File Characteristics (SELECT)	4.2.1	Handling Invalid Keys (INVALID KEY)	6.1.12.3
FILE-STATUS Values	Figure 4-15	The OPEN Statement	6.4.29

6.2.32. READY TRACE

Figure 6-84 - READY TRACE Syntax

READY TRACE

The **READY TRACE** verb turns procedure or procedure+statement tracing on.

1. This statement will cause procedure or procedure+statement tracing to be turned on.
2. In order for this statement to be functional, tracing code must have been generated into the compiled program using either the “**-ftrace**” (procedures only) or “**-ftraceall**” (procedures + statements) compiler options.
3. Tracing may be turned off at any point by executing the **RESET TRACE** statement (section).
4. See the **COB_SET_TRACE** environment variable for another way to control tracing.

See Also...

The **RESET TRACE** Statement [6.4.34](#)

Execution-time Environment Variables [8.2.4](#)

Compiler Switches Reference [8.1.2](#)

6.2.33. RELEASE

Figure 6-85 - RELEASE Syntax

```
RELEASE record-name-1 [ FROM { literal-1  
                                identifier-1 } ]
```

The **RELEASE** statement adds a new record to a *sort work file*.

1. The **RELEASE** statement is valid only within the **INPUT PROCEDURE** of a **SORT** statement.
2. *Record-name-1* must be a record defined to a sort description (**SD**) entry.

See Also...

Describing the Structure of a File (**FD/SD**) [5.1](#)

The **SORT** Statement (File Sort) [6.4.40.1](#)

6.2.34. RESET TRACE

Figure 6-86 - RESET TRACE Syntax

RESET TRACE

The **RESET TRACE** verb turns procedure or procedure+statement tracing off.

1. This statement will cause procedure or procedure+statement tracing to be turned off.
2. By default, procedure and procedure+statement tracing is OFF as programs begin execution. Use the **READY TRACE** statement (section to turn tracing on).
3. In order for this statement to be functional, tracing code must have been generated into the compiled program using either the “-ftrace” (procedures only) or “-ftraceall” (procedures + statements) compiler options.
4. See the **COB_SET_TRACE** environment variable for another way to control tracing.

See Also...

The **READY TRACE** Statement [6.2.32](#)

Compiler Switches Reference [8.1.2](#)

Execution-time Environment Variables [8.2.4](#)

6.2.35. RETURN

Figure 6-87 - RETURN Syntax

```

RETURN sort-file-name-1 RECORD
  [ INTO identifier-1 ]
  [ at-end-clause ]
  [ END-RETURN ]

```

The **RETURN** statement reads a record from a sort- or merge work file.

1. The **RETURN** statement is valid only within the **OUTPUT PROCEDURE** of a **SORT** or **MERGE** statement.
2. *Sort-file-name-1* must be a sort- or merge work file defined with a sort description (**SD**) entry.
3. A successful **RETURN** will retrieve the next available record from *sort-file-name-1*. The newly-retrieved record data will be saved into the 01-level record structure(s) that immediately follow the file's **SD**. If the optional **INTO** clause is present, a copy of the just-retrieved record will be automatically **MOVED** to *identifier-1*.
4. The optional *at-end-clause* may be used to detect situations where all sorted records have been **RETURNed** (known as an end-of-file condition). Without using one of these clauses, a program would need to test the returned **FILE STATUS** value after each **RETURN**.

See Also...

Describing the Structure of a File (FD/SD)	5.1
Handling End-of-File Conditions (AT END)	6.1.12.1
The MERGE Statement	6.4.25

The MOVE Statement	6.2.26
The SORT Statement (File Sort)	6.4.40.1

6.2.36. REWRITE

Figure 6-88 - REWRITE Syntax

```

REWRITE record-name-1
  [ FROM { literal-1
             identifier-1 } ]
  [ { WITH LOCK
      WITH NO LOCK } ]
  [ invalid-key-clause ]
[ END-REWRITE ]

```

The **REWRITE** statement replaces a logical record on a disk file.

1. *Record-name-1* must be defined as an 01-level record subordinate to the File Description of a file that is currently **OPEN** for **I-O**.
2. The optional **FROM** clause will cause *literal-1* or *identifier-1* to be implicitly **MOVED** into *record-name-1* prior to writing *record-name-1* to the file.
3. The **REWRITE** statement may not be used with **ORGANIZATION IS LINE SEQUENTIAL** files.
4. If the optional **LOCK** clause is omitted, the effect will be as if **WITH NO LOCK** was coded – that is, the rewritten record will not be locked against access by other programs.
5. Rewriting a record does not cause the record contents of the file to be physically updated until the next block of the file is read, a **COMMIT** or **UNLOCK** statement is issued or that file is **CLOSED**.
6. If the file has **ORGANIZATION RECORD BINARY SEQUENTIAL**:
 - a. The record to be rewritten will be the one retrieved by the most-recently executed **READ** of the file.
 - b. If the FD of the file contains the **RECORD CONTAINS / RECORD IS VARYING** clause and it allows record size to vary, the size of *record-name-1* cannot be altered.
7. If the file has **ORGANIZATION RELATIVE** or **ORGANIZATION INDEXED**:
 - a. If the file has **ACCESS MODE SEQUENTIAL**, the record to be rewritten will be the one retrieved by the most-recently executed **READ** of the file. If the file has **ACCESS MODE RANDOM** or **ACCESS MODE DYNAMIC**, no **READ** is required before a record may be rewritten – the **RELATIVE KEY / RECORD KEY** definition for the file will specify the record to be updated.
 - b. The size of *record-name-1* may be updated.
8. The optional *invalid-key-clause* allows the program to detect and recover from attempts to rewrite non-existent records.

See Also...

Types of Files	1.3.3.5	The COMMIT Statement	6.4.8
Describing the Structure of a File (FD/SD)	5.1	The MOVE Statement	6.2.26
Record Locking	6.1.9.2	The OPEN Statement	6.4.29
Handling Invalid Keys (INVALID KEY)	6.1.12.3	The READ Statement	6.4.31
The CLOSE Statement	6.4.7	The UNLOCK Statement	6.4.48

6.2.37. ROLLBACK

Figure 6-89 - ROLLBACK Syntax

ROLLBACK

The **ROLLBACK** verb reverts changes made to all files since the start of the program or since the last **COMMIT**.

1. GNU COBOL does not (currently, at least) support file rollback. The GNU COBOL **ROLLBACK** statement will have the same effect as the **COMMIT** verb.

See Also...

The **COMMIT** Statement [6.4.8](#)

6.2.38. SEARCH

6.2.38.1. SEARCH Format 1 – Sequential Search

Figure 6-90 - Sequential SEARCH Syntax

```

SEARCH table-name-1
  [ VARYING index-name-1 ]
  [ AT END imperative-statement-1 ]
  { WHEN conditional-expression-1 imperative-statement-2 } ...
[ END-SEARCH ]

```

The **SEARCH** statement is used to sequentially search a table, stopping either once a specific value is located within the table or when the table has been completely searched.

1. The *index-name-1* identifier specified on the **VARYING** clause must be **USAGE INDEX**.
2. If no **VARYING** clause is specified, then the table being searched must have been created with an **INDEXED BY** clause.
3. At the time the **SEARCH** statement is executed, the current value of *index-name-1* (or the table's defined **INDEXED BY** index if no **VARYING** clause is specified) will define the starting position in the table where the searching process will begin. Typically, one initializes that index to a value of 1 before starting the **SEARCH**, as follows:


```

SET index-name-1 TO 1

```
4. During the searching process, the *conditional-expression-1* will be evaluated and – if TRUE – will cause *imperative-statement-2* to be executed, after which control will fall into the next statement after the **SEARCH**.
5. If multiple **WHEN** clauses exist, each *conditional-expression-n* will be evaluated in-turn and the first one that evaluates to TRUE will cause the corresponding *imperative-statement-n* to be executed, after which control will fall into the next statement after the **SEARCH**.
6. If no *conditional-expression-n* evaluates to TRUE, the value of *index-name-1* will be incremented to point to the next entry in the table. If the value of *index-name-1* is still within the **OCCURS** scope of *table-name-1*, the **WHEN** clause(s) will again be re-evaluated. This process will continue until a **WHEN** clause *conditional-expression-n* evaluates to TRUE or until the value of *index-name-1* is no longer within the **OCCURS** scope of *table-name-1*.
7. If no *conditional-expression-n* ever evaluates to TRUE and the value of *index-name-1* is no longer within the **OCCURS** scope of *table-name*, the *imperative-statement-1* which is part of the **AT END** clause will be executed. After this, control will fall into the next statement following the **SEARCH**. If there is no **AT END** clause, control simply falls into the next statement following the **SEARCH**.

See Also...

Defining Tables (**OCCURS**) 0

Storage Format of Data (**USAGE**) [5.2.1.11](#)

The **SET index** Statement [6.2.39.4](#)

6.2.38.2. SEARCH Format 2 – Binary, or Half-interval Search (SEARCH ALL)

Figure 6-91 - Binary SEARCH (ALL) Syntax

```

SEARCH ALL table-name-1
  [ AT END imperative-statement-1 ]
  WHEN conditional-expression-1 imperative-statement-2
  [ END-SEARCH ]

```

This format of the **SEARCH** statement performs a binary, or half-interval, search against a sorted table.

1. The definition of *table-name-1* must include the **OCCURS**, **ASCENDING** (and/or **DESCENDING**) **KEY** and **INDEXED BY** clauses.
2. In order for a table to be searchable via the **SEARCH ALL** statement, each of the following must be true:
 - a. The table meets the requirements of rule #1 above.
 - b. Just because the table has one or more **KEY** clauses doesn't mean the data is actually in that sequence in the table – the actual sequence of the data must agree with the **KEY** clause(s)!³³
 - c. No two records in the table may have the same **KEY** field values. If the table has multiple **KEY** definitions, then no two records in the table may have the same combination of **KEY** field values.

If rule "a" is violated, the compiler will reject the **SEARCH ALL**. If rules "b" and/or "c" are violated, there will be no message issued by the compiler, but the run-time results of a **SEARCH ALL** against the table will probably be incorrect.

3. The *conditional-expression-1* should involve the **KEY** fields, using the table's **INDEXED BY** index name as a subscript.
4. The **WHEN** clause is mandatory, unlike format 1 of the **SEARCH** statement.
5. There can only be one **WHEN** clause specified.
6. The function of the **WHEN** is to compare the key field(s) of the table, as indexed by the table's **INDEXED BY** index data item, against whatever literal and/or identifier values you are searching for in order to locate the desired entry in the table. The table's index will be automatically varied by the **SEARCH ALL** statement in a manner designed to require the minimum number of tests.
7. The internal processing of the **SEARCH ALL** statement begins by setting internal "first" and "last" pointers to the 1st and last entry locations of the table. Processing then proceeds as follows³⁴:
 - a. The entry half-way between "first" and "last" is identified. We'll call this the "current" entry, and will set its table entry location into *index-name-1*.
 - b. The **WHEN** is evaluated. This comparison of the key(s) against the target literal/identifier values will have one of three possible outcomes:
 - i. If the key(s) and value(s) match, *imperative-statement-2* is executed, after which control falls thru into the next statement following the **SEARCH ALL**.
 - ii. If the key(s) are *LESS THAN* the value(s), then the table entry being searched for can only occur in the "current" to "last" range of the table, so a new "first" pointer value is set (it will be set to the "current" pointer).
 - iii. If the key(s) are *GREATER THAN* the value(s), then the table entry being searched for can only occur in the "first" to "current" range of the table, so a new "last" pointer value is set (it will be set to the "current" pointer).
 - c. If the new "first" and "last" pointers are different than the old "first" and "last" pointers, there's more left to be searched, so return to step "a" and continue.

³³ Of course, if the data sequence doesn't agree with the **KEY** clause, you can easily make it that way using a table **SORT**

³⁴ This is a simplified view of the algorithm intended purely as a pedagogical tool – an actual implementation of it requires a few additional picky little details to make it work (such as what to do when rule "a" identifies a "current" entry of 12.5!)

- d. If the new “first” and “last” pointers are the same as the old “first” and “last” pointers, the table has been exhausted and the entry being searched for cannot be found; *imperative-statement-1* is executed, after which control falls thru into the next statement following the **SEARCH ALL**. If there is no **AT END** clause coded, control simply falls into the next statement following the **SEARCH ALL**.

The net effect of the above algorithm is that only a fraction of the number of elements in the table need ever be tested in order to decide whether or not a particular entry exists. This is because the **SEARCH ALL** discards half the remaining entries in the table each time it checks an entry.

Computer scientists will compare these two search techniques as follows:

- ▶ A sequential search (format 1) will need an average of $n/2$ tests and a worst case of n tests in order to find an entry and n tests to identify that an entry doesn't exist (n = the number of entries in the table).
- ▶ A binary search (format 2) will need worst case of $\log_2 n$ tests in order to find an entry and $\log_2 n$ tests to identify that an entry doesn't exist (n = the number of entries in the table).

Here's a more practical view of the difference. Let's say that a table has 1,000 entries in it. With a sequential (format 1) search, on average, you'll have to check 500 of them to find an entry and you'll have to look at all 1,000 of them to find that an entry doesn't exist. With a binary search, express the number of entries as a binary number ($1,000_{10} = 1111101000_2$) and count the number of digits in the result (10) -THAT is the worst-case number of tests required to find an entry or to identify that it doesn't exist. That's quite an improvement!

See Also...

Defining Tables (OCCURS)	0
Storage Format of Data (USAGE)	5.2.1.11

The SORT Statement (Table Sort)	6.4.40.2
----------------------------------------	--------------------------

6.2.39. SET

6.2.39.1. SET Format 1 – SET ENVIRONMENT

Figure 6-92 - SET ENVIRONMENT Syntax

```
SET ENVIRONMENT { literal-1 } TO { literal-2 }
                { identifier-1 }
```

A **SET ENVIRONMENT** statement provides a straight-forward means of setting environment values from within a program.

1. Environment variables created or changed from within GNU COBOL programs will be available to any sub-shell processes spawned by that program (i.e. **CALL "SYSTEM"**) but will not be known to the shell or console window that started the GNU COBOL program.
2. This is a much simpler and more readable means of setting environment variables than by using the **DISPLAY** statement. For example, these two code sequences produce identical results:

```
DISPLAY                                SET ENVIRONMENT "VARNAME" TO "VALUE"
  "VARNAME" UPON ENVIRONMENT-NAME
END-DISPLAY
DISPLAY
  "VALUE" UPON ENVIRONMENT-VALUE
END-DISPLAY
```

See Also...

The **DISPLAY** Statement (Environment) [6.2.12.3](#)

6.2.39.2. SET Format 2 – SET Program-Pointer

Figure 6-93 - SET Program Pointer Syntax

```
SET program-pointer-1 TO ENTRY { literal-1 }
                                     { identifier-1 }
```

This form of **SET** allows you to retrieve the address of a **PROCEDURE DIVISION** code module – specifically a declared entry-point into the **PROCEDURE DIVISION**.

1. If you have used other versions of COBOL before (particularly mainframe implementations), you've possibly seen subroutine **CALLS** made passing a **PROCEDURE DIVISION** paragraph or **SECTION** name as an argument – that is not possible in GNU COBOL; instead, you need to know how to use this form of the **SET** statement.
2. The **USAGE** of *program-pointer-1* must be **PROGRAM-POINTER**.
3. The *literal-1* or *identifier-1* value specified must name a primary entry-point name (**PROGRAM-ID** of a subroutine or **FUNCTION-ID** of a user-defined function) or an alternate entry-point defined via an **ENTRY** statement within a subprogram.
4. Once the address of a **PROCEDURE DIVISION** code area has been acquired in this way, the address could be passed to a subroutine (usually written in C) for whatever use it needs it for. For examples of **PROGRAM-POINTERS** at work, see the discussions of the **CBL_ERROR_PROC** and **CBL_EXIT_PROC** built-in subroutines.

See Also...

Storage Format of Data (**USAGE**) [5.2.1.11](#)

The **ENTRY** Statement [6.2.14](#)

The **CBL_ERROR_PROC** Subroutine [8.3.1.24](#)

The **CBL_EXIT_PROC** Subroutine [8.3.1.25](#)

6.2.39.3. SET Format 3 – SET ADDRESS

Figure 6-94 - SET ADDRESS Syntax

```
SET [ ADDRESS OF ] { pointer-name-1 } ...
                { identifier-1 }
TO [ ADDRESS OF ] { pointer-name-2 }
                { identifier-2 }
```

This form of the **SET** statement can be used to work with the addresses of data items rather than their contents.

1. When the **ADDRESS OF** clause is used before the **TO** you will be using the **SET** to alter the address of a **LINKAGE SECTION** or **BASED** data item. Without that clause you will be assigning an address to one or more **USAGE POINTER** data items.

- When the **ADDRESS OF** clause is used after the **TO**, **SET** will be identifying the address of *identifier-2* as the address to be assigned to *identifier-1* or stored in *pointer-name-1*. If the “**ADDRESS OF**” clause is absent after the **TO**, the contents of *pointer-name-2* will serve as the address to be assigned.

See Also...

The **DATA DIVISION** 5

Storage Format of Data (**USAGE**) [5.2.1.11](#)

Dynamically Allocated Items (**BASED**) [5.2.1.2](#)

6.2.39.4. SET Format 4 – SET Index

Figure 6-95 - SET Index Syntax

```
SET index-name-1 TO { literal-1
                    identifier-1 }
```

This **SET** statement assigns a value to a **USAGE INDEX** data item.

- The **USAGE** of *index-name-1* should be **INDEX**, or *index-name-1* must be identified in a table **INDEXED BY** clause.

See Also...

Defining Tables (**OCCURS**) 0

Storage Format of Data (**USAGE**) [5.2.1.11](#)

6.2.39.5. SET Format 5 – SET UP/DOWN

Figure 6-96 - SET UP/DOWN Syntax

```
SET identifier-1 { UP
                 DOWN }
BY [ LENGTH OF ] { literal-1
                  identifier-2 }
```

This format of **SET** is used to increment or decrement the value of an index or pointer by a specified amount.

- The **USAGE** of *identifier-1* must be **INDEX**, **POINTER** or **PROGRAM-POINTER**.
- The typical usage when *identifier-1* is a **USAGE INDEX** data item is to increment it's value **UP** or **DOWN** by 1, since an **INDEX** is usually being used to sequentially walk through the elements of a table.

See Also...

Defining Tables (**OCCURS**) 0

Storage Format of Data (**USAGE**) [5.2.1.11](#)

6.2.39.6. SET Format 6 – SET Condition Name

Figure 6-97 - SET Condition Name Syntax

```
SET { condition-name-1 } ... TO { TRUE
                                FALSE }
```

This format provides one method of specifying the TRUE / FALSE value of a level-88 condition name.

- By setting the specified condition name(s) to a **TRUE** or **FALSE** value, you will actually be assigning a value to the parent data item(s) to which the condition name data item(s) is subordinate to.
- When specifying **TRUE**, the value assigned to each parent data item will be the first **VALUE** specified on the condition name's definition.
- When specifying **FALSE** on the **SET**, the value assigned to each parent data item will be the value specified for the **FALSE** clause of the condition name's definition; if any *condition-name-1* occurrence lacks a **FALSE** clause, the **SET** statement will be rejected by the compiler.

See Also...

Defining Level-88 Condition Names [5.2.7](#)

6.2.39.7. SET Format 7 – SET Switch

Figure 6-98 - SET Switch Syntax

```
SET { mnemonic-name-1 } ... TO { ON
    OFF }
```

Use this **SET** statement type to turn a switch **ON** or **OFF**.

1. Switches are defined using the **SPECIAL-NAMES** paragraph.
2. Switches may be tested via the **IF** statement and a *switch-status condition*.

See Also...

The **SPECIAL-NAMES** Paragraph [4.1.4](#)

Switch-Status Conditions [6.1.4.2.4](#)

The **IF** Statement [6.2.21](#)

6.2.39.8. SET Format 8 – SET ATTRIBUTE

Figure 6-99 - SET ATTRIBUTE Syntax

```
SET identifier-1 ATTRIBUTE { BELL
    BLINK
    HIGHLIGHT
    LEFTLINE
    LOWLIGHT
    OVERLINE
    REVERSE-VIDEO
    UNDERLINE } { ON
    OFF }
```

The **SET ATTRIBUTE** statement may be used to modify one or more attributes of a **SCREEN SECTION** data item at run-time.

1. When making an attribute change to *identifier-1*, the change will not become visible on the screen until the **SCREEN SECTION** data item containing *identifier-1* is next **ACCEPT**ed (if *identifier-1* is an input field) or is next **DISPLAY**ed (if *identifier-1* is not an input field).

See Also...

Defining Screens [5.2.2](#)

The **ACCEPT** Statement (Screen Data) [6.4.1.4](#)

The **DISPLAY** Statement (Screen Data) [6.4.12.4](#)

6.2.40. SORT

6.2.40.1. SORT Format 1 – File-based Sort

Figure 6-100 - File-Based SORT Syntax

```
SORT sort-file-1
{ ON { ASCENDING
    DESCENDING } KEY identifier-1 ... } ...
[ WITH DUPLICATES IN ORDER ]
[ COLLATING SEQUENCE IS alphabet-name-1 ]
{ USING file-name-1 ...
  INPUT PROCEDURE IS procedure-name-1 [ THRU|THROUGH procedure-name-2 ] }
{ GIVING file-name-2 ...
  OUTPUT PROCEDURE IS procedure-name-3 [ THRU|THROUGH procedure-name-4 ] }
```

This format of the **SORT** statement is designed to sort large volumes of data according to one or more key fields.

1. The *sort-file-1* named on the **SORT** statement must be defined using a sort description (**SD**) in the **FILE SECTION** of the **DATA DIVISION**. This file is referred to as the “sort work file”.

2. If specified, *file-name-1* and *file-name-2* must reference **ORGANIZATION LINE SEQUENTIAL** or **ORGANIZATION RECORD BINARY SEQUENTIAL** files. These files must be defined using a file description (**FD**) in the **FILE SECTION** of the **DATA DIVISION**. The same file(s) may be used for *file-name-1* and *file-name-2*.
3. The *identifier-1* ... field(s) must be defined as field(s) within a record of *sort-file*.
4. The **WITH DUPLICATES IN ORDER** clause is supported for compatibility purposes with other versions of COBOL, but is non-functional in GNU COBOL

While any COBOL implementation's **SORT** or **MERGE** facilities guarantee that records with duplicate key values will be in proper sequence with regard to other records with different key values, they generally make no promises as to the resulting relative sequence of records having duplicate key values with one another.

Some COBOL implementations provide this optional clause to force their **SORT** and **MERGE** facilities to retain duplicate key-value records in their original input sequence, relative to one another.

GNU COBOL always behaves as if the **WITH DUPLICATES IN ORDER** clause is specified, even if it isn't.

5. A sort work file (see #1) is never **OPENed** or **CLOSEd**.
6. The **SORT** statement works in three stages, as follows:

STAGE 1 (the input phase):

- a. The data to be sorted is loaded into the sort file. This is accomplished either by taking the entire contents of the file(s) named on the **USING** clause or by utilizing an **INPUT PROCEDURE** defined as *procedure-name 1* or *procedure-name-1 THRU procedure-name-2*.
- b. When **USING** is specified, *file-name-1* ... must not be **OPEN** at the time the **SORT** is executed.
- c. When an **INPUT PROCEDURE** is used, the procedure(s) specified on the **INPUT PROCEDURE** clause will be invoked as if by a procedural **PERFORM** statement with no **VARYING** or **UNTIL** options specified. Records will be loaded into the sort work file – one at a time – within the **INPUT PROCEDURE** using the **RELEASE** statement.

As data is loaded into the sort file, it is actually being buffered in dynamically-allocated memory. Only if the amount of data to be sorted exceeds the amount of available sort memory (128 MB)³⁵ will actual disk files be allocated and utilized. These “sort work files” will be discussed again shortly.

A **GO TO** statement that transfers control out of the **INPUT PROCEDURE** will terminate the **SORT** but allows the program to continue executing from the point where the **GO TO** transferred control to. Once an **INPUT PROCEDURE** has been aborted using a **GO TO** it cannot be resumed, and the contents of the sort work file are lost. You may, however, re-execute the **SORT** statement itself. **USING A “GO TO” TO PREMATURELY TERMINATE A SORT, OR RE-STARTING A PREVIOUSLY-CANCELLED SORT IS NOT CONSIDERED GOOD PROGRAMMING STYLE AND SHOULD BE AVOIDED.**

An **INPUT PROCEDURE** is terminated in the same way a procedural **PERFORM** would be. Once the **INPUT PROCEDURE** terminates, the input phase is complete.

- d. The scope of the **INPUT PROCEDURE** must not allow a file-based **SORT**, **MERGE** or **RETURN** statement to be executed.

STAGE 2 (the sort phase):

- a. The sort will take place by arranging the data records in the sequence defined by the **ASCENDING KEY** and/or **DESCENDING KEY** specification(s) on the **SORT** statement according to the **COLLATING SEQUENCE** specified on the **SORT** (if any) or – if none was defined – the **PROGRAM COLLATING SEQUENCE** specified or implied by the **OBJECT-COMPUTER** paragraph. Keys may be any supported data type and **USAGE** except for level-78 or level-88 data items.

For example, let's assume we're sorting a series of financial transactions. The **SORT** statement might look like this:

```
SORT Sort-File
      ASCENDING KEY Transaction-Date
      ASCENDING KEY Account-Number
      DESCENDING KEY Transaction-Amount
```

³⁵ There is a runtime environment variable (COB_SORT_MEMORY) that you may use to allocate more or less memory to the sorting process. See section [8.2.4](#).

·
·
·

The effect of this statement will be to sort all transactions into ascending order of the date the transaction took place (oldest first, newest last). Unless the business running this program is going out of business, there are most-likely many transactions for any given date – therefore, within each grouping of transactions all with the same date, transactions will be sub-sorted into ascending sequence of the account number the transactions apply to. Since it's quite possible there might be multiple transactions for an account on any given date, a third level sub-sort will arrange all transactions for the same account on the same date into descending sequence of the actual amount of the transaction (largest first, smallest last). If two or more transactions of \$100.00 were recorded for account #12345 on the 31st of August 2009, those transactions will be retained in the order in which they were read from the **USING** file(s) or were **RELEASED** to the **SORT**.

Stage 3 (the output phase):

- a. Once the sort phase is complete, a copy of the sorted data will be written to each *file-name-2* if the **GIVING** clause was specified. When **GIVING** is specified, none of the *file-name-2* files can be **OPEN** at the time the **SORT** is executed.
- b. When an **OUTPUT PROCEDURE** is used, the procedure(s) specified on the **OUTPUT PROCEDURE** clause will be invoked as if by a procedural **PERFORM** statement with no **VARYING** or **UNTIL** options specified. Records will be retrieved from the sort work file – one at a time and in sorted sequence – within the **INPUT PROCEDURE** using the **RETURN** statement.

A **GO TO** statement that transfers control out of the **OUTPUT PROCEDURE** will terminate the **SORT** but allows the program to continue executing from the point where the **GO TO** transferred control to. Once an **OUTPUT PROCEDURE** has been aborted using a **GO TO** it cannot be resumed. You may, however, re-execute the **SORT** statement itself. **USING A "GO TO" TO PREMATURELY TERMINATE A SORT, OR RE-STARTING A PREVIOUSLY-CANCELLED SORT IS NOT CONSIDERED GOOD PROGRAMMING STYLE AND SHOULD BE AVOIDED.**

- c. Once the **OUTPUT PROCEDURE** terminates, the output phase – and the **SORT** statement itself - is complete. Any sorted records that have not yet been **RETURNED** from the sort work file will be lost.
 - d. The scope of the **OUTPUT PROCEDURE** must not allow a file-based **SORT**, **MERGE** or **RELEASE**.
7. Should disk work files be necessary due to the amount of data being sorted, they will be automatically allocated to disk in a folder defined by the **TMPDIR**, **TMP** or **TEMP** environment variables (checked for existence in that sequence). These disk files will be automatically purged upon **SORT** termination or program execution termination (normal or otherwise).

See Also...

Types of Files	1.3.3.5	The MERGE Statement	6.4.25
The OBJECT-COMPUTER Paragraph	4.1.2	The OPEN Statement	6.4.29
Describing the Structure of a File (FD/SD)	5.1	The RELEASE Statement	6.2.33
Defining Data Items	5.2	The RETURN Statement	6.2.35
Storage Format of Data (USAGE)	5.2.1.11	Execution-time Environment Variables	8.2.4
The CLOSE Statement	6.4.7		

6.2.40.2. SORT Format 2 – Table Sort

Figure 6-101 - Table SORT Syntax

```

SORT table-name-1
  [ ON { ASCENDING
        DESCENDING } KEY identifier-1 ... ] ...
  [ WITH DUPLICATES IN ORDER ]
  [ COLLATING SEQUENCE IS alphabet-name-1 ]

```

This format of the **SORT** statement sorts relatively small quantities of data – namely data contained in a **DATA DIVISION** table – according to one or more key fields.

1. The *table-name-1* data item must have an **OCCURS** clause in its definition.
2. The *identifier-1* ... field(s), if any, must be defined as data items subordinate to *table-name-1*.
3. The **WITH DUPLICATES IN ORDER** clause is supported for compatibility purposes, but is non-functional. See the discussion of this clause in the previous section for more information.
4. The data within *table-name-1* will be sorted in-place (i.e. no sort file is required) according to the **KEY** specification(s) made on the **SORT** statement.
5. Although the specification of **KEY** clause(s) is optional³⁶, currently, a table **SORT** with no **KEY** specification(s) made on the **SORT** statement is unsupported by GNU COBOL and will be rejected by the compiler.
6. The sort will take place by arranging the data records in the sequence defined by the **ASCENDING KEY** and/or **DESCENDING KEY** specification(s) on the **SORT** statement according to the **COLLATING SEQUENCE** specified on the **SORT** (if any) or – if none was defined – the **PROGRAM COLLATING SEQUENCE** specified or implied by the **OBJECT-COMPUTER** paragraph. Keys may be any supported data type and **USAGE** except for level-78 or level-88 data items.
7. The **SORT** will be performed in-place within *table-name-1* – no sort file is required.

³⁶ When lacking a **KEY** clause, according to the COBOL2002 standards, a table sort will use the table's **KEY** clause

6.2.41. START

Figure 6-102 - START Syntax

```

START file-name-1

    [
      KEY IS [
        IS EQUAL TO | IS ≡ | EQUALS
        IS GREATER THAN | IS ≥
        IS GREATER THAN OR EQUAL TO | IS ≥= | IS NOT LESS THAN
        IS LESS THAN | IS ≤
        IS LESS THAN OR EQUAL TO | IS ≤= | IS NOT GREATER THAN
      ] identifier-1
    ]

    [ invalid-key-clause ]

    [ END-START ]

```

The **START** statement defines the logical starting point within a file for subsequent sequential read operations.

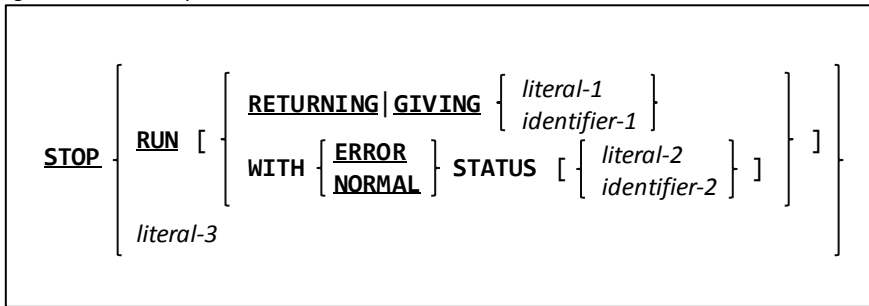
1. *file-name-1* must be an **ORGANIZATION RELATIVE** or **ORGANIZATION INDEXED** file.
2. *file-name-1* must have been **SELECT**ed with an **ACCESS MODE DYNAMIC** or **ACCESS MODE SEQUENTIAL**.
3. *file-name-1* must be **OPEN** in either **INPUT** or **I-O** mode at the time the **START** is executed.
4. If no **KEY** clause is specified, “**KEY IS EQUAL TO** *identifier-1*” will be assumed (see the next point for the definition of *identifier-1*).
5. If *file-name-1* is an **ORGANIZATION RELATIVE** file, *identifier-1* must be the defined **RELATIVE KEY** of the file. If *file-name-1* is an **ORGANIZATION INDEXED** file, *identifier-1* must be the defined **RECORD KEY** of the file (if no **KEY** clause was specified) or may be the **RECORD KEY** or any of the **ALTERNATE RECORD KEY** fields for the file is a **KEY** clause is specified.
6. After successful execution of a **START** statement, the internal record pointer into the *file-name-1* data will be positioned such that the next sequential **READ** statement executed against *file-name-1* will read either:
 - a. The first record that satisfies the **KEY** clause specification if the relation check specified is **EQUAL TO**, **GREATER THAN** or **GREATER THAN OR EQUAL TO** (or any of their syntactical equivalents), or ...
 - b. The last record that satisfies the **KEY** clause specification if the relation check specified is **LESS THAN** or **LESS THAN OR EQUAL TO** (or any of their syntactical equivalents).
7. The **START** statement only positions the file for a subsequent sequential **READ** – it does not actually populate *file-name-1*s 01-level records with new data. You must issue a sequential **READ** after a successful **START** to actually read the record that satisfies the **KEY** clause.
8. The optional *invalid-key-clause* may be used to detect and recover from errors encountered during execution of the **START**. Such errors might be actual I/O errors or “Key Not Exists” errors (**FILE STATUS 23**), indicating no record exists that satisfies the **KEY** clause requirements. Lacking such a clause, you’ll need to test the file’s **FILE STATUS** data item manually after the **START** in order to determine success or failure.

See Also...

Types of Files	1.3.3.5	Relation Tests	6.1.8.2.5
Defining File Characteristics (SELECT)	4.2.1	The OPEN Statement	6.4.29
FILE-STATUS Values	Figure 4-15	The READ Statement	6.4.31

6.2.42. STOP

Figure 6-103 - STOP Syntax



The **STOP** statement halts the program, returning control to the operating system.

1. The **RUN** clause halts the program without displaying any special message to that effect.
2. The *literal-2* clause displays the specified text on **SYSDOUT/STDOUT**, waits for the user to press the **Enter** key and then – once the key has been pressed – allows the program to continue execution.
3. The optional **RETURNING/GIVING** clause (the **RETURNING** and **GIVING** clauses may be used interchangeably) provides the opportunity to return a numeric value to the operating system (a “return code”). The manner in which the return code may be interrogated by the operating system varies, but Windows can use %ERRORLEVEL% to query the return code while Unix shells such as sh, bash and ksh can query the return code as “\$?”. Other Unix shells may have different ways to access return code values.
4. The **STATUS** clause provides another means of returning a return code. Using the **STATUS** clause with a literal/identifier specification is functionally equivalent to using the **RETURNING/GIVING** clause.

Using the **STATUS** clause without a literal/identifier specification will return a return code of 0 if the **NORMAL** keyword is used or a 1 if **ERROR** was specified.
5. Your program will **ALWAYS** return a return code, even if no **RETURNING/GIVING** or **STATUS** clause is specified. In the absence of the use of these clauses, the value in the special register **RETURN-CODE** at the time the **STOP** statement is executed will be used as the return code.
6. Any programmer-defined exit procedure (established via the **CBL_EXIT_PROC** built-in subroutine) will be executed by **STOP RUN**, but not by **STOP literal**.
7. Valid return code values can be in the range -2147483648 to +2147483647.
8. The code snippets below are all equivalent – they show different ways in which a GNU COBOL program may be coded to pass a return code value of 16 back to the operating system and then halt.

STOP RUN RETURNING 16	MOVE 16 TO RETURN-CODE STOP RUN
STOP RUN WITH ERROR STATUS 16	

See Also...

Built-in Device Names	Figure 4-8
Special Registers	6.1.13

The CBL_EXIT_PROC Subroutine	8.3.1.25
-------------------------------------	--------------------------

6.2.43. STRING

Figure 6-104 - STRING Syntax

```

STRING
  { { literal-1 } [ DELIMITED BY { SIZE { literal-2 } } ] } ...
    INTO identifier-3
    [ WITH POINTER identifier-4 ]
    [ overflow-clause ]
  [ END-STRING ]

```

The **STRING** statement is used to concatenate all or a part of one or strings together, forming a new string.

1. *Literal-1*, *literal-2*, *identifier-1*, *identifier-2* and *identifier-3* must be explicitly or implicitly defined as alphanumeric **USAGE DISPLAY** data. Any of those identifiers may be group items.
2. *Identifier-4* must be a non-edited elementary integer numeric data item with a value greater than zero.
3. Each *literal-1* / *identifier-1* will be known as a sending item.
4. During the processing of the **STRING** statement, data will be copied from each sending item, in turn, into *identifier-3*, one character at a time at a position defined by the *current character pointer*.
5. The initial value of the *current character pointer* will be the value of *identifier-4* at the time the **STRING** statement began execution. If no **WITH POINTER** clause is coded, a value of 1 (meaning "the 1st character position") will be assumed for the *current character pointer*.
6. For each sending item, the contents of the sending item will be copied – character-by-character – into *identifier-3* at the character position specified by the *current character pointer*. After a character is copied, the *current character pointer* will be incremented by 1 so that it points to the position within *identifier-3* where the next character should be copied.
7. The **DELIMITED BY** clause specifies how much of each sending item will be copied into the *identifier-3*. **DELIMITED BY SIZE** (the default if no **DELIMITED BY** clause is specified) causes the entire contents of the sending item to be copied into *identifier-3*. Using **DELIMITED BY *literal-2*** or **DELIMITED BY *identifier-2*** causes only the contents of the sending item up to but not including the character sequence specified by the literal or identifier to be copied.
8. **STRING** processing will cease when one of the following occurs:
 - a. All sending items have been fully processed, or ...
 - b. The initial value of the *current character pointer* is less than 1, or ...
 - c. The value of the *current character pointer* exceeds the size of *identifier-3* at the point the **STRING** statement wants to copy a character into *identifier-3*

Events b and c reflect an overflow condition, which may be handled by use of the optional *overflow-clause*. Note that in the case event b occurs, no data will be copied into *identifier-3*.
9. *Identifier-3* is neither automatically initialized (to **SPACES** or any other value) at the start of a **STRING** statement nor will it be **SPACE** filled should the total number of sending item characters copied into it be less than its size. You may explicitly initialize *identifier-3* yourself via the **INITIALIZE** or **MOVE** statements before executing the **STRING** if you wish.

See Also...

Storage Format of Data (USAGE)	5.2.1.11	The INITIALIZE Statement	6.2.22
Handling Overflow (ON OVERFLOW)	6.1.12.5	The MOVE Statement	6.2.26

6.2.44. SUBTRACT

6.2.44.1. SUBTRACT Format 1 – SUBTRACT FROM

Figure 6-105 - SUBTRACT FROM Syntax

```

SUBTRACT { literal-1
             identifier-1 }
           FROM { identifier-2 [ rounding-option ] } ...
           [ size-error-clause ]
           [ END-SUBTRACT ]

```

This format of the **ADD** statement generates the arithmetic sum of all arguments that appear before the **FROM** (*identifier-1* or *literal-1*) and subtracts that sum from each *identifier-2*.

1. *Identifier-1* and *identifier-2* must be numeric unedited data items.
2. *Literal-1* must be a numeric literal.
3. The optional “*rounding-option*” clause available to each *identifier-2* will control how non-integer results will be saved.
4. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-2* is insufficiently sized to hold the generated results.

See Also...

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

Rounding Options [6.1.12.7](#)

6.2.44.2. SUBTRACT Format 2 – SUBTRACT GIVING

Figure 6-106 - SUBTRACT GIVING Syntax

```

SUBTRACT { literal-1
             identifier-1 }
           FROM identifier-2
           GIVING { identifier-3 [ rounding-option ] } ...
           [ size-error-clause ]
           [ END-SUBTRACT ]

```

This format of the **SUBTRACT** statement generates the arithmetic sum of all arguments that appear before the **FROM** (*identifier-1* or *literal-1*), subtracts that sum from the contents of *identifier-2* and then replaces the contents of the identifiers listed after the **GIVING** (*identifier-3*) with that result.

1. *Identifier-1* and *identifier-2* must be numeric unedited data items.
2. *Identifier-3* must be a numeric (edited or unedited) data item.
3. *Literal-1* must be a numeric literal.
4. The optional “*rounding-option*” clause available to each *identifier-2* will control how non-integer results will be saved.
5. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-2* is insufficiently sized to hold the generated results.

See Also...

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

Rounding Options [6.1.12.7](#)

6.2.44.3. SUBTRACT Format 3 – SUBTRACT CORRESPONDING

Figure 6-107 - SUBTRACT CORRESPONDING Syntax

```

SUBTRACT CORRESPONDING identifier-1
      FROM identifier-2 [ rounding-option ]
      [ size-error-clause ]
[ END-SUBTRACT ]

```

This format of the **SUBTRACT** statement generates code equivalent to individual **SUBTRACT FROM** statements for corresponding matches of data items found subordinate to the two identifiers.

4. When corresponding matches are established, the effect of a **SUBTRACT CORRESPONDING** on those matches will be as if a series of individual **SUBTRACT FROM** statements were done – one for each match.
5. The optional “*rounding-option*” clause available to each *identifier-2* will control how non-integer results will be saved.
6. The optional *size-error-clause* may be used to detect arithmetic overflow situations where *identifier-2* is insufficiently sized to hold the generated results.

See Also...

The **CORRESPONDING** Clause [6.1.12.2](#)

Rounding Options [6.1.12.7](#)

Handling Size Errors (**ON SIZE ERROR**) [6.1.12.6](#)

6.2.45. SUPPRESS

Figure 6-108 - SUPPRESS Syntax



SUPPRESS PRINTING

Although syntactically recognized by the GNU COBOL compiler, the **SUPPRESS** statement is non-functional because the RWCS (COBOL Report Writer) is not currently supported by GNU COBOL.

6.2.46. TERMINATE

Figure 6-109 - TERMINATE Syntax

```
TERMINATE identifier-1 ...
```

Although syntactically recognized by the GNU COBOL compiler, the **TERMINATE** statement is non-functional because the RWCS (COBOL Report Writer) is not currently supported by GNU COBOL.

6.2.47. TRANSFORM

Figure 6-110 - TRANSFORM Syntax

TRANSFORM <i>identifier-1</i> FROM { <i>literal-1</i> <i>identifier-2</i> } TO { <i>literal-2</i> <i>identifier-3</i> }

The **TRANSFORM** statement scans a data item performing a series of monoalphabetic substitutions, defined by the arguments before and after the “**TO**” clause.

1. Both *literal-1* and/or *literal-2* must be alphanumeric literals.
2. All of *identifier-1*, *identifier-2* and *identifier-3* must either be group items or alphanumeric data items. Data items that are **PICTURE 9 USAGE DISPLAY** are accepted, but will generate warning messages from the compiler.
3. The **TRANSFORM** statement will replace characters within *identifier-1* that are found in the string specified before the **TO** keyword with the corresponding characters from the string specified after the **TO** keyword.
4. This statement exists within GNU COBOL only to provide compatibility with COBOL programs written to pre-1985 standards. The **TRANSFORM** verb was made obsolete in the 1985 standard of COBOL, having been replaced by the **CONVERTING** clause of the **INSPECT** statement. New programs should be coded to use **INSPECT CONVERTING** rather than **TRANSFORM**.

See Also...

Defining a Data Item's PICTURE	5.2.1.6
---------------------------------------	-------------------------

The INSPECT Statement	6.2.24.3
------------------------------	--------------------------

Storage Format of Data (USAGE)	5.2.1.11
-----------------------------------------	--------------------------

6.2.48. UNLOCK

Figure 6-111 - UNLOCK Syntax

```
UNLOCK file-name-1 { RECORD  
                          RECORDS }
```

This statement syncs any as-yet unwritten file I/O buffers to the specified file (if any) and releases any record locks held for records belonging to the named file.

1. If *file-name-1* is a Sort/Merge work file, no action will be taken.
2. Not all GNU COBOL implementations support locking. Whether they do or not depends upon the operating system they were built for and the build options that were used when GNU COBOL was generated.³⁷ When a program using one of those GNU COBOL implementations issues an **UNLOCK**, it will be ignored. There will be no compiler message issued. Buffer syncing, if needed, will still occur.

See Also...

Record Locking [6.1.9.1](#)

³⁷ The author of this manual – for example – uses a GNU COBOL build for Windows that utilizes the MinGW build/runtime environment and uses the Berkeley Database module for advanced file I/O. That GNU COBOL build does NOT support LOCKing. Generally speaking, UNIX builds will support record locking.

6.2.49. UNSTRING

Figure 6-112 - UNSTRING Syntax

```

UNSTRING identifier-1
      DELIMITED BY { [ ALL ] literal-1 } [ OR { [ ALL ] literal-2 } ] ...
      INTO { identifier-4 [ DELIMITER IN identifier-5 ] [ COUNT IN identifier-6 ] } ...
      [ WITH POINTER identifier-7 ]
      [ TALLYING IN identifier-8 ]
      [ overflow-clause ]
      [ END-UNSTRING ]

```

The **UNSTRING** statement parses a string, extracting any number of substrings from it.

- Identifier-1* through *identifier-5* must be explicitly or implicitly defined as alphanumeric **USAGE DISPLAY** data. Any of those identifiers may be group items.
- Literal-1* and *literal-2* must be alphanumeric literals.
- Identifier-7* and *identifier-8* must be elementary non-edited integer numeric items.
- Identifier-7* must have a value greater than 0.
- Identifier-1* is known as the *source string*. *Identifier-4* is known as the *destination field*.
- The source string will be broken up into substrings starting from the character position indicated by *identifier-7* (or from position 1 if there is no **WITH POINTER** clause). If the initial value of *identifier-7* is less than 1 or greater than the size of the source string, an "overflow" condition results. An overflow condition can be detected and dealt with using the optional *overflow-clause*.
- Substrings are identified by using the various delimiter strings specified on the **DELIMITED BY** clause as inter-substring separators. Using the "**ALL**" option allows a delimiter sequence to be an arbitrarily long sequence of occurrences of the delimiter literal whereas its absence treats each occurrence as a separate delimiter. When multiple delimiters are specified, they will be looked for in the source string in the sequence in which they are coded.
- Two consecutive delimiter sequences will identify a null substring.
- Each destination field may have an optional **DELIMITER** clause. If a **DELIMITER** clause is specified, *identifier-5* will have the delimiter character string used to identify the substring for the destination field **MOVED** to it **if and only if** data was actually found for that destination field (if not, *identifier-5* remains unchanged).
- Each destination field may have an optional **COUNT** clause. If a **COUNT** clause is specified, *identifier-6* will have the size of the substring for the destination field **MOVED** to it **if and only if** data was actually found for that destination field (if not, *identifier-6* remains unchanged).
- The **TALLYING** clause – if present – will be incremented by 1 each time a parsed substring is **MOVED** to a destination field.
- None of *identifier-4*, *identifier-5*, *identifier-6*, *identifier-7* or *identifier-8* are initialized by the **UNSTRING** statement. You need to do that yourself via **MOVE** or **INITIALIZE**.

The following sample program illustrates the **UNSTRING** statement.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DEMOUNSTRING.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Full-Name          PIC X(40).
   05 Last-Name       PIC X(15).
   05 First-Name      PIC X(15).
   05 MI              PIC X(1).
   05 Delim-LN        PIC X(1).
   05 Delim-FN        PIC X(1).
   05 Delim-MI        PIC X(1).
   05 Count-LN        BINARY-CHAR.
   05 Count-FN        BINARY-CHAR.
   05 Count-MI        BINARY-CHAR.
   05 Tallying-Ctr    BINARY-CHAR.
PROCEDURE DIVISION.
P1. PERFORM UNTIL EXIT
    DISPLAY "Enter Full Name (null quits):"
        WITH NO ADVANCING
    ACCEPT Full-Name
    IF Full-Name = SPACES
        EXIT PERFORM
    END-IF
    INITIALIZE Parsed-Info
    UNSTRING Full-Name DELIMITED BY ", "
        OR ", "
        OR ALL
    SPACES
        INTO Last-Name DELIMITER IN Delim-LN
            COUNT IN Count-LN
            First-Name DELIMITER IN Delim-FN
            COUNT IN Count-FN
            MI DELIMITER IN Delim-MI
            COUNT IN Count-MI
        TALLYING Tallying-Ctr
    DISPLAY "First-Name=" First-Name
        " Delim=" Delim-FN
        " Count=" Count-FN
    DISPLAY "MI =" MI "
    "
        " Delim=" Delim-MI
        " Count=" Count-MI
    DISPLAY "Last-Name =" Last-Name
        " Delim=" Delim-LN
        " Count=" Count-LN
    DISPLAY "Tally=" Tallying-Ctr
END-PERFORM
DISPLAY "Bye!"
STOP RUN
.
```

The following is sample output from the program:

```
Enter Full Name (null quits):Cutler, Gary L
First-Name=Gary          Delim=' ' Count=+004
MI =L                    Delim=' ' Count=+001
Last-Name =Cutler        Delim=', ' Count=+006
Tally= +003
Enter Full Name (null quits):Snoddgrass,Throckmorton,P
First-Name=Throckmorton Delim=', ' Count=+012
MI =P                    Delim=' ' Count=+001
Last-Name =Snoddgrass   Delim=', ' Count=+010
Tally= +003
Enter Full Name (null quits):Munster Herman
First-Name=Herman       Delim=' ' Count=+006
MI =                    Delim=' ' Count=+000
Last-Name =Munster      Delim=', ' Count=+007
Tally= +002
Enter Full Name (null quits):
Bye!
```

See Also...

Storage Format of Data (**USAGE**) [5.2.1.11](#)

Handling Overflow (**ON OVERFLOW**) [6.1.12.5](#)

The **INITIALIZE** Statement [6.2.22](#)

The **MOVE** Statement [6.2.26](#)

6.2.50. WRITE

Figure 6-113 - WRITE Syntax

```

WRITE record-name-1
  [ FROM { literal-1
            | identifier-1 } ]
  [ { WITH LOCK
      | WITH NO LOCK } ]
  [ { AFTER
      | BEFORE } ADVANCING { { literal-2
                                   | identifier-2 } LINE | LINES
                                   | mnemonic-name-1
                                   | PAGE } ]
  [ AT END-OF-PAGE | EOP imperative-statement-1 ]
  [ NOT AT END-OF-PAGE | EOP imperative-statement-2 ]
  [ invalid-key-clause ]
  [ END-WRITE ]

```

The **WRITE** statement writes a new record to an **OPEN** file.

1. *Record-name-1* must be defined as an 01-level record subordinate to the File Description (**FD**) of a file that is currently OPEN for **OUTPUT**, **I-O** or **EXTEND**.
2. *Literal-1* or *identifier-1* must be explicitly or implicitly defined as alphanumeric **USAGE DISPLAY** data. *Identifier-1* may be a group item.
3. The optional **FROM** clause will cause *literal-1* or *identifier-1* to be implicitly **MOVED** into *record-name-1* prior to writing *record-name-1* to the file.
4. The optional **LOCK** clauses allow you to lock the newly-written record (**LOCK**) or to explicitly state that it should not be locked (**NO LOCK**). The default is **WITH NO LOCK**.
5. The optional *invalid-key-clause* is legal only on **WRITE** statements used against for **ORGANIZATION RELATIVE** or **ORGANIZATION INDEXED** files; it may be used to detect and recover from situations where a non-zero **FILE STATUS** results from the **WRITE** (as might be the case if you try to **WRITE** a relative file record that already exists (use **REWRITE** instead) or attempt to duplicate a **RECORD KEY** value when **WRITE**ing to an **INDEXED** file.

The following points apply exclusively to files **SELECT**ed and **ASSIGN**ed to a **LINE ADVANCING** file, or to files with an **ORGANIZATION** of **LINE SEQUENTIAL**

6. The **ADVANCING** and **END-OF-PAGE** clauses are intended for use only with these types of files. Using this clause with any other **ORGANIZATION** will either be rejected outright by the compiler (**ORGANIZATION IS RELATIVE** or **ORGANIZATION IS INDEXED**) or may introduce unwanted characters into the file (**ORGANIZATION IS RECORD BINARY SEQUENTIAL**).
7. Both of these file types will use an end-of-record delimiter character sequence to signify where one record ends and the next record begins. This delimiter sequence may be any of the following:
 - a. A line-terminator sequence consisting of an ASCII carriage-return/line-feed character sequence (X'0D0A') if you are running a MinGW or native Windows build of GNU COBOL
 - b. A line-terminator sequence consisting of an ASCII line-feed character (X'0A') if you are running a Cygwin, Linux, Unix or OSX build of GNU COBOL
 - c. An ASCII formfeed character
8. If no **ADVANCING** clause is specified on a **WRITE** to an **ORGANIZATION LINE SEQUENTIAL** file, **BEFORE ADVANCING 1 LINE** will be assumed.
9. If no **ADVANCING** clause is specified on a **WRITE** to a **LINE ADVANCING** file, **AFTER ADVANCING 1 LINE** will be assumed.

The following points apply exclusively to files **SELECT**ed and **ASSIGN**ed to a **LINE ADVANCING** file, or to files with an **ORGANIZATION** of **LINE SEQUENTIAL**

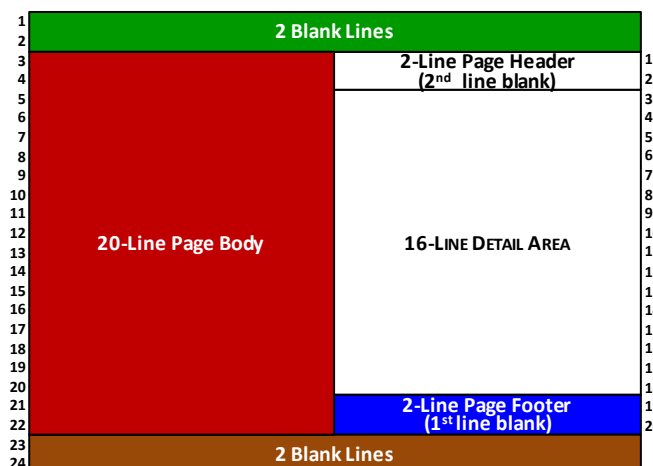
10. When **BEFORE ADVANCING** is used (or implied), the record is written to the file before the **ADVANCING** action writes line-terminator characters to the file.
11. If **AFTER ADVANCING** is used (or implied), the **ADVANCING** action takes place and then the record data is written to the file.
12. The **ADVANCING n LINES** clause will introduce the specified number of line-terminator character sequences into the file either before the written record (**AFTER ADVANCING**) or after the written record (**BEFORE ADVANCING**).
13. If the **LINAGE** clause is absent from the file's **FD**:
 - a. The **ADVANCING PAGE** clause will introduce an ASCII formfeed character into the file either before the written record (**AFTER PAGE**) or after the written record (**BEFORE ADVANCING**).
 - b. Management of areas on the printed page such as top-of page headers, bottom-of-page footers, dealing with "full page" situations and the like are the complete responsibility of the programmer
14. If the **LINAGE** clause is present in the file's **FD**:
 - a. The **ADVANCING PAGE** clause will introduce the appropriate number of line-terminator character sequences into the file either before the written record (**AFTER ADVANCING**) or after the written record (**BEFORE ADVANCING**) so as to force the printer to automatically advance to a new sheet of paper when the file prints. When **LINAGE** is specified, no formfeed characters will be generated. Instead, it is assumed that the printer to which the report will be printed will be loaded with special forms with specific characteristics as to page body size (the total number of printable lines on the paper) and skipped top-and/or bottom-of-page margins within which printing physically could occur, but in the case of these forms shouldn't.
 - b. Management of areas on the printed page such as top-of page headers, bottom-of-page footers, dealing with "full page" situations and the like are now the joint responsibility of the programmer and the GNU COBOL run-time library, which provides tools such as the **LINAGE-COUNTER** special; register and the **AT END-OF-PAGE** clause on the **WRITE** statement to deal with page formatting issues.
 - c. The **AT END-OF-PAGE** and **NOT AT END-OF-PAGE** clauses are legal only for **ORGANIZATION LINE SEQUENTIAL** or **ORGANIZATION RECORD BINARY SEQUENTIAL** files whose file descriptions contain a **LINAGE** clause. The **AT END-OF-PAGE** clause will be triggered (thus executing *imperative-statement-1*) if the **WRITE** statement introduces a data line or line-feed character into the file at a line position within the Page Footer area (see [Figure 5-3](#)). The **NOT AT END-OF-PAGE** clause will be triggered (thus executing *imperative-statement-2*) if no end-of-page condition occurred during the **WRITE**.

A report is to be written to a special form that consists of 24 total possible printed lines; the layout of the report is shown to the right.

The GNU COBOL **LINAGE** clause that describes this layout is as follows. Colors in the code below relate to the colored areas on the page layout.

```
LINAGE IS 20 LINES
FOOTING 19
LINES AT TOP 2
LINES AT BOTTOM 2
```

The total vertical size of the form (as measured in printable lines) is the sum of the **LINES AT TOP**, **LINAGE** and **LINES AT BOTTOM** clause values. The **FOOTING** clause indicates at what relative line number within the page body (the value specified on the **LINAGE** clause) the detail area is to end and the footer area is to begin. It is at the point where printing reaches this **FOOTING** point that an **END-OF-PAGE** condition exists.



The following program generates a test report (of 25 detail lines) using the page layout just described.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DEMOLINAGE.
ENVIRONMENT DIVISION.
```

And here are the pages of the generated report:

1. []


```

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT Data-File          ASSIGN TO
                                "linage-ls.txt"
                                LINE SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD Data-File
    LINAGE IS 20 LINES
    FOOTING 19
    LINES AT TOP 2
    LINES AT BOTTOM 2.
01 Data-Rec.
    05 FILLER                    PIC X(7).
    05 DR-Write-No                PIC 9(2).
    05 FILLER                    PIC X(28).
    05 DR-LINAGE-COUNTER         PIC 9(3).
WORKING-STORAGE SECTION.
01 Flags.
    05 Report-Complete-Flag     PIC X(1).
    88 Report-Complete VALUE 'Y' FALSE IS 'N'.
01 I                             PIC 9(2).
PROCEDURE DIVISION.
000-Main.
*>-----
*> Open the report file and print the initial page
*> header
*>-----
    OPEN OUTPUT Data-File
    SET Report-Complete TO FALSE
    PERFORM 100-Page-Header
*>-----
*> Print 25 report detail lines
*>-----
    PERFORM VARYING I FROM 1 BY 1 UNTIL I > 25
        MOVE "Detail NN          LINAGE-COUNTER="
            TO Data-Rec
        MOVE I TO DR-Write-No
        MOVE LINAGE-COUNTER OF Data-File
            TO DR-LINAGE-COUNTER
        WRITE Data-Rec
            AT EOP
            IF LINAGE-COUNTER >= 19
                PERFORM 200-Page-Footer
                PERFORM 100-Page-Header
            ELSE
                PERFORM 100-Page-Header
        END-IF
    END-WRITE
    END-PERFORM
*>-----
*> Print enough blank detail lines to produce the
*> final page footer
*>-----
    SET Report-Complete TO TRUE
    PERFORM UNTIL LINAGE-COUNTER OF DATA-FILE >= 19
        MOVE '                LINAGE-COUNTER='
            TO Data-Rec
        MOVE LINAGE-COUNTER OF Data-File
            TO DR-LINAGE-COUNTER
        WRITE Data-Rec
            AT EOP
            PERFORM 200-Page-Footer
        EXIT PERFORM
    END-WRITE
    END-PERFORM
*>-----
** All done!
*>-----
    CLOSE Data-File
    STOP RUN
.
100-Page-Header.
    MOVE "Page Header          LINAGE-COUNTER="
        TO Data-Rec
    MOVE LINAGE-COUNTER OF Data-File TO
        DR-LINAGE-COUNTER
    WRITE Data-Rec BEFORE ADVANCING 2 LINES
.

```

2.		
3.	Page Header	LINAGE-COUNTER= 001
4.		
5.	Detail 01	LINAGE-COUNTER= 003
6.	Detail 02	LINAGE-COUNTER= 004
7.	Detail 03	LINAGE-COUNTER= 005
8.	Detail 04	LINAGE-COUNTER= 006
9.	Detail 05	LINAGE-COUNTER= 007
10.	Detail 06	LINAGE-COUNTER= 008
11.	Detail 07	LINAGE-COUNTER= 009
12.	Detail 08	LINAGE-COUNTER= 010
13.	Detail 09	LINAGE-COUNTER= 011
14.	Detail 10	LINAGE-COUNTER= 012
15.	Detail 11	LINAGE-COUNTER= 013
16.	Detail 12	LINAGE-COUNTER= 014
17.	Detail 13	LINAGE-COUNTER= 015
18.	Detail 14	LINAGE-COUNTER= 016
19.	Detail 15	LINAGE-COUNTER= 017
20.	Detail 16	LINAGE-COUNTER= 018
21.		
22.	Page Footer	LINAGE-COUNTER= 020
23.		
24.		

1.		
2.		
3.	Page Header	LINAGE-COUNTER= 001
4.		
5.	Detail 17	LINAGE-COUNTER= 003
6.	Detail 18	LINAGE-COUNTER= 004
7.	Detail 19	LINAGE-COUNTER= 005
8.	Detail 20	LINAGE-COUNTER= 006
9.	Detail 21	LINAGE-COUNTER= 007
10.	Detail 22	LINAGE-COUNTER= 008
11.	Detail 23	LINAGE-COUNTER= 009
12.	Detail 24	LINAGE-COUNTER= 010
13.	Detail 25	LINAGE-COUNTER= 011
14.		LINAGE-COUNTER= 012
15.		LINAGE-COUNTER= 013
16.		LINAGE-COUNTER= 014
17.		LINAGE-COUNTER= 015
18.		LINAGE-COUNTER= 016
19.		LINAGE-COUNTER= 017
20.		LINAGE-COUNTER= 018
21.		
22.	Page Footer	LINAGE-COUNTER= 020
23.		
24.		

```

200-Page-Footer.
  WRITE Data-Rec FROM SPACES
    BEFORE ADVANCING 1 LINES
  MOVE "Page Footer"          LINAGE-COUNTER="
    TO Data-Rec
  MOVE LINAGE-COUNTER OF Data-File
    TO DR-LINAGE-COUNTER
  IF Report-Complete
*>-----
*> "BEFORE 0 LINES" Won't push into the next page
*>-----
  WRITE Data-Rec BEFORE ADVANCING 0 LINES
  ELSE
  WRITE Data-Rec BEFORE ADVANCING PAGE
  END-IF
.

```

See Also...

Types of Files	1.3.3.5
Defining File Characteristics (SELECT)	4.2.1
FILE-STATUS Values	Figure 4-15
Describing the Structure of a File (FD/SD)	5.1
Describing Record Layouts	5.1.1

Storage Format of Data (USAGE)	5.2.1.11
Handling Invalid Keys (INVALID KEY)	6.1.12.3
The CLOSE Statement	6.4.7
The MOVE Statement	6.2.26
The OPEN Statement	6.4.29
The REWRITE Statement	6.4.36

7. Sub-Programming with GNU COBOL

7.1. Subprograms, Subroutines and User-Defined Functions

Simply stated, a **SUBPROGRAM** is a program that is invoked by another program; the subprogram performs whatever its designed operations are and – when complete – typically returns control back to the program that invoked it. There are two different types of subprograms supported by GNU COBOL – subroutines and user-defined functions. The distinction between these two subprogram types lies in the manner in which they are executed.

When program “A” invokes subprogram “B” as a **SUBROUTINE**, it does so using a special statement dedicated to that function – the **CALL** statement – just as if “B” were one of the built-in system subroutines. When program “A” invokes program “B” as a **USER-DEFINED FUNCTION**, it does so in a manner identical to how “B” would have been invoked had it been one of the many built-in intrinsic functions. In either instance, program “A” is referred to as the **CALLING PROGRAM** while program- “B” is known as the **CALLED PROGRAM**. GNU COBOL programs may be a calling program, a called program or both. A program written in the C programming language may serve as either the calling or called program too. A called program may act as a calling program to a called program. When a calling program does not serve as a called program to any program, that calling program is known as a **MAIN PROGRAM**.

Both subroutines and user-defined functions may return a value. The value they return will be a **USAGE BINARY LONG SIGNED** integer in the range -2147483648 to +2147483647. This value will be available in the register **RETURN-CODE** and also as the value of the data item specified on the **RETURNING/GIVING** clause of a subroutine’s **CALL**.

See Also...

Storage Format of Data (USAGE)	5.2.1.11	The CALL Statement	6.4.5
Intrinsic Functions	6.1.7	Built-in System Subroutines	8.3
Special Registers	6.1.13		

7.2. Specifying and Using Alternate Entry Points

Any subroutine (but not a user-defined function) may have multiple entry-points defined within it. This means the subroutine could be called either via a “**CALL** ‘*effective-program-name*’” or a “**CALL** ‘*entry-point*’” statement. There may be any number of alternate entry-points defined within a subroutine.

The intent of alternate entry-points is to provide multiple ways in which the same subroutine could be **CALLED**, under the assumption that each entry-point will provide some different functionality to the calling program. For example, if you wished to write a subroutine that manipulates “student” records in a database, you might have the primary entry-point name (section 3) be for the coding that retrieves a student record from the database, while the alternate entry points “**ADD-STUDENT**”, “**UPDATE-STUDENT**” and “**DELETE-STUDENT**” provide the alternate functions implied by their entry-point names. The alternative to using multiple entry points in your subroutine, by the way, would be to include an additional argument to the primary (and only) entry point of the subroutine; this new argument might be named “**STUDENT-FUNCTION**” and might have values of “**FETCH**”, “**ADD**”, “**UPDATE**” or “**DELETE**”.

The primary entry-point for any subroutine is always the first executable (and non-**DECLARATIVES**) statement in the **PROCEDURE DIVISION**. The name of that entry-point (the name that will be **CALLED**) is the subroutine’s **PROGRAM-ID**.

Alternate entry points are added to a subroutine simply by adding **ENTRY** statements to the subroutine.

When an alternate entry-point is **CALLED**, execution within the subroutine will begin at the first executable statement following the **ENTRY** statement.

See Also...

The IDENTIFICATION DIVISION	3	The CALL Statement	6.4.5
Using DECLARATIVES	6.1.4	The ENTRY Statement	6.2.14

7.3. Dynamic Versus Static Subprograms

Any subprogram may be either statically or dynamically loaded into memory.

STATICALLY-LOADED (or simply **STATIC**) subprograms are part of the same executable file as their calling program and are loaded into memory as part of and at the same time as the calling program. **DYNAMICALLY-LOADED** (or **DYNAMIC**) subprograms exist as an executable file separate from that containing the calling program; these dynamic subprograms are located and loaded into memory the first time they are executed. Dynamic subprograms may be unloaded from memory via the **CANCEL** statement, if desired.

There are no functional differences between static and dynamic subprograms other than how they are compiled and when they are loaded into memory.

Here are the rules about GNU COBOL dynamically-loadable subprogram modules:

1. There may be any number of GNU COBOL subprograms contained within a single dynamically-loadable module.
2. Dynamically-loadable modules will be named “xxxxxxx.dll” on a Windows system or “xxxxxxx.so” on a Unix system, where “xxxxxxx” exactly matches, including the usage of upper- and lower-case letters, the primary entry-point name (**PROGRAM-ID**) or an alternate entry point name defined via the **ENTRY** statement of one of the GNU COBOL programs included in that module.
3. The first time any of the GNU COBOL subprograms in the dynamically-loadable module are invoked, the entry-point referenced must be the one for which the “.dll” or “.so” file is named (see rule #2).
4. The dynamically-loadable module file will be sought in the same directory from which the main program was loaded. If it cannot be found there, each directory named in the PATH that is in-effect for the main program’s execution will be searched. If it still cannot be found, execution will be terminated with an error message (“**libcob: Cannot find module 'xxxxxxx'**”).
5. Once the dynamically-loadable module has been successfully loaded (see rule #3), any of the entry-points contained within it are now available for reference, even if the dynamically-loadable module is subsequently **CANCEL**ed.

See Also...

The IDENTIFICATION DIVISION	3
The CANCEL Statement	6.2.6
The ENTRY Statement	6.2.14

Compiling & Dynamic-Linking Programs	8.1.3.2
Compiling & Static-Linking Programs	8.1.3.3

7.4. Subprogram Execution Flow

When a subprogram is invoked, the flow of execution will differ slightly depending on whether the subprogram is a subroutine or a user-defined function.

7.4.1. Subroutine Execution Flow

1. The calling program issues a statement of the form **CALL “entry-point” USING ...** to transfer control to the subroutine.
2. The called program will be located. If it is a **STATIC** subroutine it will already be part of the executable program issuing the **CALL**. If it is a **DYNAMIC** subroutine, it will be located and loaded as needed.
3. Execution of the calling program is suspended and control will transfer to the called program, as follows:
 - a. If the **PROGRAM-ID** clause of the subprogram included the **INITIAL** clause (section 3), the program will be reinitialized back to its compile-time state.
 - b. **LOCAL-STORAGE**, if any, will be allocated and initialized.
 - c. Execution will begin at the first executable statement following the subprograms entry-point. The entry point will be either:
 - ▶ The top of the **PROCEDURE DIVISION**, following any **DECLARATIVES** that might be present, if the subprogram was invoked using its primary entry-point name.
 - ▶ The first executable statement following the **ENTRY** statement naming the entry-point specified on the **CALL** if the subprogram was invoked using an alternate entry point.

4. The flow of execution will then progress through the coding of the subprogram as it would with any other program.
5. If the subprogram issues a **STOP RUN** statement, program execution ceases and control returns to the operating system or whatever execution monitor invoked the main program.
6. If the subprogram wishes to return control back to the calling program, it will do so using either the **GOBACK** or **EXIT PROGRAM** statement. At this time:
 - a. If the subprograms **PROCEDURE DIVISION** header or **ENTRY** statements included a **RETURNING** clause, the value of the data item found on that clause is **MOVED** to the **RETURN-CODE** special register³⁸.
 - b. **LOCAL-STORAGE**, if any, is de-allocated.
 - c. If the calling program included a **RETURNING** clause on the **CALL** statement that invoked the subprogram, the value of the “**RETURNING**” data item in the subroutine (see #6.a above) is **MOVED** to that data item.
 - d. Execution will resume back in the calling program with the first executable statement following the **CALL** that invoked the subprogram.

See Also...

The IDENTIFICATION DIVISION	3
The DATA DIVISION	5
The PROCEDURE DIVISION	6
Special Registers	6.1.13
Using DECLARATIVES	6.1.4
The CALL Statement	6.4.5

The ENTRY Statement	6.2.14
The EXIT PROGRAM Statement	6.2.16
The GOBACK Statement	6.2.19
The STOP RUN Statement	6.4.42
Dynamic vs Static Subprograms	7.3

7.4.2. User-Defined Function Execution Flow

1. The calling program, while in the process of evaluating an expression, encounters a reference to a user-defined function. Note that, unlike the built-in intrinsic functions, user-defined functions need never have the “**FUNCTION**” keyword coded in their references; the reason for this is that any program referencing a user-defined function must include that function in its **REPOSITORY** paragraph – that is sufficient to allow the compiler to recognize the function name as a function when it encounters a reference to it.
2. The called program which is that user-defined function will be located. If it is a **STATIC** user-defined function it will already be part of the executable program. If it is a **DYNAMIC** user-defined function, it will be located and loaded. Note that user-defined functions can only have primary entry points – the **ENTRY** statement is not valid within a user-defined function.
3. Execution of the calling program is suspended and control will transfer to the called program, as follows:
 - a. **LOCAL-STORAGE**, if any, will be allocated and initialized.
 - b. Execution will begin at the top of the **PROCEDURE DIVISION**, following any **DECLARATIVES** that might be present.
4. The flow of execution will then progress through the coding of the subprogram as it would with any other program.
5. If the subprogram issues a **STOP RUN** statement, program execution ceases and control returns to the operating system or whatever execution monitor invoked the main program.
6. If the subprogram wishes to return control back to the calling program, it will do so using either the **GOBACK** or **EXIT FUNCTION** statement. At this time:
 - a. The value of the data item found on the user-defined functions **PROCEDURE DIVISION RETURNING** clause is **MOVED** to the **RETURN-CODE** special register.
 - b. **LOCAL-STORAGE**, if any, is de-allocated.

³⁸ This behavior can be altered utilizing the **CALL-CONVENTION** feature of the **SPECIAL-NAMES** paragraph to define a subroutine calling convention that leaves **RETURN-CODE** unchanged, and then using that calling convention on the **CALL** that invokes the subroutine.

- c. Execution will resume back in the calling program at the point in the expression evaluation process where the returned value of the function is needed. At that point, the value in the **RETURN-CODE** special register will be used for the functions value.

See Also...

The REPOSITORY Paragraph	4.1.3	The EXIT FUNCTION Statement	6.2.16
The DATA DIVISION	5	The GOBACK Statement	6.2.19
Special Registers	6.1.13	The STOP RUN Statement	6.4.42
Using DECLARATIVES	6.1.4	Dynamic vs Static Subprograms	7.3
The ENTRY Statement	6.2.14		

7.5. Sharing Data Between Calling and Called Programs

7.5.1. Subprogram Arguments

7.5.1.1. Calling Program Considerations

Data items defined in a calling program may be passed to either type of called program (subroutine or user-defined function) as **ARGUMENTS**.

Arguments must be described in both the calling and called programs, and should be described in an identical manner with regard to the following characteristics:

- ▶ **PICTURE** (including both type and length)
- ▶ **SIGN**
- ▶ **SYNCHRONIZED**
- ▶ **USAGE**

A subroutine may be passed a maximum of 36 arguments³⁹. There is no built-in GNU COBOL limit to how many arguments a user-defined function may be passed.

Whether or not changes made to an argument within a subroutine will be “visible” to the calling program depends on how the argument was passed. There are three ways in which arguments may be passed from a calling program to a subroutine, as defined by the use of optional “**BY**” clauses in the **CALL** statement’s list of arguments.

As an example, the following **CALL** statement passes three arguments to a subroutine – each argument is passed differently.

```
CALL "subroutine" USING BY REFERENCE arg-1
                        BY CONTENT  arg-2
                        BY VALUE   arg-3
END-CALL
```

The three ways arguments are passed are as follows.

BY REFERENCE When a subroutine argument is passed **BY REFERENCE**, the subroutine is passed the address of the actual data item being passed as an argument. The item may anything defined within the **DATA DIVISION** of the program. If the subroutine modifies the contents of this argument, the calling program will “see” the results of that change when the subroutine returns control. This is the default manner in which GNU COBOL passes arguments to a subroutine, should no “**BY**” clauses be included on the **CALL**.

BY CONTENT When a subroutine is passed an argument **BY CONTENT**, the subroutine is passed the address of a copy of the actual data being passed as an argument. The item may anything defined within the **DATA DIVISION** of the program. The copy is made each time the **CALL** statement is executed, immediately before the **CALL** is actually executed. If the subroutine modifies the contents of this argument, it will be the copy that is modified, not the original data item specified on the **CALL**; the calling program will therefore not “see” the results of that change when the subroutine returns control.

³⁹ If you build the GNU COBOL software yourself from the distributed source, you CAN change this value by altering the defined value of `COB_MAX_FIELD_PARAMS` in the “common.h” header file.

BY VALUE Passing a subroutine argument **BY VALUE** passes the actual value of the data being passed as an argument. The item may any elementary binary numeric item (see [Figure 7-1](#)) defined within the **DATA DIVISION** of the program. If the subroutine modifies the contents of this argument, the calling program will not “see” the results of that change when the subroutine returns control.

The first two ways in which arguments may be passed (**BY REFERENCE** and **BY CONTENT**) are intended for use when a GNU COBOL program, is being called, while the first and third (**BY REFERENCE** and **BY VALUE**) are intended for use when a C program is being called. You can use **BY VALUE** arguments when calling GNU COBOL subroutines, but remember that those arguments are limited to being a numeric binary data type.

Each “**BY**” clause on a **CALL** statement may list multiple arguments.

Arguments to user-defined functions are automatically passed **BY REFERENCE**.

See Also...

Defining Data Items 5.2	Subprograms: Subroutines vs Functions 7.1
The CALL Statement 6.4.5	

7.5.1.2. Called Program Considerations

When coding a GNU COBOL subprogram (a subroutine or user-defined function), all arguments to the subprogram must be defined in the subprogram’s **LINKAGE SECTION**. These arguments must be explicitly included on the **PROCEDURE DIVISION** header via a “**USING**” clause that lists the arguments in the sequence in which they will be passed to the subprogram.

These arguments listed in a **USING** clause included on the **PROCEDURE DIVISION** header may each be defined as either “**BY REFERENCE**”, if they are being passed to the subprogram as “**BY REFERENCE**” or “**BY CONTENT**” arguments (on the **CALL**) or as “**BY VALUE**” if they are being passed “**BY VALUE**”. By default, all arguments are assumed to be “**BY REFERENCE**” unless explicitly stated otherwise. Arguments to a user-defined function are always to be specified as “**BY REFERENCE**” (either explicitly or by not using any “**BY**”).

If the subprogram returns a value, the data item in which the value is returned must also be defined in the subprogram’s **LINKAGE SECTION**, with an effective **PICTURE** and **USAGE** of **BINARY-LONG SIGNED**.

See Also...

Defining a Data Item’s PICTURE 5.2.1.6	The PROCEDURE DIVISION 6
Storage Format of Data (USAGE) 5.2.1.11	

7.5.2. GLOBAL Data Items

Another way in which a data item may be shared between a calling program (“A”) and a called program (“B”) is by defining the data item in the calling program and attaching the **GLOBAL** clause to it so that it may be used within the called program. In order for this to work, program “B” (the one called by program “A”) must be a *nested subprogram* within program “A”.

Here’s a small example:

Program Source Code	DISPLAYed Output When Executed
<pre> IDENTIFICATION DIVISION. PROGRAM-ID. DemoGLOBAL. ENVIRONMENT DIVISION. DATA DIVISION. WORKING-STORAGE SECTION. 01 Arg GLOBAL PIC X(10). PROCEDURE DIVISION. 000-Main. MOVE ALL "X" TO Arg CALL "DemoSub" END-CALL DISPLAY "DemoGLOBAL: " Arg END-DISPLAY GOBACK . IDENTIFICATION DIVISION. PROGRAM-ID. DemoSub. PROCEDURE DIVISION. 000-Main. </pre>	<pre> DemoGLOBAL: ***** </pre>


```

MOVE ALL "*" TO Arg.
GOBACK
.
END PROGRAM DemoSub.

END PROGRAM DemoGLOBAL.
    
```

See Also...

Details of Nested Subprograms [7.6](#)

7.5.3. EXTERNAL Data Items

The final way in which a data item may be shared between a calling program (“A”) and a called program (“B”) is by defining the data item (with the same name) in both programs and attaching the **EXTERNAL** clause to it (again, in both programs). This approach works regardless of whether the called program is nested within the calling program or not. It also works even if the two programs are compiled separately.

Here’s a small example:

Program Source Code	DISPLAYed Output When Executed
<pre> IDENTIFICATION DIVISION. PROGRAM-ID. DemoEXTERNAL. ENVIRONMENT DIVISION. DATA DIVISION. WORKING-STORAGE SECTION. 01 Arg EXTERNAL PIC X(10). PROCEDURE DIVISION. 000-Main. MOVE ALL "X" TO Arg CALL "DemoSub" END-CALL DISPLAY "DemoEXTERNAL: " Arg END-DISPLAY GOBACK . END PROGRAM DemoEXTERNAL. IDENTIFICATION DIVISION. PROGRAM-ID. DemoSub. DATA DIVISION. WORKING-STORAGE SECTION. 01 Arg EXTERNAL PIC X(10). PROCEDURE DIVISION. 000-Main. MOVE ALL "*" TO Arg. GOBACK . END PROGRAM DemoSub. </pre>	<pre> DemoEXTERNAL: ***** </pre>

7.6. Nested Subprograms

Normally, GNU COBOL source files contain the coding for a single program; that program may be a main program or a subprogram.

There’s no reason, however, why you cannot include multiple GNU COBOL programs into a single source file – one after the other – provided you structure the programs in the source file as follows:

```

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG1.
...
    
```

Program source code may be concatenated as shown here, provided an “**END PROGRAM**” statement naming the **PROGRAM-ID** of the just-completed program is used to separate one program from another.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG2.
...
    
```

There’s no reason that user-defined functions cannot be included too – they’ll just have **FUNCTION-IDs** and will be ended by “**END FUNCTION**” statements.

The last program in any GNU COBOL source file need not have an **END PROGRAM** (or **END FUNCTION**) statement.

When multiple programs occur in a source file, it is assumed that the programs are related to one another in that they will be **CALLed** or executed as functions from the others.

It is also possible to create source files where GNU COBOL programs are nested inside each other. Take for example these four GNU COBOL programs:

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG1.

...

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG2.

...

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG3.

...

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG4.

...

Here we see that PROG2 is nested inside of PROG1 because there is no **END PROGRAM** statement separating them. This means that data items or files defined within PROG1 can be used within PROG2 simply by attaching the "GLOBAL" attribute to them back in PROG1 when they are defined.

Similarly, since there is no **END PROGRAM** statement separating PROG3 from PROG2, it is possible for PROG3 to access GLOBAL files and data items defined within PROG2. Since PROG2 is nested within PROG1, any **GLOBAL** resources defined within PROG1 will be available to PROG3 as well.

The two **END PROGRAM** statements for PROG3 and PROG2 (note their sequence) mean that PROG4 is nested within PROG1 only. It will not have access to any **GLOBAL** resources defined within either PROG2 or PROG3.

See Also...

Program Structure 1.5.2	The CALL Statement 6.4.5
The IDENTIFICATION DIVISION 3	

7.7. Recursive GNU COBOL Subprograms

It is possible for a subroutine to **CALL** itself, either directly or indirectly from another subroutine that it **CALLS**. Any subroutine that indulges in this sort of behavior (called **RECURSION**) is called a **RECURSIVE SUBROUTINE**. A GNU COBOL subroutine can be recursively invoked *only if it is defined to the GNU COBOL compiler as being a recursive subroutine*. This is accomplished by adding the **RECURSIVE** attribute to the subroutines **PROGRAM-ID** clause.

All User-defined functions can be invoked recursively.

Here is an example of a main program (DEMOFACT) that **CALLS** both a subprogram (RECURSIVESUB) and a user-defined function (RECURSIVEFUNC) to compute the factorial value of a number.

DEMOFACT (Main Program)	
IDENTIFICATION DIVISION. PROGRAM-ID. DEMOFACT. ENVIRONMENT DIVISION. CONFIGURATION SECTION. REPOSITORY. FUNCTION RECURSIVEFUNC. DATA DIVISION. WORKING-STORAGE SECTION. 01 Result USAGE BINARY-LONG. 01 Arg USAGE BINARY-LONG VALUE 6. PROCEDURE DIVISION. 000-Main. CALL "RECURSIVESUB" USING BY CONTENT Arg RETURNING Result DISPLAY Arg "! = " Result DISPLAY Arg "! = " RECURSIVEFUNC(Arg) GOBACK . END PROGRAM DEMOFACT.	
RECURSIVESUB (a RECURSIVE subroutine)	RECURSIVEFUNC (a user-defined function)
IDENTIFICATION DIVISION. PROGRAM-ID. RECURSIVESUB RECURSIVE. DATA DIVISION. WORKING-STORAGE SECTION. 01 Result USAGE BINARY-LONG. 01 Next-Arg USAGE BINARY-LONG. 01 Next-Result USAGE BINARY-LONG. LINKAGE SECTION. 01 Arg USAGE BINARY-LONG.	IDENTIFICATION DIVISION. FUNCTION-ID. RECURSIVEFUNC. ENVIRONMENT DIVISION. CONFIGURATION SECTION. REPOSITORY. FUNCTION RECURSIVEFUNC. DATA DIVISION. WORKING-STORAGE SECTION. LINKAGE SECTION.

<pre> PROCEDURE DIVISION USING Arg RETURNING Result. 000-Main. DISPLAY "Entering RECURSIVESUB Arg=" Arg IF Arg = 1 MOVE 1 TO Result DISPLAY "Leaving RECURSIVESUB Returning " Result ELSE SUBTRACT 1 FROM Arg GIVING Next-Arg CALL "RECURSIVESUB" USING BY CONTENT Next-Arg RETURNING Next-Result COMPUTE Result = Arg * Next-Result DISPLAY "Leaving RECURSIVESUB Returning " Result "=" Arg "*" Next-Result END-IF GOBACK . END PROGRAM RECURSIVESUB. </pre>	<pre> 01 Arg USAGE BINARY-LONG. 01 Result USAGE BINARY-LONG SIGNED. PROCEDURE DIVISION USING Arg RETURNING Result. 000-Main. DISPLAY "Entering RECURSIVEFUNC Arg=" Arg IF Arg = 1 MOVE 1 TO Result ELSE COMPUTE Result = Arg * RECURSIVEFUNC (Arg - 1) END-IF DISPLAY "Leaving RECURSIVEFUNC Returning " Result GOBACK . END FUNCTION RECURSIVEFUNC. </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

When DEMOFACT is executed, the output shown to the right is generated.

```

E:\Programs\Demos>demofact
Entering RECURSIVESUB Arg=+0000000006
Entering RECURSIVESUB Arg=+0000000005
Entering RECURSIVESUB Arg=+0000000004
Entering RECURSIVESUB Arg=+0000000003
Entering RECURSIVESUB Arg=+0000000002
Entering RECURSIVESUB Arg=+0000000001
Leaving RECURSIVESUB Returning +0000000001
Leaving RECURSIVESUB Returning +0000000002=+0000000002*+0000000001
Leaving RECURSIVESUB Returning +0000000006=+0000000003*+0000000002
Leaving RECURSIVESUB Returning +0000000024=+0000000004*+0000000006
Leaving RECURSIVESUB Returning +0000000120=+0000000005*+0000000024
Leaving RECURSIVESUB Returning +0000000720=+0000000006*+0000000120
+0000000006! = +0000000720
Entering RECURSIVEFUNC Arg=+0000000006
Entering RECURSIVEFUNC Arg=+0000000005
Entering RECURSIVEFUNC Arg=+0000000004
Entering RECURSIVEFUNC Arg=+0000000003
Entering RECURSIVEFUNC Arg=+0000000002
Entering RECURSIVEFUNC Arg=+0000000001
Leaving RECURSIVEFUNC Returning +0000000001
Leaving RECURSIVEFUNC Returning +0000000002
Leaving RECURSIVEFUNC Returning +0000000006
Leaving RECURSIVEFUNC Returning +0000000024
Leaving RECURSIVEFUNC Returning +0000000120
Leaving RECURSIVEFUNC Returning +0000000720
+0000000006! = +0000000720

```

See Also...

The IDENTIFICATION DIVISION [3](#)

The CALL Statement [6.4.5](#)

Subprograms: Subroutines vs Functions [7.1](#)

7.8. Combining COBOL and C Programs

Linkage between GNU COBOL and C language programs is possible, but may require a little bit of special coding in one program or the other in order to meaningfully pass data between them. The issues involved deal predominantly with three topics, as follows. Each issue is discussed, with upcoming coding samples illustrating specifics as to how those issues are overcome in actual program code.

7.8.1. GNU COBOL Run-Time Library Requirements

Like most other implementations of the COBOL language, GNU COBOL utilizes a run-time library. When the first program executed in a given execution sequence is a GNU COBOL program, any run-time library initialization will be performed by that COBOL code in a manner that is transparent to the C-language programmer. If, however, a C program is the first to execute, the burden of perform GNU COBOL run-time library initialization falls upon the C program.

7.8.2. String Allocation Differences Between GNU COBOL and C

Both languages store strings as a fixed-length continuous sequence of characters.

COBOL stores these character sequences up to a specific quantity limit imposed by the **PICTURE** clause of the data item. For example:

```
01 LastName PIC X(15).
```

There is never an issue of exactly what the length of a string contained in a **USAGE DISPLAY** data item is – there are always exactly how ever many characters as were allowed for by the **PICTURE** clause. In the example above, “LastName” will always contain exactly fifteen characters; of course, there may be anywhere from 0 to 15 trailing **SPACES** as part of the current LastName value.

C actually has no “string” datatype – rather, it stores strings as an array of “char” datatype items where each element of the array is a single character. Being an array, there is an upper limit to how many characters may be stored in a given “string”. For example:

```
char lastName[15]; /* 15 chars: lastName[0] thru lastName[14] */
```

C provides a robust set of string-manipulation functions to copy strings from one char array to another, search strings for certain characters, compare one char array to another, concatenate char arrays and so forth. To make these functions possible, it was necessary to be able to define the logical end of a string. C accomplishes this via the expectation that all strings (char arrays) will be terminated by a NULL character (x'00'). Of course, no one forces a programmer to do this, but if [s]he ever expects to use any of the C standard functions to manipulate that string they had better be doing it.

So, GNU COBOL programmers expecting to pass strings to or receive strings from C programs had best be prepared to deal with the null-termination issue.

See Also...

Defining Data Items [5.2](#)

7.8.3. Matching C Data Types with GNU COBOL USAGES

This is pretty simple, the GNU COBOL and C programmer must just be aware of the following correspondence between C data types and COBOL USAGE specifications:

Figure 7-1 - C/GNU COBOL Data Type Matches

This COBOL USAGE... (no PICTURE allowed)	Occupies this space...	Holds these numeric values...	And corresponds to this C data type...
BINARY-CHAR BINARY-CHAR UNSIGNED	1 byte	0 to 255	unsigned char
BINARY-CHAR SIGNED	1 byte	-128 to +127	signed char
BINARY-SHORT BINARY-SHORT UNSIGNED	2 bytes	0 to 65535	unsigned unsigned int unsigned short unsigned short int
BINARY-SHORT SIGNED	2 bytes	-32768 to +32767	int short short int signed int signed short signed short int
BINARY-LONG BINARY-LONG UNSIGNED	4 bytes	0 to 4294967295	unsigned long unsigned long int
BINARY-LONG SIGNED BINARY-INT	4 bytes	-2147483648 to +2147483647	long long int signed long

This COBOL USAGE... (no PICTURE allowed)	Occupies this space...	Holds these numeric values...	And corresponds to this C data type...
			signed long int
BINARY-C-LONG SIGNED	4 bytes or 8 bytes	-2147483648 to +2147483647 Or -9223372036854775808 to +9223372036854775807	long (see the description of USAGE BINARY-C-LONG in Figure 5-10)
BINARY-DOUBLE BINARY-DOUBLE UNSIGNED	8 bytes	0 to 18446744073709551615	unsigned long long unsigned long long int
BINARY-DOUBLE SIGNED BINARY-LONG-LONG	8 bytes	-9223372036854775808 to +9223372036854775807	long long int signed long long int
COMPUTATIONAL-1	4 bytes	-3.4×10^{38} to $+3.4 \times 10^{38}$ (six decimal digits of precision)	float
COMPUTATIONAL-2	8 bytes	-1.7×10^{308} to $+1.7 \times 10^{308}$ (15 decimal digits of precision)	double
N/A (no GNU COBOL equivalent)	12 bytes	-1.19×10^{4932} to $+1.19 \times 10^{4932}$ (18 decimal digits of precision)	long double

There are other GNU COBOL PICTURE/USAGE combinations that can define the same storage size and value range combinations, but (with the exception of COMP-1 and COMP-2), these are the ANSI2002 standard specifications for C-program data compatibility and GNU COBOL programmers should get used to using them when data is being shared with C programs (they're good documentation too, highlighting the fact that the data will be "shared" with a C program).

The minimum values shown for the various SIGNED integer USAGES are appropriate for a computer system that uses 2s-complement representation for negative signed binary values (such as those CPUs typically found in Windows PCs). A computer system using 1s-complement representation for negative signed binary values would have minimum values that are 1 greater (-127 instead of -128, for example).

7.8.4. GNU COBOL Main Programs CALLing C Subprograms

Here are samples of a GNU COBOL program that **CALLs** a C subprogram.

Figure 7-2 - GNU COBOL CALLing C

(maincob.cbl) This GNU COBOL MAIN PROGRAM...	(subc.c) ...wants to CALL this C SubProgram
<pre> IDENTIFICATION DIVISION. PROGRAM-ID. maincob. DATA DIVISION. WORKING-STORAGE SECTION. 01 Arg1 PIC X(7). 01 Arg2 PIC X(7). 01 Arg3 USAGE BINARY-LONG. PROCEDURE DIVISION. 000-Main. DISPLAY 'Starting cobmain'. MOVE 123456789 TO Arg3. STRING 'Arg1' X'00' DELIMITED SIZE INTO Arg1 END-STRING. STRING 'Arg2' X'00' DELIMITED SIZE INTO Arg2 END-STRING. CALL 'subc' USING BY CONTENT Arg1, BY REFERENCE Arg2, BY REFERENCE Arg3. DISPLAY 'Back'. DISPLAY 'Arg1=' Arg1. DISPLAY 'Arg2=' Arg2. DISPLAY 'Arg3=' Arg3. DISPLAY 'Returned value=' RETURN-CODE. STOP RUN. </pre>	<pre> #include <stdio.h> int subc(char *arg1, char *arg2, unsigned long *arg3) { char nu1[7]="New1"; char nu2[7]="New2"; printf("Starting subc\n"); printf("Arg1=%s\n",arg1); printf("Arg2=%s\n",arg2); printf("Arg3=%d\n",*arg3); arg1[0]='X'; arg2[0]='Y'; *arg3=987654321; return 2; } </pre>

The idea is to pass two string and one full-word unsigned arguments to the subprogram, have the subprogram print them out, change all three and pass a return code of 2 back to the caller. The caller will then re-display the three arguments (showing changes only to the two BY REFERENCE arguments), display the return code and halt. While simple, these two programs illustrate the techniques required quite nicely.

Note how the COBOL program ensures that a null end-of-string terminator is present on both string arguments.

Since the C program is planning on making changes to all three arguments, it declares all three as pointers in the function header and references the third argument as a pointer in the function body.⁴⁰

⁴⁰ It actually had no choice for the two string (char array) arguments – they must be defined as pointers in the function even though the function code references them without the leading “*” that normally signifies pointers.

These programs are compiled and executed as follows. The example assumes a UNIX system with a GNU COBOL build that uses the native C compiler on that system; the technique works equally well regardless of which C compiler and which operating system you're using.

```
$ cc -c subc.c
$ cobc -x maincob.cbl subc.o
$ maincob
Starting cobmain
Starting subc
Arg1=Arg1
Arg2=Arg2
Arg3=123456789
Back
Arg1=Arg1
Arg2=Yrg2
Arg3=+0987654321
Returned value=+000000002
$
```

Remember that the null characters are actually in the GNU COBOL "Arg1" and "Arg2" data items. They don't appear in the output, but they ARE there. When passing character strings to C programs, it's probably a good idea to make a null-terminated copy of the string items and pass those copies to the C program.

7.8.5. C Main Programs CALLing GNU COBOL Subprograms

Now, the roles of the two languages in the previous section will be reversed, having a C main program execute a GNU COBOL subprogram.

Figure 7-3 - C CALLing GNU COBOL

(mainc.c) This C MAIN PROGRAM...	(subcob.cbl) ...wants to CALL this GNU COBOL SubProgram
<pre>#include <stdio.h> int main (int argc, char **argv) { int returnCode; char arg1[7] = "Arg1"; char arg2[7] = "Arg2"; unsigned long arg3 = 123456789; printf("Starting mainc...\n"); returnCode = subcob(arg1,arg2,&arg3); printf("Back\n"); printf("Arg1=%s\n",arg1); printf("Arg2=%s\n",arg2); printf("Arg3=%d\n",arg3); printf("Returned value=%d\n",returnCode); return returnCode; }</pre>	<pre>IDENTIFICATION DIVISION. PROGRAM-ID. subcob. DATA DIVISION. LINKAGE SECTION. 01 Arg1 PIC X(7). 01 Arg2 PIC X(7). 01 Arg3 USAGE BINARY-LONG. PROCEDURE DIVISION USING BY VALUE Arg1, BY REFERENCE Arg2, BY REFERENCE Arg3. 000-Main. DISPLAY 'Starting cobsub.cbl'. DISPLAY 'Arg1=' Arg1. DISPLAY 'Arg2=' Arg2. DISPLAY 'Arg3=' Arg3. MOVE 'X' TO Arg1 (1:1). MOVE 'Y' TO Arg2 (1:1). MOVE 987654321 TO Arg3. MOVE 2 TO RETURN-CODE. GOBACK.</pre>

Since the C program is the one that will execute first, before the GNU COBOL subroutine, the burden of initializing the GNU COBOL run-time environment lies with that C program; it will have to invoke the "cob_init" function, which is part of the "libcob" library. The two required C statements are shown

The arguments to the "cob_init" routine are the argument count and value parameters passed to the main function when the program began execution. By passing them into the GNU COBOL subprogram, it will be possible for that GNU COBOL program to retrieve the command line or individual command-line arguments. If that won't be necessary, "cob_init(0,NULL);" could be specified instead.

Since the C program wants to allow "arg3" to be changed by the subprogram, it prefixes it with a "&" to force a **CALL BY REFERENCE** for that argument. Since "arg1" and "arg2" are strings (char arrays), they are automatically passed by reference.

Here's the output of the compilation process as well as the program's execution. The example assumes a Windows system with a GNU COBOL build that uses the GNU C compiler on that system; the technique works equally well regardless of which C compiler and which operating system you're using.

```
C:\Users\Gary\Documents\Programs> cobc -S subcob.cbl
C:\Users\Gary\Documents\Programs> gcc mainc.c subcob.s -o mainc.exe -llibcob
C:\Users\Gary\Documents\Programs> mainc.exe
Starting mainc...
Starting cobsu.cbl
Arg1=Arg1
Arg2=Arg2
Arg3=+0123456789
Back
Arg1=Xrg1
Arg2=Yrg2
Arg3=987654321
Returned value=2
C:\Users\Gary\Documents\Programs>
```

Note that even though we told GNU COBOL that the 1st argument was to be BY VALUE, it was treated as if it were BY REFERENCE anyway. String (char array) arguments passed from C callers to GNU COBOL subprograms will be modifiable by the subprogram. It's best to pass a copy of such data if you want to ensure that the subprogram doesn't change it.

The third argument is different, however. Since it's not an array you have the choice of passing it either BY REFERENCE⁴¹ or BY VALUE⁴².

⁴¹ Use "&" with the argument in the C calling program; specify the argument as BY REFERENCE in the COBOL subprogram

⁴² Don't use "&" with the argument in the C calling program; specify the argument as BY VALUE in the COBOL subprogram

8. The GNU COBOL System Interface

8.1. Using the GNU COBOL Compiler (cobc)

8.1.1. Introduction

Program source files should have extensions of “.cob” or “.cbl”.

Program filenames should match exactly the specification of PROGRAM-ID (including case). The reason for this was discussed in section 3.

Spaces cannot be included in primary entry-point names (section 3) and therefore should not be included in program filenames.

The GNU COBOL compiler will translate your COBOL program into C source code, compile that C source code into executable binary form using the “C” compiler specified when GNU COBOL was built and link that executable binary into either directly executable form, static-linkable form or dynamically-loadable executable form.

The GNU COBOL compiler is named “cobc” (“cobc.exe” on a Windows system).

8.1.2. Syntax and Options

The following describes the syntax and option switches of the cobc command. This information may be displayed by entering the command “cobc --help”.

Usage: cobc [options] file ...

Options:

-help	Display this message
-version, -V	Display compiler version
-info, -i	Display compiler build information
-v	Display the commands invoked by the compiler
-x	Build an executable program
-m	Build a dynamically loadable module (default)
-std=<dialect>	Warnings/features for a specific dialect : cobol2002 Cobol 2002 cobol85 Cobol 85 ibm IBM Compatible mvs MVS Compatible bs2000 BS2000 Compatible mf Micro Focus Compatible default When not specified See config/default.conf and config/*.conf
-free	Use free source format
-fixed	Use fixed source format (default)
-O, -O2, -Os	Enable optimization
-g	Enable C compiler debug / stack check / trace
-debug	Enable all run-time error checking
-o <file>	Place the output into <file>
-b	Combine all input files into a single dynamically loadable module
-E	Preprocess only; do not compile or link
-C	Translation only; convert COBOL to C
-S	Compile only; output assembly file
-c	Compile and assemble, but do not link
-P(=<dir or file>)	Generate preprocessed program listing (.lst)
-Xref	Generate cross reference through 'cobxref' (V. Coen's 'cobxref' must be in path)
-I <directory>	Add <directory> to copy/include search path
-L <directory>	Add <directory> to library search path
-l <lib>	Link the library <lib>
-A <options>	Add <options> to the C compile phase
-Q <options>	Add <options> to the C link phase
-D <define>	DEFINE <define> to the COBOL compiler
-K <entry>	Generate CALL to <entry> as static
-conf=<file>	User defined dialect configuration - See -std=
-list-reserved	Display reserved words
-list-intrinsics	Display intrinsic functions
-list-mnemonics	Display mnemonic names

```

-list-system          Display system routines
-save-temps(=<dir>)  Save intermediate files
                    - Default : current directory
-ext <extension>    Add default file extension

-W                  Enable ALL warnings
-Wall              Enable all warnings except as noted below
-Wobsolete         Warn if obsolete features are used
-Warchaic          Warn if archaic features are used
-Wredefinition     Warn incompatible redefinition of data items
-Wconstant         Warn inconsistent constant
-Woverlap          Warn overlapping MOVE items
-Wparentheses      Warn lack of parentheses around AND within OR
-Wstrict-typing    Warn type mismatch strictly
-Wimplicit-define  Warn implicitly defined data items
-Wcorresponding    Warn CORRESPONDING with no matching items
-Wexternal-value   Warn EXTERNAL item with VALUE clause
-Wcall-params      Warn non 01/77 items for CALL params
                    - NOT set with -Wall
-Wcolumn-overflow  Warn text after column 72, FIXED format
                    - NOT set with -Wall
-Wterminator       Warn lack of scope terminator END-XXX
                    - NOT set with -Wall
-Wtruncate         Warn possible field truncation
                    - NOT set with -Wall
-Wlinkage          Warn dangling LINKAGE items
                    - NOT set with -Wall
-Wunreachable      Warn unreachable statements
                    - NOT set with -Wall

-fsign=<value>      Define display sign representation
                    - ASCII or EBCDIC (Default : machine native)
-ffold-copy=<value> Fold COPY subject to value
                    - UPPER or LOWER (Default : no transformation)
-ffold-call=<value> Fold PROGRAM-ID, CALL, CANCEL subject to value
                    - UPPER or LOWER (Default : no transformation)
-fdefaultbyte=<value> Initialize fields without VALUE to decimal value
                    - 0 to 255 (Default : initialize to picture)
-fintrinsics=<value> Intrinsic to be used without FUNCTION keyword
                    - ALL or intrinsic function name (,name,...)
-ftrace            Generate trace code
                    - Executed SECTION/PARAGRAPH
-ftraceall         Generate trace code
                    - Executed SECTION/PARAGRAPH/STATEMENTS
                    - Turned on by -debug
-fsyntax-only      Syntax error checking only; don't emit any output
-fdebugging-line   Enable debugging lines
                    - 'D' in indicator column or floating >>D
-fsource-location  Generate source location code
                    - Turned on by -debug/-g/-ftraceall
-fimplicit-init    Automatic initialization of the Cobol runtime system
-fstack-check      PERFORM stack checking
                    - Turned on by -debug or -g
-fsyntax-extension Allow syntax extensions
                    - eg. Switch name SW1, etc.
-fwrite-after      Use AFTER 1 for WRITE of LINE SEQUENTIAL
                    - Default : BEFORE 1
-fmfcomment        '*' or '/' in column 1 treated as comment
                    - FIXED format only
-fnotrunc         Allow numeric field overflow
                    - Non-ANSI behaviour
-fodoslide        Adjust items following OCCURS DEPENDING
                    - Requires implicit/explicit relaxed syntax
-fsingle-quote     Use a single quote (apostrophe) for QUOTE
                    - Default : double quote
-frecursive-check  Check recursive program call
-frelax-syntax     Relax syntax checking
                    - eg. REDEFINES position
-foptional-file    Treat all files as OPTIONAL
                    - unless NOT OPTIONAL specified

```

As discussed in section 2, program compilation groups may consist of multiple programs defined sequentially in a single source file. By specifying multiple source files on the “cobc” command, it is possible for a single execution of the “cobc” command to process multiple compilation groups.

8.1.3. Compiling GNU COBOL Programs

8.1.3.1. Compiling Directly-Executable GNU COBOL Programs

The simplest mode of compilation is to generate a single executable file from one or more GNU COBOL source files:

```
cobc -x prog1.cbl prog2.cbl prog3.cbl
```

The main program must be the first program found in the “prog1.cbl” file. The remainder of “prog1.cbl” as well as all of “prog2.cbl” and “prog3.cbl” must be subprograms (subroutines or user-defined functions) or nested subprograms.

This will generate a single executable file (UNIX) or exe file (Windows) which has all COBOL programs contained within the source files specified on the “cobc” command included in the file. The first program found in the first specified source file is presumed to be the main program and all other programs found in the remainder of that first source file as well as in all the remaining source files will be static subroutines and/or user-defined functions. Any subroutines or user-defined functions that weren’t included in any of the source files will be treated as dynamically loadable subprograms.

Optionally, the “-o” option may be used to specify the name of the generated executable file. If “-o” is not specified, otherwise, the filename of the 1st source file named on the command will be used. The appropriate extension for the generated file (“exe”, on a Windows computer, for example) will be added to the filename that is explicitly or implicitly used for the output file.

8.1.3.2. Compiling Dynamically-Loadable GNU COBOL Subprograms

Subprograms that are to be dynamically loaded into memory at execution time must be compiled using the “-m” option on the cobc command, as follows:

```
cobc -m sprog1.cbl
cobc -m sprog1.cbl sprog2.cbl sprog3.cbl
cobc -m -b sprog1.cbl sprog2.cbl sprog3.cbl
```

The first command above generates a single dynamically-loadable module. The second example generates three dynamically-loadable modules (one for each source file). The third command generates a single dynamically-loadable module.

Optionally, when a single output file is being generated, the “-o” option may be used to specify its name (otherwise, the filename of the 1st source file named on the command will be used). The appropriate extension for the generated file (“dll”, on a Windows computer, for example) will be added to the filename that is explicitly or implicitly used for the output file.

It is also possible to generate main programs as dynamically-loadable libraries. Just use the “-m” option (as shown here) rather than the “-x” option. To execute these main programs, you’ll need to utilize the cobcrun command, as discussed in section [8.2.2](#).

8.1.3.3. Compiling Static GNU COBOL Subprograms

You may compile GNU COBOL subprograms into assembler source code which can then be assembled and linked with a main program when that main program is compiled. To create such an assembler source file, compile the subprogram(s) as follows:

```
cobc -S sprog1.cbl
```

(Note: “-S” is an uppercase-S)

This will create an assembler source file named “sprog1.s”. If you specify multiple input files, they’ll each create their own “.s” files.

To compile a main program, assemble an assembler source file and static-link it all together:

```
cobc -x mainprog.cbl sprog1.s
```

If multiple subprograms are needed, simply add their “.s” files to the command line. Any subprogram *entry-points* for which “.s” files were not specified will be **CALL**ed at runtime as dynamically-loadable modules.

Precompiled subroutines intended to be statically linked (usually they end in “.o”) may be automatically located by the GNU COBOL compiler (cobc) and the loader (ld) by using the LD_LIBRARY_PATH environment variable (section [8.1.4](#)).

8.1.4. Important Compilation-Time Environment Variables

The following chart documents the various environment variables that can play a role in the compilation of GNU COBOL programs.

Figure 8-1 - Compiler Environment Variables

Environment Variable	Use
COB_CC	Set to the name of the C compiler you wish GNU COBOL to use. USE THIS FEATURE AT YOUR OWN RISK – YOU SHOULD ALWAYS USE THE C COMPILER YOUR GNU COBOL BUILD WAS GENERATED FOR
COB_CFLAGS ⁴³	Set to any switches that you’d like to pass on to the C compiler from the cobc compiler (in addition to any that cobc will specify). The default is “-l <i>prefix</i> /include”, where “ <i>prefix</i> ” is the path prefix specified when the GNU COBOL binaries you are using were created.
COB_CONFIG_DIR	Set to the path to the folder where GNU COBOL “config” files are kept.
COB_COPY_DIR	If copybooks your program needs are NOT stored in the same directory as your program, set this environment variable to the folder in which the copybooks may be found (IBM mainframe programmers will recognize this as “SYSLIB”).
COB_LDADD ⁴¹	Set to any additional linker switches (ld) that can specify where standard libraries that must be linked with the program can be found. The default is “” (null).
COB_LDFLAGS ⁴¹	Set to any linker/loader (ld) switches that you’d like to pass on to the C compiler from the cobc compiler (in addition to any that cobc will specify). The default is none.
COB_LIBS ⁴¹	Set to any linker switches (ld) that specify where standard libraries that must be linked with the program can be found. The default is “-L <i>prefix</i> /lib -lcob”, where “ <i>prefix</i> ” is the path prefix specified when the GNU COBOL binaries you are using were created.
COBCPY	This environment variable provides an additional means of specifying where copybooks may be found by the compiler (see also COB_COPY_DIR, above).
LD_LIBRARY_PATH	If you are planning on using static-linked subroutine libraries, set this variable to the path to the directory containing your libraries.

⁴³ These switches are intended for use only in very special circumstances by very advanced users; their usage is discouraged. A future release of GNU COBOL will introduce a better way to pass switched to the C compiler and/or the loader from the cobc command.

Environment Variable	Use
TMPDIR TMP (checked in this order)	Set to a directory/folder appropriate to create temporary files in. The intermediate working files created by cobc will be created here (and deleted once they're no longer needed). On a Windows system, the TMP environment variable is normally set for you when you logon. If you wish to use a <u>different</u> temporary folder, you may set TMPDIR yourself and have no fear of disrupting other Windows software that relies on TMP.

See Also...

Copybooks 1.3.3.3	Compiler Switches Reference 8.1.2
The COPY Statement 2.1.1	GNU COBOL "config" Files 8.1.6

8.1.5. Locating Copybooks at Compilation Time

The GNU COBOL compiler will attempt to locate copybooks by searching for them in the following folders. The search will occur in the sequence shown below, and will terminate once a copybook is found.

- ▶ The folder named as the *library-name-1* on the **COPY** statement.
- ▶ The folder in which the program being compiled resides.
- ▶ The folder named on the "-I" compiler switch
- ▶ Each of the folders named on the COBCPY environment variable (see section [8.1.4](#)). A single folder may be named or multiple folders may be specified, separated by a system-appropriate delimiter character.⁴⁴ When multiple folders are specified, they will be searched in the order they are named on the environment variable.
- ▶ The folder specified on the COB_COPY_DIR environment variable.

As each of the above folders is searched for a copybook - "COPY XXXXXXXX.", for example – the GNU COBOL compiler will attempt to locate the copybook file by any of the following names, in the sequence shown:

- ▶ XXXXXXXX.CPY
- ▶ XXXXXXXX.CBL
- ▶ XXXXXXXX.COB
- ▶ XXXXXXXX.cpy
- ▶ XXXXXXXX.cbl
- ▶ XXXXXXXX.cob
- ▶ XXXXXXXX

The COPY command is case-sensitive on UNIX systems; "COPY copybookname" and "COPY COPYBOOKNAME" will both fail to locate the "CopyBookName" copybook on a UNIX system. Windows implementations of GNU COBOL may or may not be similarly case sensitive with regard to copybook names, depending upon the Windows version and GNU COBOL build options – it is safest to simply treat the COPY command as case-sensitive in all environments.

See Also...

Copybooks 1.3.3.3	Compiler Switches Reference 8.1.2
The COPY Statement 2.1.1	Compilation-time Environment Variables 8.1.4

8.1.6. Using Compiler Configuration Files

GNU COBOL uses compiler configuration files to define various options that will control the compilation process. These configuration files are specified using the "-conf" compilation switch or are found in the folder defined by the COB_CONFIG_PATH environment variable.

The following is a verbatim listing of the "default" configuration file (the one used if you don't specify the "-conf" switch), just to show you the types of settings that may appear:

```
# COBOL compiler configuration                                -*- sh -*-
```

⁴⁴ If the GNU COBOL compiler you are using was built to utilize a native Windows environment, use a semicolon (;). If, however, the GNU COBOL compiler was built for a Unix or Linux environment, or was built for a Windows environment utilizing either the Cygwin or MinGW Unix "emulators", use a colon character (:) as the separator.

```

# Value: any string
name: "GNU COBOL"

# Value: int
tab-width: 8
text-column: 72

# Value: 'cobol2002', 'mf', 'ibm'
#
assign-clause: mf

# If yes, file names are resolved at run time using environment variables.
# For example, given ASSIGN TO "DATAFILE", the actual file name will be
# 1. the value of environment variable 'DD_DATAFILE' or
# 2. the value of environment variable 'dd_DATAFILE' or
# 3. the value of environment variable 'DATAFILE' or
# 4. the literal "DATAFILE"
# If no, the value of the assign clause is the file name.
#
# Value: 'yes', 'no'
filename-mapping: yes

# Value: 'yes', 'no'
pretty-display: yes

# Value: 'yes', 'no'
auto-initialize: yes

# Value: 'yes', 'no'
complex-odo: no

# Value: 'yes', 'no'
indirect-redefines: no

# Binary byte size - defines the allocated bytes according to PIC
# Value:          signed  unsigned  bytes
#
# '2-4-8'         1 - 4      5 - 9      10 - 18     2
#
#                 5 - 9      10 - 18     8
#
# '1-2-4-8'       1 - 2      3 - 4      5 - 9      10 - 18     1
#
#                 3 - 4      5 - 9      10 - 18     2
#
#                 5 - 9      10 - 18     8
#
# '1--8'          1 - 2      3 - 4      5 - 6      7 - 9      10 - 11    12 - 14    15 - 16    17 - 18     1
#
#                 3 - 4      5 - 6      7 - 9      10 - 11    12 - 14    15 - 16    17 - 18     2
#
#                 5 - 6      7 - 9      10 - 11    12 - 14    15 - 16    17 - 18     3
#
#                 7 - 9      10 - 11    12 - 14    15 - 16    17 - 18     4
#
#                 10 - 11   12 - 14    15 - 16    17 - 18     5
#
#                 12 - 14   15 - 16    17 - 18     6
#
#                 15 - 16   17 - 18     7
#
#                 17 - 18     8
binary-size: 1-2-4-8

# Value: 'yes', 'no'
binary-truncate: yes

# Value: 'native', 'big-endian'
binary-byteorder: big-endian

# Value: 'yes', 'no'
larger-redefines-ok: no

# Value: 'yes', 'no'
relaxed-syntax-check: no

# Perform type OSVS - If yes, the exit point of any currently executing perform

```

```

# is recognized if reached.
# Value: 'yes', 'no'
perform-osvs: no

# If yes, linkage-section items remain allocated
# between invocations.
# Value: 'yes', 'no'
sticky-linkage: no

# If yes, allow non-matching level numbers
# Value: 'yes', 'no'
relax-level-hierarchy: no

# not-reserved:
# Value: Word to be taken out of the reserved words list
# (case independent)

# Dialect features
# Value: 'ok', 'archaic', 'obsolete', 'skip', 'ignore', 'unconformable'
author-Paragraph                obsolete
memory-size-clause:             obsolete
multiple-file-tape-clause:      obsolete
label-records-clause:           obsolete
value-of-clause:                obsolete
data-records-clause:            obsolete
top-level-occurs-clause:        skip
synchronized-clause:           ok
goto-statement-without-name:    obsolete
stop-literal-Statement          obsolete
debugging-line:                 obsolete
padding-character-clause:       obsolete
next-sentence-phrase:          archaic
eject-Statement                 skip
entry-Statement                 obsolete
move-noninteger-to-alphanumeric: error
odo-without-to:                 ok

```

8.2. Running GNU COBOL Programs

8.2.1. Executing Programs Directly

GNU COBOL programs compiled with the “-x” option will be generated as directly-executable programs. For example, on a Windows system, the “-x” option will be generated as an “.exe” file.

These native executables are appropriate for execution as non-graphical user interface programs.

On a UNIX system this means the programs may be executed from a command shell such as bash, csh, ksh and so forth. When a GNU COBOL program runs on a Windows system, it runs within a console window (i.e. “cmd.exe”).

Interactions between the program and the user will take place using the standard input, standard output and standard error streams. Any SCREEN SECTION I/O performed by the program will take place within the command shell “window”.

Direct program execution syntax is as follows:

[path]program [arguments]

For example:

/usr/local/printaccount ACCT=6625378

Or...

C:\Users\Me\Documents\Programs\printaccount.exe ACCT=6625378

8.2.2. Using the “cobcrun” Utility

It is possible to generate executable modules for all GNU COBOL programs, not just subroutines, by choosing to use the “-m” option to specify the compiler output format even for main programs.

Some may prefer to compile their GNU COBOL main programs into these dynamically-loadable modules in the interests of using the same general compilation command for all programs without having to think “Is it a main program or a subroutine?”.

Main programs compiled in this manner should be executed as follows:

[path]cobcrun program [arguments]

Do not specify the “.so” or “.dll” extension on the program name. The “program” value must exactly match the primary entry-point name (section 3) of the main program (including upper- and lower-case letters).

The general usage and syntax of cobcrun is as follows:

```
Usage: cobcrun PROGRAM [param ...]
or   : cobcrun --help (-h)
      Print this help
or   : cobcrun --version (-V)
      Print version information
or   : cobcrun --info (-i)
      Print build information
```

For an example of the use of cobcrun:

```
cd /usr/local
cobcrun printaccount ACCT=6625378
```

Or...

```
cd C:\Users\Me\Documents\Programs
cobcrun printaccount.exe ACCT=6625378
```

Note how the cobcrun command does not allow a path to be specified with the program name –the directory in which the programs dynamically loadable module exists must either be the current directory or must be defined in the current PATH.

See Also...

Compiler Switches Reference [8.1.2](#)

8.2.3. Program Arguments

Regardless of the manner in which a program is executed, any arguments specified to the program may be retrieved via either of the following,:

- ▶ ACCEPT ... FROM COMMAND-LINE
- ▶ ACCEPT ... FROM ARGUMENT-VALUE

▶ *See Also...*

The ACCEPT Statement (Command Line) [6.2.1.2](#)

8.2.4. Important Execution-Time Environment Variables

The following chart documents the various environment variables that can play a role in the execution of GNU COBOL programs.

Figure 8-2 - Run-Time Environment Variables

Environment Variable	Use
COB_DISPLAY_WARNINGS	If set to a value of “Y”, any run-time warnings (such as noting the implicit CLOSE of open files when a GOBACK or STOP RUN is executed) will be displayed. Any other value for this environment variable (including not setting the variable at all) will suppress such messages.

Environment Variable	Use
COB_LIBRARY_PATH	At runtime, GNU COBOL will attempt to locate and load any application dynamically-loadable libraries from the PATH and the directory in which the program executable was found. If these library files could be somewhere else, specify the directory path using this variable.
COB_PRE_LOAD	If set to any non-null value, this variable will cause all dynamically-loadable libraries to be loaded when the program begins execution (rather than searching for and loading the module upon first use).
COB_SET_DEBUG	If a USE FOR DEBUGGING section is included in DECLARATIVES , the code within it will be disabled unless this environment variable is set to a value of “Y”, “y” or “1”.
COB_SET_TRACE	If the “ -ftrace ” or “ -ftraceall ” options were used when the program was compiled, setting this environment variable to a value of “Y” will activate the trace at the point the program begins execution. Setting this environment variable to any other value (or never setting it to ANY value) will disable tracing. See the READY TRACE and RESET TRACE statements for additional ways to control tracing.
COB_SCREEN_ESC	If set to any non-blank value, this variable allows the ACCEPT verb to detect the Esc key. See Figure 6-23 for additional information.
COB_SCREEN_EXCEPTIONS	Setting this variable to any non-blank value will allow the ACCEPT verb to detect the Esc, PgUp and PgDn keys. See Figure 6-23 for additional information.
COB_SORT_MEMORY	The value of this variable (an integer) will be used to define how much memory will be allocated for use in sorting. If the value is 1048576 or greater, that value will be used “as is” as the amount of memory (in bytes) to allocate. If the value is less than 1048576. The default sort memory amount is 128 MB.
COB_SWITCH_n	(n=0 to 15); These environment variables correspond to SWITCH-0 through SWITCH-15 , defined in the SPECIAL-NAMES paragraph. Setting them to “ON” will activate them; any other value turns them off.
COB_SYNC	If set to a value of upper- or lowercase “p”, this variable will force a file commit every time a file is written to (ensuring that data is <u>immediately</u> written to the file rather than retained in memory until a future commit occurs). This will slow-down update access to files, but will provide for better integrity in the event of a program failure.
COB_TRACE_FILE	If set to a value, this environment variable specifies the file to which all -ftrace and -ftraceall output will be written. If this is NOT set to a value, all -ftrace and -ftraceall output will be written to STDERR , where it may be piped via a “2> filename” on the command that executes the program.

Environment Variable	Use
DB_HOME	If your GNU COBOL build uses the Berkeley Database (BDB) package, use this environment variable to specify the folder in which the lock management files to be associated with all non- SORT files opened by the program will be stored ⁴⁵ . Having this variable defined will activate record locking features on the READ , REWRITE and WRITE statements ⁴⁶ .
PATH	The GNU COBOL “bin” directory should be defined in the PATH .
TMPDIR TMP TEMP (checked in this order)	Set to a directory/folder appropriate to create temporary files in. This will be used by SORT and MERGE to create temporary work files. You may also use this folder for any temporary files your application may require. Good form dictates that – if your application DOES create temporary working files – it should clean-up after itself. ⁴⁷

See Also...

The SPECIAL-NAMES Paragraph 4.1.4	The READY TRACE Statement 6.2.32
Using DECLARATIVES 6.1.4	The RESET TRACE Statement 6.4.34
The ACCEPT Statement (Screen Data) 6.4.1.4	The REWRITE Statement 6.4.36
The CLOSE Statement 6.4.7	The SORT Statement (File Sort) 6.4.40.1
The GOBACK Statement 6.2.19	The STOP RUN Statement 6.4.42
The MERGE Statement 6.4.25	The WRITE Statement 6.4.50
The READ Statement 6.4.31	Compiler Switches Reference 8.1.2

8.3. Built-In System Subroutines

8.3.1. “Call by Name” Routines

There are a number of built-in system subroutines included with GNU COBOL. Generally, these routines are intended to match those available in Micro Focus COBOL (CBL_...) or ACUCOBOL (C\$...).

These routines, all executed via their UPPERCASE NAMES, are capable of performing the following Functions

- ▶ Changing the current directory
- ▶ Copying files
- ▶ Creating a directory
- ▶ Creating, Opening, Closing, Reading and Writing byte-stream files
- ▶ Deleting directories (folders)
- ▶ Deleting files
- ▶ Determining how many arguments were passed to a subroutine

⁴⁵ **ORGANIZATION INDEXED** files will also have their data file allocated in the DB_HOME folder, if DB_HOME exists.

⁴⁶ Even with DB_HOME, locking will not work with **ORGANIZATION SEQUENTIAL** (either type) or **ORGANIZAION RELATIVE** files with GNU COBOL builds created for Windows/MinGW. **ORGANIZATION INDEXED** locks will work with Windows/MinGW + BDB and all locks will work for all file organizations with UNIX GNU COBOL builds.

⁴⁷ Take a look at the **C\$DELETE** and **CBL_DELETE_FILE** built-in subroutines.

- ▶ Getting file information (size and last-modification date/time)
- ▶ Getting the length (in bytes) of an argument passed to a subroutine
- ▶ Justifying a field left-, right- or center-aligned
- ▶ Moving files (a destructive “copy”)
- ▶ Putting the program ‘to sleep’, specifying the sleep time in seconds
- ▶ Putting the program ‘to sleep’, specifying the sleep time in nanoseconds; CAVEAT: although you’ll express the time in nanoseconds, Windows systems will only be able to sleep at a millisecond granularity
- ▶ Retrieving information about the currently-executing program
- ▶ Submitting a command to the shell environment appropriate for the version of GNU COBOL you are using for execution

The following table describes the various built-in subroutines. ALL SUBROUTINE ARGUMENTS ARE MANDATORY EXCEPT WHERE EXPLICITLY NOTED TO THE CONTRARY. Any subroutine returning a value to **RETURN-CODE** could utilize the **RETURNING/GIVING** clause on the **CALL** to return the result back to the full-word binary COMP-5 data item of your choice.

See Also...

The CALL Statement 6.4.5

8.3.1.1. CALL “C\$CALLED BY” USING *prog-name-area*

This routine returns the name of the program that **CALL**ed the currently-executing program. The program name will be returned, left-justified and SPACE filled, in the specified *prog-name-area* argument, which should be a PIC X elementary item or a group item. If *prog-name-area* is too small to receive the entire program name, the program name value will be truncated to fit the size of the argument.

The **RETURN-CODE** register will be set to one of the following values:

- 1 An error occurred. The *prog-name-area* contents will be unchanged.
- 0 The program **CALL**ing “C\$CALLED BY” was not called by any other program (in other words, it is a main program,). The *prog-name-area* contents will be set entirely to SPACES.
- 1 The program **CALL**ing “C\$CALLED BY” was indeed called by another program, and that program’s name has been saved in *prog-name-area*.

8.3.1.2. CALL “C\$CHDIR” USING *directory-path, result*

This routine makes *directory-path* (an alphanumeric literal or identifier) the current directory.

The return code of the operation is returned both in the *result* argument (any non-edited numeric identifier) as well as in the **RETURN-CODE** special register. The return code of the operation will be either 0=Success or 128=failure.

The directory change remains in effect until the program terminates (in which the original current directory at the time the program was restarted will be automatically restored) or until another C\$CHDIR is executed.

8.3.1.3. CALL “C\$COPY” USING *src-file-path, dest-file-path, 0*

Use this subroutine to copy file *src-file-path* to *dest-file-path* as if it were done via the “CP” (Unix) or “COPY” (Windows) command.

Both file path arguments may be alphanumeric literals or identifiers.

The third argument is required, but is unused.

If the attempt to copy the file fails (for example, it or the destination directory doesn't exist), **RETURN-CODE** will be set to 128; on successful completion it will be set to 0.

8.3.1.4. CALL “C\$DELETE” USING *file-path*, 0

This routine deletes the file specified by the *file-path* argument (an alphanumeric literal or identifier) just as if that were done using the “RM” (Unix) or “ERASE” (Windows) command.

The second argument is required, but is unused.

If the attempt to delete the file fails (for example, it doesn't exist), **RETURN-CODE** will be set to 128; on successful completion it will be set to 0.

8.3.1.5. CALL “C\$FILEINFO” USING *file-path*, *file-info*

With this routine you may retrieve the size of the file⁴⁸ specified as the *file-path* argument (an alphanumeric literal or identifier) and the date/time that file was last modified. The information is returned to the *file-info* argument, which is defined as the following 16-byte area:

```

01 File-Info.
   05 File-Size-In-Bytes PIC 9(18) COMP.
   05 Mod-YYYYMMDD      PIC 9(8)  COMP. *> Modification Date
   05 Mod-HHMMSS00     PIC 9(8)  COMP. *> Modification Time

```

The last two decimal digits in the modification time will always be 0.

If the subroutine is successful, a value of 0 will be returned in **RETURN-CODE**. Failure to retrieve the needed statistics on the file will cause a **RETURN-CODE** value of 35 to be passed back. Supplying less than two arguments will generate a 128 **RETURN-CODE** value.

8.3.1.6. CALL “C\$GETPID”

Use the C\$GETPID to return the PID of the executing GNU COBOL program. The PID value is returned into the **RETURN-CODE** register.

As you can see, there are no arguments to this routine.

8.3.1.7. CALL “C\$JUSTIFY” USING *data-item*, “*justification-type*”

Use C\$JUSTIFY to left, right or center-justify an alphabetic, alphanumeric or numeric edited *data-item*. The *justification-type* argument indicates the type of the justification to be performed. The value of that argument will be interpreted as follows:

absent	Treated the same as if it were "R"
Cxxx...	If it begins with a capital "C", the value will be centered
Rxxx...	If it begins with a capital "R", the value will be right-justified, space-filled to the left
Lxxx...	If it begins with a capital "L", the value will be left-justified, space-filled to the right
anything else	Treated as if it were "R"

8.3.1.8. CALL “C\$MAKEDIR” USING *dir-path*

With this routine you may create a new directory – the name of which is supplied as the *dir-path* argument (an alphanumeric literal or identifier).

Only the lowest-level directory (last) in the specified path can be created – all others must already exist. This subroutine will NOT behave as a “mkdir -p” (Unix) or “mkdir /p” (Windows).

RETURN-CODE will be set to the return code of the operation; the value will be either 0=Success or 128=failure.

⁴⁸ File size information may not be available in the particular GNU COBOL build / Operating System combination you are using and may therefore always be returned as zero.

8.3.1.9. CALL “C\$NARG” USING *arg-count-result*

C\$NARG returns the number of arguments passed to a [subroutine](#) that calls C\$NARG back to the numeric field *arg-count-result*. When called from within a user-defined function, a value of one (1) is returned if any arguments were passed to the function or a zero (0) otherwise.

When CALLED from a main program, the returned value will always be 0.

8.3.1.10. CALL “C\$PARAMSIZE” USING *argument-number*

This subroutine returns the size (in bytes) of the subroutine argument supplied using the *argument-number* parameter (a numeric literal or data item).

The size is returned in the RETURN-CODE special register.

If the specified argument does not exist, or an invalid argument number is specified, a value of 0 is returned.

8.3.1.11. CALL “C\$PRINTABLE” USING *data-item* [, *char*]

The C\$PRINTABLE subroutine converts the contents of the *data-item* specified as the first argument to printable characters. Those characters that are deemed printable (as defined by the character set used by *data-item*) will remain unchanged, while those that are NOT printable will be converted to the character specified as the second argument. If no second argument is provided, a period (“.”) will be used.

8.3.1.12. CALL “C\$SLEEP” USING *seconds-to-sleep*

C\$SLEEP puts the program to sleep for the specified number of seconds. The *seconds-to-sleep* argument may be a numeric literal or data item.

Sleep times less than 1 will be interpreted as 0, which immediately returns without any sleep delay.

8.3.1.13. CALL “C\$TOLOWER” USING *data-item*, BY VALUE *convert-length*

This routine will convert *convert-length* (a numeric literal or data item) leading characters of *data-item* (an alphanumeric identifier) to lower-case.

The *convert-length* argument must be specified . It specifies how many (leading) characters in *data-item* will be converted – any characters after that will remain unchanged.

If *convert-length* is negative or zero, no conversion will be performed.

8.3.1.14. CALL “C\$TOUPPER” USING *data-item*, BY VALUE *convert-length*

Use the C\$TOUPPER subroutine to change the *convert-length* (a numeric literal or data item) leading characters of *data-item* (an alphanumeric identifier) to upper-case.

The *convert-length* argument must be specified . It specifies how many (leading) characters in *data-item* will be converted – any characters after that will remain unchanged.

If *convert-length* is negative or zero, no conversion will be performed.

8.3.1.15. CALL “CBL_AND” USING *item-1*, *item-2*, BY VALUE *byte-length*

This subroutine performs a bit-by-bit logical AND operation between the left-most 8**byte-length* corresponding bits of *item-1* and *item-2*, storing the resulting bit string into *item-2*.

Item-1 may be an alphanumeric literal or a data item. *Item-2* must be a data item. The length of both *item-1* and *item-2* must be at least 8**byte-length*.

Byte-length may be a numeric literal or data item, and must be specified using .

The truth table shown to the right documents the “AND” process.

Any bits in *item-2* after the 8**byte-length* point will be unaffected.

A result of zero will be passed back in the RETURN-CODE register.

Arg #1 bit	Arg #2 bit	New Arg #2 bit
---------------	---------------	----------------------

0	0	0
0	1	0
1	0	0
1	1	1

8.3.1.16. CALL “CBL_CHANGE_DIR” USING *directory-path*

This routine makes *directory-path* (an alphanumeric literal or identifier) the current directory.

The directory change remains in effect until the program terminates (in which the original current directory at the time the program was restarted will be automatically restored) or until another CBL_CHANGE_DIR (or C\$CHDIR) is executed.

The return code of the operation is returned in the **RETURN-CODE** special register. The return code of the operation will be either 0=Success or 128=failure.

8.3.1.17. CALL “CBL_CHECK_FILE_EXIST” USING *file-path*, *file-info*

With this routine you may retrieve the size of the file⁴⁹ specified as the *file-path* argument (an alphanumeric literal or identifier) and the date/time that file was last modified. The information is returned to the *file-info* argument, which is defined as the following 16-byte area:

```

01 Argument-2.
   05 File-Size-In-Bytes PIC 9(18) COMP.
   05 Mod-DD             PIC 9(2)  COMP.  *> Modification Time
   05 Mod-MO             PIC 9(2)  COMP.
   05 Mod-YYYY           PIC 9(4)  COMP.  *> Modification Date
   05 Mod-HH             PIC 9(2)  COMP.
   05 Mod-MM             PIC 9(2)  COMP.
   05 Mod-SS             PIC 9(2)  COMP.
   05 FILLER             PIC 9(2)  COMP.  *> This will always be 00

```

If the subroutine is successful, a value of 0 will be returned in **RETURN-CODE**. Failure to retrieve the needed statistics on the file will cause a **RETURN-CODE** value of 35 to be passed back. Supplying less than two arguments will generate a 128 **RETURN-CODE** value.

8.3.1.18. CALL “CBL_CLOSE_FILE” USING *file-handle*

The CBL_CLOSE_FILE subroutine closes a bytestream file previously opened by either the **CBL_OPEN_FILE** or **CBL_CREATE_FILE** subroutines.

If the file defined by the *file-handle* argument (a PIC X(4) USAGE COMP-X data item) was opened for output, an implicit **CBL_FLUSH_FILE** will be performed before the file is closed.

If the subroutine is successful, a value of 0 will be returned in **RETURN-CODE**. Failure will cause a **RETURN-CODE** value of -1 to be passed back.

8.3.1.19. CALL “CBL_COPY_FILE” USING *src-file-path*, *dest-file-path*

Use this subroutine to copy file *src-file-path* to *dest-file-path* as if it were done via the “CP” (Unix) or “COPY” (Windows) command.

Both file path arguments may be alphanumeric literals or identifiers.

If the attempt to copy the file fails (for example, it or the destination directory doesn't exist), **RETURN-CODE** will be set to 128; on successful completion it will be set to 0.

⁴⁹ File size information may not be available in the particular GNU COBOL build / Operating System combination you are using and may therefore always be returned as zero.

8.3.1.20. CALL “CBL_CREATE_DIR” USING *dir-path*

With this routine you may create a new directory – the name of which is supplied as the *dir-path* argument (an alphanumeric literal or identifier).

Only the lowest-level directory (last) in the specified path can be created – all others must already exist. This subroutine will NOT behave as a “mkdir -p” (Unix) or “mkdir /p” (Windows).

RETURN-CODE will be set to the return code of the operation; the value will be either 0=Success or 128=failure.

8.3.1.21. CALL “CBL_CREATE_FILE” USING *file-path*, 2, 0, 0, *file-handle*

The CBL_CREATE_FILE subroutine creates the new file specified using the *file-path* argument and opens it for output as a byte-stream file usable by **CBL_WRITE_FILE**.

Arguments 2, 3 and 4 should be coded as the constant values shown.⁵⁰

A *file handle* (PIC X(4) USAGE COMP-X) will be returned, for any subsequent **CBL_WRITE_FILE** or **CBL_CLOSE_FILE** calls.

The success or failure of the subroutine will be reported back in the **RETURN-CODE** register, with a **RETURN-CODE** value of -1 indicating an invalid argument and a value of 0 indicating success.

8.3.1.22. CALL “CBL_DELETE_DIR” USING *dir-path*

Delete an empty directory via CBL_DELETE_DIR.

The only argument – *dir-path* (an alphanumeric literal or identifier) – is the name of the directory to be deleted.

Only the lowest-level directory (last) in the specified path will be deleted, and that directory must be empty to be deleted.

RETURN-CODE will be set to the return code of the operation; the value will be either 0=Success or 128=failure.

8.3.1.23. CALL “CBL_DELETE_FILE” USING *file-path*

This routine deletes the file specified by the *file-path* argument (an alphanumeric literal or identifier) just as if that were done using the “RM” (Unix) or “ERASE” (Windows) command.

If the attempt to delete the file fails (for example, it doesn't exist), **RETURN-CODE** will be set to 128; on successful completion it will be set to 0.

8.3.1.24. CALL “CBL_ERROR_PROC” USING *function*, *program-pointer*

This routine registers a general error-handling routine.

The *function* argument must be a numeric literal or a 32-bit binary **COMP-5** data item (**USAGE BINARY-LONG**, for example) with a value of 0 or 1. A value of 0 means that you will be registering (“installing”) an error procedure while a value of 1 indicates you’re deregistering (“uninstalling”) a previously-installed error procedure.

The *program-pointer* must be a **USAGE PROGRAM-POINTER** data item containing the address of your error procedure. This item should be given a value using the **SET program-pointer** statement. If the error procedure is written in GNU COBOL, it must be a subroutine, not a user-defined function.

A success (0) or failure (non-0) result will be passed back in the **RETURN-CODE** register.

A custom error procedure, will trigger when a runtime error condition is encountered. An error procedure may be registered by a main program or a subprogram, but regardless of from where it was registered, it applies to the overall program compilation group and will trigger when a runtime error occurs anywhere in the executable program. If the error procedure was defined by a subprogram, that program must be loaded at the time the error procedure is executed.

⁵⁰ CBL_CREATE_FILE is actually a special-case of the CBL_OPEN_FILE routine - see that routine for a description of the meanings of arguments 2, 3 and 4.

The code within the handler will be executed and – once the handler issues a return (C) or an **EXIT PROGRAM** or **GOBACK** (GNU COBOL), the system-standard error handling routine will be executed.

Only one user-defined error procedure may be in effect at any time.

The following is a sample GNU COBOL program that registers an error procedure. The output of that program is shown as well - as you can see, the error handler's messages appear followed by the standard GNU COBOL message.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DemoERRPROC.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Err-Proc-Address          USAGE PROGRAM-POINTER.
PROCEDURE DIVISION.
S1.
  DISPLAY 'Program is starting'
  SET Err-Proc-Address TO ENTRY 'ErrProc'
  CALL 'CBL_ERROR_PROC' USING 0, Err-Proc-Address
  CALL 'Tilt' *> THIS DOESN'T EXIST!!!!
  DISPLAY 'Program is stopping'
  STOP RUN
.
END PROGRAM DemoERRPROC.

IDENTIFICATION DIVISION.
PROGRAM-ID. ErrProc.
PROCEDURE DIVISION.
000-Main.
  DISPLAY 'Error: ' EXCEPTION-LOCATION
  DISPLAY ' ' EXCEPTION-STATEMENT
  DISPLAY ' ' EXCEPTION-FILE
  DISPLAY ' ' EXCEPTION-STATUS
  DISPLAY '*** Returning to Standard Error Routine ***'
  EXIT PROGRAM
.
END PROGRAM ErrProc.
```

When executed, this sample program generates the following console output:

```
>demoerrproc
Program is starting
Error: DemoERRPROC; S1; 13
CALL
00
EC-PROGRAM-NOT-FOUND
*** Returning to Standard Error Routine ***
DEMOERRPROC.cbl: 28: libcob: Cannot find module 'Tilt'
```

8.3.1.25. CALL “CBL_EXIT_PROC” USING *function, program-pointer*

This routine registers a general exit-handling routine.

The *function* argument must be a numeric literal or a 32-bit binary COMP-5 data item (USAGE BINARY-LONG, for example) with a value of 0 or 1. A value of 0 means that you will be registering (“installing”) an exit procedure while a value of 1 indicates you’re deregistering (“uninstalling”) a previously-installed exit procedure.

The *program-pointer* must be a **USAGE PROGRAM-POINTER** data item containing the address of your exit procedure.

A success (0) or failure (non-0) result will be passed back in the **RETURN-CODE** register.

An exit procedure will trigger when a “**STOP RUN**” or its equivalent (i.e. “**GOBACK**” executed in a main program) is executed. The exit procedure code will be executed and – once it issues an **EXIT PROGRAM** or a **GOBACK**, the system-standard program termination routine will be executed.

Only one user-defined exit procedure may be in effect at any time.

An exit procedure may be defined by a main program or a subprogram, but regardless of from where it was registered, it applies to the overall program compilation group and will trigger when a **STOP RUN** is executed anywhere in the executable program. If the exit procedure was defined by a subprogram, that program must be loaded at the time the exit procedure is executed.

An exit procedure should terminate using **EXIT PROGRAM** or a **GOBACK**.

The following is a sample GNU COBOL program that registers an exit procedure. The output of that program is shown as well.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. demoexitproc.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
78 Exit-Proc-Install          VALUE 0.
01 Current-Date              PIC X(8).
01 Current-Time              PIC X(8).
01 Exit-Proc-Address         USAGE PROCEDURE-POINTER.
01 Formatted-Date            PIC XXXX/XX/XX.
01 Formatted-Time            PIC XX/XX/XX.
PROCEDURE DIVISION.
000-Register-Exit-Proc.
  SET Exit-Proc-Address TO ENTRY "999-Exit"
  CALL "CBL_EXIT_PROC"
    USING Exit-Proc-Install, Exit-Proc-Address
  END-CALL
  IF RETURN-CODE NOT = 0
    DISPLAY 'Error: Could not register Exit Procedure'
  END-IF
.
099-Now-Test-Exit-Proc.
  DISPLAY
    'Executing a STOP RUN...'
  END-DISPLAY
  GOBACK
.
999-Exit-Proc.
  ENTRY "999-Exit"
  DISPLAY
    '*** STOP RUN has been executed ***'
  END-DISPLAY
  ACCEPT
    Current-Date FROM DATE YYYYMMDD
  END-ACCEPT
  ACCEPT
    Current-Time FROM TIME
  END-ACCEPT
  MOVE Current-Date TO Formatted-Date
  MOVE Current-Time TO Formatted-Time
  INSPECT Formatted-Time REPLACING ALL '/' BY ':'
  DISPLAY
    '***   ' Formatted-Date '   ' Formatted-Time '   ***'
  END-DISPLAY
  GOBACK
.

```

Program output...

```

Executing a STOP RUN...
*** STOP RUN has been executed ***
***   2009/08/28  10:01:29   ***

```

8.3.1.26. CALL “CBL_EQ” USING *item-1*, *item-2*, BY VALUE *byte-length*

This subroutine performs a bit-by-bit test for equality between the left-most $8 \times \textit{byte-length}$ corresponding bits of *item-1* and *item-2*, storing the resulting bit string into *item-2*.

Item-1 may be an alphanumeric literal or a data item. *Item-2* must be a data item. The length of both *item-1* and *item-2* must be at least $8 \times \textit{byte-length}$.

Byte-length may be a numeric literal or data item, and must be specified using

The truth table shown to the right documents the “EQ” process.

Arg #1 **Arg #2** **New**

Any bits in *item-2* after the $8 \times \text{byte-length}$ point will be unaffected.

A result of zero will be passed back in the **RETURN-CODE** register.

bit	bit	Arg #2 bit
0	0	1
0	1	0
1	0	0
1	1	1

8.3.1.27. CALL “CBL_FLUSH_FILE” USING *file-handle*

In Micro Focus COBOL, **CALL**ing this subroutine flushes any as-yet unwritten memory buffers for the (output) file whose *file-handle* is specified as the argument to disk.

This routine is non-functional in GNU COBOL. It exists only to provide compatibility for applications that may have been developed for Micro Focus COBOL.

8.3.1.28. CALL “CBL_GET_CURRENT_DIR” USING BY VALUE 0, BY VALUE *length*, BY REFERENCE *buffer*

This retrieves the fully-qualified pathname of the current directory, saving up to *length* characters of that name into the specified *buffer*.

The first argument is unused, but must be specified. It must be specified

The *length* argument must be specified

The *buffer* argument must be specified

The value specified for the *length* argument (a numeric literal or data item) should not exceed the actual length of the *buffer* argument.

If the value specified for the *length* argument is LESS THAN the actual length of the *buffer* argument, the current directory path will be left-justified and space filled within the first *length* bytes of *buffer* – any bytes in *buffer* after that point will be unchanged.

If the routine is successful, a value of 0 will be returned to the **RETURN-CODE** register. If the routine failed because of a problem with an argument (such as a negative or 0 *length*), a **RETURN-CODE** value of 128 will result. Finally, if the 1st argument value is anything but zero, the routine will fail with a 129 **RETURN-CODE**.

8.3.1.29. CALL “CBL_GET_CSR_POS” USING *cursor-locn-buffer*

This subroutine will retrieve the current cursor location on the screen, returning a 2-byte value into the supplied *cursor-locn-buffer*. The first byte of *cursor-locn-buffer* will receive the current line (row) location while the second receives the current column location.

The returned location data will be in exact binary (i.e. USAGE COMPUTATIONAL) form, and will be based upon starting values of 0, meaning that if the cursor is located at line 15, column 12 at the time this routine is called, a value of (14,11) will be returned.

The following is a typical *cursor-locn-buffer* definition:

```
01  CURSOR-LOCN-BUFFER.
    05  CURSOR-LINE           USAGE BINARY-CHAR.
    05  CURSOR-COLUMN        USAGE BINARY-CHAR.
```

Values of 1 (Line) and 1 (column) will be returned if GNU COBOL was not generated to include screen I/O.

8.3.1.30. CALL “CBL_GET_SCR_SIZE” USING *no-of-lines*, *no-of-cols*

Use this subroutine to retrieve the current console screen size. When the system is running in a windowed environment, this will be the sizing of the console window in which the program is executing. When the system is not running a windowing environment, the physical console screen attributes will be returned. In environments such as a Windows console window, where the logical size of the window may far exceed that of the physical console window, the size returned will be that of the physical console window. Two one-byte values will be returned – the first will be the current number of lines (rows) while the second will be the number of columns.

The returned size data will be in exact binary (i.e. USAGE COMPUTATIONAL) form.

The following are typical *no-of-lines* and *no-of-columns* Definitions

```
01 NO-OF-LINES          USAGE BINARY-CHAR.
01 NO-OF-COLUMNS      USAGE BINARY-CHAR.
```

GNU COBOL run-time screen management must have been initialized prior to **CALLING** this routine in order to receive meaningful values. This means that a screen-data DISPLAY and/or a screen-data ACCEPT must have been executed prior to the **CALL**.

Zero values will be returned if the screen has not been initialized and values of 24 (lines) and 80 (columns) will be returned if GNU COBOL was not generated to include screen I/O.. Compare this result with that of a screen-information ACCEPT.

See Also...

The ACCEPT Statement (Screen Data) [6.4.1.4](#)

The DISPLAY Statement (Screen Data) [6.4.12.4](#)

The **ACCEPT** Statement (Screen Info): [6.4.1.6](#)

8.3.1.31. CALL “CBL_IMP” USING *item-1*, *item-2*, BY VALUE *byte-length*

This subroutine performs a bit-by-bit “implies” test between the left-most $8 \times \text{byte-length}$ corresponding bits of *item-1* and *item-2*, storing the resulting bit string into *item-2*.

Item-1 may be an alphanumeric literal or a data item. *Item-2* must be a data item. The length of both *item-1* and *item-2* must be at least $8 \times \text{byte-length}$.

Byte-length may be a numeric literal or data item, and must be specified using

The truth table shown to the right documents the “IMP” process.

Any bits in *item-2* after the $8 \times \text{byte-length}$ point will be unaffected.

A result of zero will be passed back in the **RETURN-CODE** register.

Arg #1 bit	Arg #2 bit	New Arg #2 bit
0	0	1
0	1	1
1	0	0
1	1	1

8.3.1.32. CALL “CBL_NIMP” USING *item-1*, *item-2*, BY VALUE *byte-length*

This subroutine performs the negation of a bit-by-bit “implies” test between the left-most $8 \times \text{byte-length}$ corresponding bits of *item-1* and *item-2*, storing the resulting bit string into *item-2*.

Item-1 may be an alphanumeric literal or a data item. *Item-2* must be a data item. The length of both *item-1* and *item-2* must be at least $8 \times \text{byte-length}$.

Byte-length may be a numeric literal or data item, and must be specified using

The truth table shown to the right documents the “NIMP” process.

Any bits in *item-2* after the $8 \times \text{byte-length}$ point will be unaffected.

A result of zero will be passed back in the **RETURN-CODE** register.

Arg #1 bit	Arg #2 bit	New Arg #2 bit
0	0	0
0	1	0
1	0	1
1	1	0

8.3.1.33. CALL “CBL_NOR” USING *item-1*, *item-2*, BY VALUE *byte-length*

This subroutine performs the negation of a bit-by-bit “OR” test between the left-most $8 \times \text{byte-length}$ corresponding bits of *item-1* and *item-2*, storing the resulting bit string into *item-2*.

Item-1 may be an alphanumeric literal or a data item. *Item-2* must be a data item. The length of both *item-1* and *item-2* must be at least $8 \times \text{byte-length}$.

Byte-length may be a numeric literal or data item, and must be specified using

The truth table shown to the right documents the “NOR” process.

Any bits in *item-2* after the $8 \times \text{byte-length}$ point will be unaffected.

Arg #1 bit	Arg #2 bit	New Arg #2 bit
0	0	1
0	1	0
1	0	0
1	1	0

A result of zero will be passed back in the **RETURN-CODE** register.

		bit
0	0	1
0	1	0
1	0	0
1	1	0

8.3.1.34. CALL “CBL_NOT” USING *item-1*, BY VALUE *byte-length*

This subroutine “flips” the left-most 8**byte-length* bits of *item-2*, storing the resulting bit string into *item-2*.

Item-2 must be a data item. The length of *item-2* must be at least 8**byte-length*.

Byte-length may be a numeric literal or data item, and must be specified using .

The truth table shown to the right documents the “NOT” process.

Any bits in *item-2* after the 8**byte-length* point will be unaffected.

A result of zero will be passed back in the **RETURN-CODE** register.

Old Arg #2 bit	New Arg #2 bit
0	1
1	0

8.3.1.35. CALL “CBL_OC_NANOSLEEP” USING *nanoseconds-to-sleep*

CB_OC_NANOSLEEP puts the program to sleep for the specified number of nanoseconds.

The *nanoseconds-to-sleep* argument is a numeric literal or data item.

There are one BILLION nanoseconds in a second, so if you wanted to put the program to sleep for 1/4 second you'd use a *nanoseconds-to-sleep* value of 250000000.

8.3.1.36. CALL “CBL_OPEN_FILE” *file-path, access-mode, 0, 0, handle*

.This routine opens an existing file for use as a byte-stream file usable by **CBL_WRITE_FILE** or **CBL_READ_FILE**.

The *file-path* argument is an alphanumeric literal or data-item.

The *access-mode* argument is a numeric literal or data item with a PIC X USAGE COMP-X (or USAGE BINARY-CHAR) definition; it specifies how you wish to use the file, as follows:

1 = input (read-only)

2 = output (write-only)

3 = input and/or output

The third and fourth arguments would specify a locking mode and device specification, respectively, but they're not implemented in GNU COBOL (currently, at least) – just specify each as 0.

The final argument – *handle* - is a PIC X(4) USAGE COMP-X item that will receive the handle to the file. That handle is used on all other byte-stream functions to reference this specific file.

A **RETURN-CODE** value of -1 indicates an invalid argument, while a value of 0 indicates success. A value of 35 means the file does not exist.

8.3.1.37. CALL “CBL_OR” USING *item-1, item-2*, BY VALUE *byte-length*

This subroutine performs a bit-by-bit “OR” test between the left-most 8**byte-length* corresponding bits of *item-1* and *item-2*, storing the resulting bit string into *item-2*.

Item-1 may be an alphanumeric literal or a data item. *Item-2* must be a data item. The length of both *item-1* and *item-2* must be at least 8**byte-length*.

Byte-length may be a numeric literal or data item, and must be specified using .

The truth table shown below documents the “OR” process.

Any bits in *item-2* after the 8**byte-length* point will be unaffected.

Arg #1 bit	Arg #2 bit	New Arg #2
0	0	0
0	1	1
1	0	1
1	1	1

A result of zero will be passed back in the **RETURN-CODE** register.

		bit
0	0	0
0	1	1
1	0	1
1	1	1

8.3.1.38. CALL “CBL_READ_FILE” USING *handle*, *offset*, *nbytes*, *flag*, *buffer*

This routine reads *nbytes* of data starting at byte number *offset* from the byte-stream file defined by *handle* into the specified *buffer*.

The *handle* argument (PIC X(4) USAGE COMP-X) must have been populated by a prior call to CBL_OPEN_FILE.

The *offset* argument (PIC X(8) USAGE COMP-X) defines the location in the file of the first byte to be read. The first byte of a file is byte offset 0.

The *nbytes* argument (PIC X(4) USAGE COMP-X) specifies how many bytes (maximum) will be read.

If the *flags* argument is specified as 128, the size of the file (in bytes) will be returned into the file offset argument (argument 2) upon completion.⁵¹ The only other valid value for *flags* is 0. This argument may be specified either as a numeric literal or as a PIC X USAGE COMP-X data item.

Upon completion, **RETURN-CODE** will be set to 0 if the read was successful or to 10 if an “end-of-file” condition occurred. If **RETURN-CODE** has a value of -1, a problem was identified with the subroutine arguments.

8.3.1.39. CALL “CBL_RENAME_FILE” USING *old-file-path*, *new-file-path*

You may use this subroutine to rename a file.

The file specified by *old-file-path* will be “renamed” to the name specified as *new-file-path*. Each argument may be an alphanumeric literal or data item.

Despite what the name of this routine might make you believe, this routine is more than just a simple “rename” – it will actually move the file supplied as the 1st argument to the file specified as the 2nd argument. Think of it as a two-step sequence, first copying the *old-file-path* to the *new-file-path* and then a second step where the *old-file-path* is deleted.

If the attempt to move the file fails (for example, it doesn't exist), **RETURN-CODE** will be set to 128; on successful completion it will be set to 0.

8.3.1.40. CALL “CBL_TOLOWER” USING *data-item*, BY VALUE *convert-length*

This routine will convert *convert-length* (a numeric literal or data item) leading characters of *data-item* (an alphanumeric identifier) to lower-case.

The *convert-length* argument must be specified . It specifies how many (leading) characters in *data-item* will be converted – any characters after that will remain unchanged.

If *convert-length* is negative or zero, no conversion will be performed.

8.3.1.41. CALL “CBL_TOUPPER” USING *data-item*, BY VALUE *length*

Use C\$TOUPPER to change the *convert-length* (a numeric literal or data item) leading characters of *data-item* (an alphanumeric identifier) to upper-case.

The *convert-length* argument must be specified . It specifies how many (leading) characters in *data-item* will be converted – any characters after that will remain unchanged.

If *convert-length* is negative or zero, no conversion will be performed.

⁵¹ Not all operating system/GNU COBOL environments may be able to retrieve file sizes – in such cases, a value of zero will be returned.

8.3.1.42. CALL “CBL_WRITE_FILE” USING *handle*, *offset*, *nbytes*, 0, *buffer*

This routine writes *nbytes* of data from the specified *buffer* to the byte-stream file defined by *handle* starting at byte number *offset*.

The *handle* argument (PIC X(4) USAGE COMP-X) must have been populated by a prior call to CBL_OPEN_FILE.

The *offset* argument (PIC X(8) USAGE COMP-X) defines the location in the file of the first byte to be written to. The first byte of a file is byte offset 0.

The *nbytes* argument (PIC X(4) USAGE COMP-X) specifies how many bytes (maximum) will be written.

The only allowable value for the *flags* argument is 0. This argument may be specified either as a numeric literal or as a PIC X USAGE COMP-X data item.

Upon completion, **RETURN-CODE** will be set to 0 if the write was successful or to 30 if an I/O error condition occurred. If **RETURN-CODE** has a value of -1, a problem was identified with the subroutine arguments.

8.3.1.43. CALL “CBL_XOR” USING *item-1*, *item-2*, BY VALUE *byte-length*

This subroutine performs a bit-by-bit exclusive “OR” test between the left-most $8 * \textit{byte-length}$ corresponding bits of *item-1* and *item-2*, storing the resulting bit string into *item-2*.

Item-1 may be an alphanumeric literal or a data item. *Item-2* must be a data item. The length of both *item-1* and *item-2* must be at least $8 * \textit{byte-length}$.

Byte-length may be a numeric literal or data item, and must be specified using

The truth table shown to the right documents the “XOR” process.

Any bits in *item-2* after the $8 * \textit{byte-length}$ point will be unaffected.

A result of zero will be passed back in the **RETURN-CODE** register.

Arg #1 bit	Arg #2 bit	New Arg #2 bit
0	0	0
0	1	1
1	0	1
1	1	0

8.3.1.44. CALL “SYSTEM” USING *command*

This subroutine submits the specified *command* (an alphanumeric literal or data item) to a command shell.

A shell will be opened subordinate to the GNU COBOL program issuing the **CALL** to SYSTEM.

Output from the command (if any) will appear in the command window in which the GNU COBOL program was executed.

On a Unix system, the shell environment will be established using the default shell program. This is also true when using a GNU COBOL build created with and for the Cygwin Unix emulator.

With native Windows Windows/MinGW builds, the shell environment will be the Windows console window command processor (usually “cmd.exe”) appropriate for the version of Windows you’re using.

To trap output from the executed command and process it within the GNU COBOL program, use a pipe (>) to send the command output to a temporary file which you then READ from within the program once control returns.

8.3.2. “Call by Number” Subroutines

Early versions of Micro Focus COBOL allowed programmers to access various runtime library routines by using a single two-digit hexadecimal number as the entry-point name. These were known as call-by-number routines. Over time, Micro Focus COBOL evolved, replacing most of the call-by-number routines with ones accessible using a more conventional call-by-name technique.

Most of the call-by-number routines have evolved into even more powerful call-by-name routines, many of which are supported by GNU COBOL and were already presented in section 8.3

Three of the original call-by-number routines never evolved call-by-name equivalents; GNU COBOL supports these routines.

8.3.2.1. CALL X”91” USING *return-code*, *function-code*, *binary-variable-*

arg

The original Micro Focus version of this routine is capable of providing a wide variety of functions – GNU COBOL supports just three of those Functions

- ▶ Turning runtime switches (SWITCH-1, ... , SWITCH-8) on
- ▶ Turning runtime switches (SWITCH-1, ... , SWITCH-8) off
- ▶ Retrieving the number of arguments passed to a subroutine⁵²

The *return-code* argument must be a binary numeric data item (USAGE BINARY-CHAR is recommended). It will receive a value of 0 if the operation was successful, 1 otherwise.

The *function code* argument must be either a numeric literal or a binary numeric data item (USAGE BINARY-CHAR is recommended).

The third argument – *variable-arg* – is defined differently depending upon the function-code value, as follows:

Value of function-code	Action To Be Performed	Definition and usage of variable-arg
11	Sets and/or clears <u>all eight</u> of the COBOL switches (SWITCH-1 through SWITCH-8) that are available for definition within SPECIAL-NAMES (see section 4.1.4) ⁵³	<i>Variable-arg</i> should be an OCCURS 8 TIMES array of USAGE BINARY-CHAR. Each occurrence that is set to a value of zero prior to the CALL will cause the corresponding switch to be cleared. Each occurrence set to 1 prior to the CALL will cause the corresponding switch to be set. Values other than 0 or 1 will be ignored.
12	Reads <u>all eight</u> of the COBOL switches (SWITCH-1 through SWITCH-8) that are available for definition within SPECIAL-NAMES (see section 4.1.4)	This argument should be an OCCURS 8 TIMES array of USAGE BINARY-CHAR. Each of the 1 st eight occurrences of the array will be set to either 0 or 1 – 1 if the corresponding switch is set, 0 otherwise.
16	Retrieves the number of arguments passed to the program executing the CALL X"91"	This argument should be a binary numeric data item (USAGE BINARY-CHAR is recommended). The number of arguments passed to the subroutine executing the CALL X"91" will be stored here.

8.3.2.2. CALL X"E4"

Use X"E4" to clear the screen. There are no arguments and no returned value.

8.3.2.3. CALL X"E5"

The X"E5" routine will sound the PC "bell". There are no arguments and no returned value.

8.3.2.4. CALL X"F4" USING *byte*, *table*

The Routine X"F4" packs an 8-byte area containing 8 1-byte binary values of 0 or 1 into the corresponding bit positions of a 1-byte data item.

The *byte* data item need be only a single byte in size. If it is longer, the excess will be unaffected by this subroutine.

Table must be a data item at least 8 bytes long. If it is longer, the excess will be ignored by this subroutine. Typically, *table* is defined similarly to the following:

```

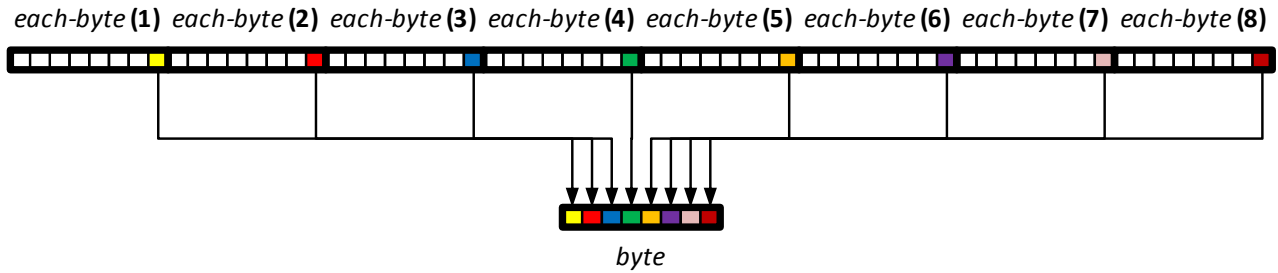
01 table.
05 each-byte OCCURS 8 TIMES USAGE BINARY-CHAR.

```

⁵² GNU COBOL actually has two other ways to accomplish this task – the **C\$NARG** subroutine and the **NUMBER-OF-CALL-PARAMETERS** special register; I recommend you use one of these methods instead of the X"91" routine when coding new programs

⁵³ If you only wish to set and/or clear some of the switches, it is recommended that you first use function 12 to read the current values of the switches and then change the *variable-arg* occurrences for the switch(es) you wish to change before using function 11 to actually make the changes.

The following diagram illustrates how this subroutine works.



The colored squares represent the bits in the 1st 8 bytes of *array* that will be packed into *byte*. The white squares represent the bits in each *each-byte* that will be ignored.

8.3.2.5. CALL X"F5" USING *byte, table*

This routine unpacks each bit of a byte into an 8-byte area so they may be individually accessed and manipulated.

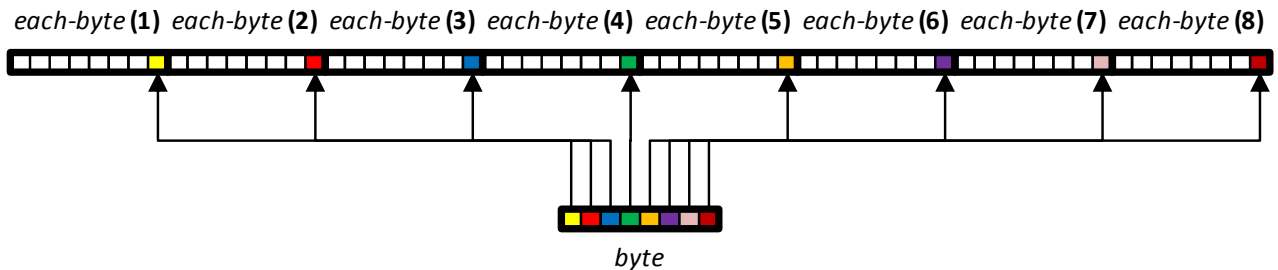
The *byte* data item need be only a single byte in size. If it is longer, the excess will be ignored by this subroutine.

Table must be a data item at least 8 bytes long. If it is longer, the excess will be unaffected by this subroutine.

Typically, *table* is defined similarly to the following:

```
01 table.
05 each-byte OCCURS 8 TIMES USAGE BINARY-CHAR.
```

The following diagram illustrates how this subroutine works.



The colored squares represent each of the 8 bits in *byte*. The diagram shows how those bits will be “unpacked” into the rightmost bit of each of the 1st 8 consecutive bytes of *array*. The white squares represent the remaining bits in each of the 1st 8 *each-byte* occurrences – all of which will be set to 0.

8.4. Binary Truncation

By default, the GNU COBOL compiler will truncate binary data items to the precision indicated by their PICTURE clause. For example, the following data item will have 2 bytes of storage allocated for it:

```
01 Comp-5-Item PIC 9(3) COMP-5.
```

Because of truncation, even though this field has enough bits allocated (16) to store values from 0 to 65535, it will be limited to values of 0 to 999 because of its PICTURE.

Or is it?

Take a look at the small demo program shown here. This program will perform three different types of operations against a binary field, displaying the results of each.

Here are the results when the program is compiled (with truncation in-effect by default) and executed:

```
Bin-Item-1=760 Disp-Item-1=032760
Bin-Item-1=765 Disp-Item-1=032765
Bin-Item-1=767 Disp-Item-1=032767
```

You can see that truncation affected the DISPLAY statements but appears to have had no impact whatsoever on the MOVE and ADD statements. This is the hidden secret about truncation in GNU COBOL: it doesn't really truncate the internally-stored values – it just truncates the DISPLAY of them!

If that same program is recompiled without truncation (by adding the “-fnotrunc” switch to the ‘cobc’ command), the results are as follows:

```
Bin-Item-1=32760 Disp-Item-1=032760
Bin-Item-1=32765 Disp-Item-1=032765
Bin-Item-1=32767 Disp-Item-1=032767
```

against all types of numeric data items (even USAGE DISPLAY) are slowed down.

Before continuing, it's worth making the point that we're NOT talking about astronomical performance degradations here. Today's computers are FAST, and a user sitting at the keyboard, running a GNU COBOL program is unlikely to notice. BUT ... if you have a GNU COBOL program that has to process large amounts of data, performing some significant “number crunching” against that data as it goes, the impact of truncation could become noticeable.

The demo program shown in [Figure 8-4](#) compares the performance of performing arithmetic operations (in a totally non-scientific, non-rigorous way) against USAGE DISPLAY, COMP, COMP-5 and BINARY-xxx⁵⁴ numeric data. It was actually my intent when I first wrote the program to merely demonstrate the relative performance differences between the first three types of numeric data storage, and it certainly met that objective.

Imagine my surprise, however, when I discovered that the use of “-fnotrunc” also made a significant difference!

Here's what the program does:

Figure 8-3 - A Binary Truncation Demo Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DEMOTRUNC.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Bin-Item-1 PIC 9(3)
COMP-5
VALUE 32760.
01 Disp-Item-1 PIC 9(6).
PROCEDURE DIVISION.
000-Main.
MOVE Bin-Item-1 TO Disp-Item-1
DISPLAY
'Bin-Item-1=' Bin-Item-1
'Disp-Item-1=' Disp-Item-1
END-DISPLAY
ADD 5 TO Bin-Item-1
MOVE Bin-Item-1 TO Disp-Item-1
DISPLAY
'Bin-Item-1=' Bin-Item-1
'Disp-Item-1=' Disp-Item-1
END-DISPLAY
MOVE 32767 TO Bin-Item-1
MOVE Bin-Item-1 TO Disp-Item-1
DISPLAY
'Bin-Item-1=' Bin-Item-1
'Disp-Item-1=' Disp-Item-1
END-DISPLAY
STOP RUN.
```

If this was all there was to the binary truncation issue it wouldn't be worth a section in this document. The fact is, however, that binary truncation has a significant effect on the performance of GNU COBOL programs. When binary truncation is in effect, arithmetic operations performed

⁵⁴ USAGE BINARY-xxx is supposed to store numeric data identically to USAGE COMP-5, but I felt it couldn't hurt to check.

- ▶ There are four numeric data items in the program – one USAGE DISPLAY, one USAGE COMP, one USAGE COMP-5 and one USAGE BINARY-LONG. Since the program was run on a computer with an Intel-architecture processor (actually it's an AMD, but results are identical with Intel) I wanted to see just how much more efficient COMP-5 was over COMP.
- ▶ Each data item will have 7 added to it ten million times. You'll see why shortly.
- ▶ The time (to one-one-hundredth of a second) will be retrieved before and after each test and the difference between the two will be DISPLAYed. This is why the computations were done so many times – it was to make sure the timing was “measurable” with only a 1/100 second “stopwatch”.

GNU COBOL is retrieving wall-clock time, not actual CPU-used time, so other activities taking place on the computer had to be kept to a minimum while the tests were running. I also ran the tests multiple times, just to make sure I had consistent results (I did). Like I mentioned earlier – this is not a rigorous, scientific benchmark of numeric performance; it's just a quick-and-dirty comparison.

[Figure 8-4](#) shows the program and the test results received when executing both with and without the “-fnotrunc” switch.

Here are the conclusions I drew from running these tests many times (30). The timings shown are average times from all Tests

With truncation ON:

- ▶ USAGE COMP has a significant performance advantage over USAGE DISPLAY
- ▶ USAGE COMP-5 has an even greater performance advantage over USAGE COMP, than COMP did over DISPLAY
- ▶ USAGE BINARY-LONG (and presumably the other BINARY-xxx USAGES as well) perform identically (within the measurement tolerances of the test) with COMP-5; this should be no surprise since COMP-5 and BINARY-xxx both allocate data the same way

With truncation OFF:

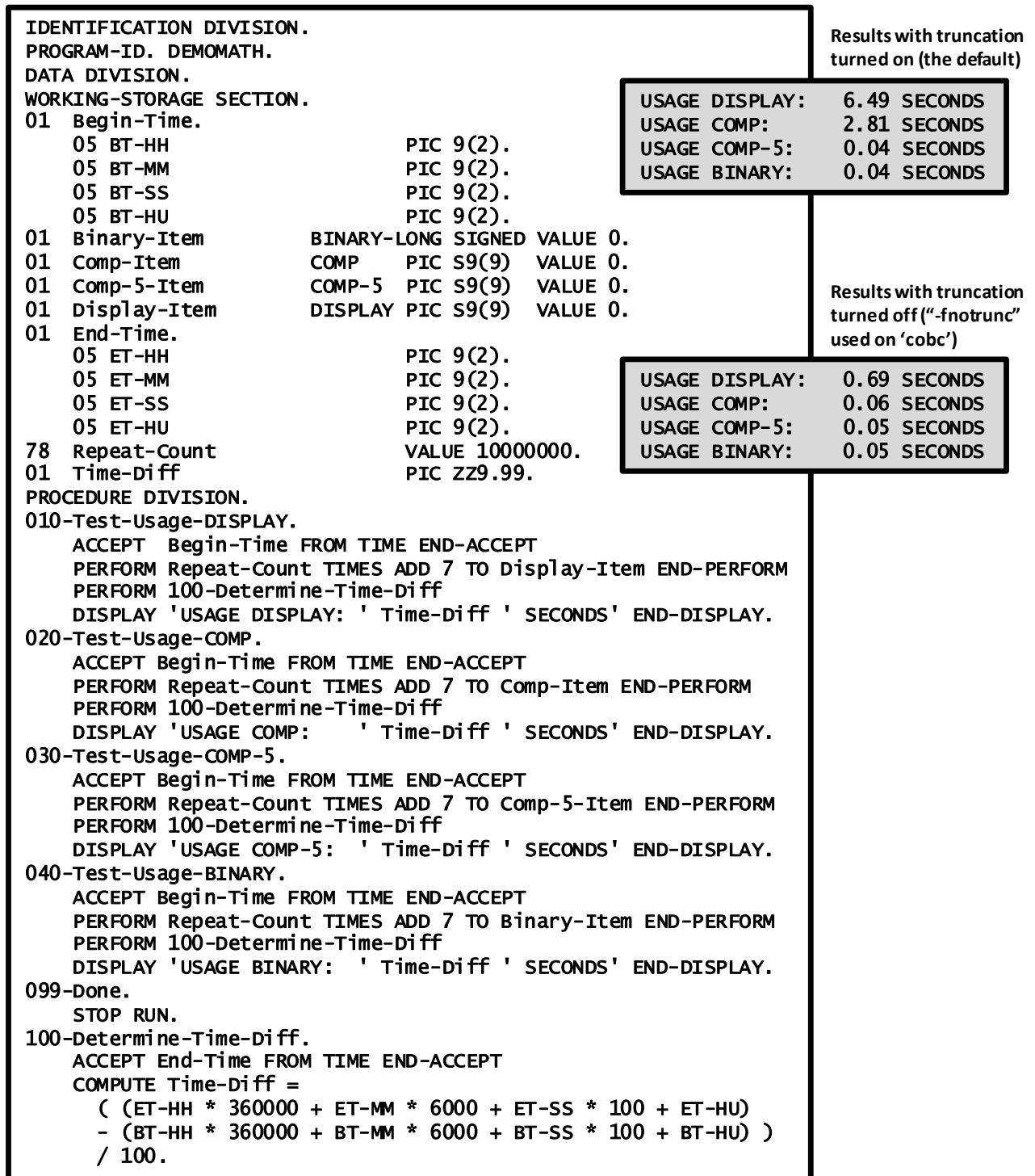
- ▶ There was a huge drop in both USAGE DISPLAY and USAGE COMP timings.
- ▶ The relative performance advantage of USAGE COMP over USAGE DISPLAY is even larger with truncation off than it was with it on.
- ▶ USAGE COMP-5 and USAGE BINARY-xxx appear to be virtually unaffected by the truncation on/off status, although there was a .01 second increase in average execution time of those tests without truncation over those with truncation. Given the number of times I ran the tests, it's obvious that something makes COMP-5/BINARY-xxx run slower without truncation than with it; that difference, however, is so miniscule that I discount it as being statistically irrelevant⁵⁵.

My final observation is that I see absolutely no reason whatsoever why the “-fnotrunc” option shouldn't be used on all GNU COBOL compilations.

If you want to squeeze every last bit of performance out of your GNU COBOL programs, don't forget to investigate the various “-O” (optimization) switches. Actually run programs using various optimization switches (or not) and compare execution times, don't just compare the generated C code because sometimes the differences can't be “seen” at the C source-code level.

⁵⁵ Remember – that's a .01 second difference over TEN MILLION iterations!

Figure 8-4 - A Non-Scientific Comparison of Numeric Data Item USAGE Performance



9. So, You're a New COBOL Programmer?

This chapter deals with a variety of stylistic issues that may be of interest to someone who is just starting out learning and using COBOL. Much of this chapter makes stylistic recommendations and suggestions for how to write your own programs. The sample programs in chapter [10](#) ("Sample Programs") were coded using almost all of these recommendations.

There's no particular order of importance to the topics presented here.

9.1. Marking Changes in Programs

For quite a while now (back to the 1980s), the "sequence number area" of a COBOL statement (columns 1-6) has come to be used as a change indicator area. Programmers would place a code in columns 1-6 of every line they changed in a program. The author works in a COBOL shop where change indicators of the form "xxmmyy" are required on every altered line of a program – "xx" is the initials of the programmer while "mmyy" are the month and two-digit year of the date the change was made. This is frequently accompanied by a comment block at or near the top of a COBOL program providing general documentation of what changes were made and what change indicator was used to mark that change.

The GCic sample program source listing provides an excellent example of such documentation.

This technique of using columns 1-6 as a change indicator will ONLY work if fixed source-record format is in effect.

Marking changes becomes more of a challenge when free-format source code is in effect. Creating a top-of-program comment block to generically describe changes that have been made isn't difficult, even in free-form. What IS difficult, however, is coming up with a scheme for per-statement markup of changes that doesn't introduce a ridiculously excessive number of source lines to the program. I'm not sure there is a good answer to this problem (if a reader has one, please let me know). Generally, I've noticed that shops using free-format conventions for their COBOL source tend to stick with just the top-of-program comment block combined with minimal comment blocks sprinkled throughout the program noting areas that underwent major changes.

See Also...

Fixed-Format Source Code [1.5.1.1](#)

Sample Programs: GCic [10.4](#)

9.2. Data Item Coding and Naming Conventions

When programs get very large, it becomes more and more challenging to keep track of the data items that will be used in the program. Here are, in no particular order of importance, are a variety of conventions that can simplify that problem.

Remember that the points described here are intended to make things easier for you – the programmer. No COBOL compiler cares one way or another whether any of these suggestions are followed.

1. Avoid the use of level 77 data items in new programs. Once (1968 and before) there were valid reasons for creating level-77 data items, but since the 1974 ANSI standard of COBOL there really hasn't been any reason why an elementary level-01 data item couldn't have been used instead of a level-77 item.
2. Allocate level-01 data items in alphabetical sequence in the program source wherever practical. This will make it vastly easier to locate the definition of an 01-level item in the program source.
3. Consider prefixing data items with an indication of where in the program structure they were created. For example:
 - Everything defined in the **FILE SECTION** starts with "F-"
 - Everything defined in **WORKING-STORAGE** starts with "WS-"
 - Everything defined in **LOCAL-STORAGE** starts with "LS-"
 - Everything defined in the **LINKAGE SECTION** starts with "L-"
 - Everything defined in the **SCREEN SECTION** starts with "S-"

A convention such as this makes it simple, when you're reviewing code in the **PROCEDURE DIVISION**, to know what section of the **DATA DIVISION** to look in to locate the detailed description of a data item.

4. Consider including an acronym to be inserted into the name of any data item defined directly or indirectly subordinate to an 01-level item, typically to be specified after any section-level tag, if you're using them, as discussed in item #3 above. For example, consider the names used in the following structure:

```

01 WS-FILE-STATUS-MESSAGE.
   05 FILLER                PIC X(13) VALUE 'Status Code: '.
   05 WS-FSM-Status-CD     PIC 9(2).
   05 FILLER                PIC X(11) VALUE ', Meaning: '.
   05 WS-FSM-Msg-TXT      PIC X(25).

```

The “-FSM-” acronyms make it easier to locate – in the program source code - the description of the 01-item the status code and message text items belong to.

5. Consider including a trailing descriptor of the nature of all data items in their names. Two examples of this – “-CD” and “-TXT” were included in the above example. The following chart presents a variety of such descriptors the author has encountered and used through the years:

Descriptor	Usage
-ADDR	The data item contains all or a part of an Address (City-ADDR, State-ADDR, Street-ADDR, ...)
-BOOL	A level-88 data item (which only has the value TRUE or FALSE)
-CD	A CODE whose value denotes information content above and beyond that of the mere value itself. Some examples could be “Error-CD”, “Status-CD”, “Billing-CD”
-CHR	A data item containing a single character of data.
-CONST	A constant, specified as a level-78 data item, a level-01 item with the CONST attribute
-DT	The data item contains a complete or partial date (Birth-DT, Birth-Month-DT, Birth-Year-DT, ...)
-DTTM	A data item containing both a date and a time
-FILE	A file name. Note that these items would probably also have a “F-” prefix.
-IDX	A data item used as a table index (see section 9.3)
-NM	All or a portion of a person's name. These could be extended to include business names, product names, etc.
-PTR	A data item whose USAGE is POINTER
-NUM	A generic numeric data item that doesn't fit into any of the other categories
-QTY	A count of something
-REC	An 01-level item defined in the FILE SECTION (constituting the layout of a record within a file). Note that these items would probably also have a “F-” prefix.
-SCR	The data item contains a complete or partial screen description (appropriate for SCREEN SECTION 01-level data items).
-SUB	A numeric item used as a table subscript (see section 9.3)
-TEL	All or part of a telephone number
-TM	The data item contains a complete or partial time value
-TXT	The data item contains generic alphanumeric text that doesn't fit into any of the other categories.

The above is by no means an exhaustive list, but good programmers will use as few of these descriptors as possible as having too many defeats any benefits of such classification/documentation efforts.

9.3. Table Subscripting versus Table Indexing

The elements of a table may be referenced either using a subscript or an index. Syntactically, this is coded using parenthesis, as per the following three examples, all of which store the letter “A” into the 17th occurrence of a data item named WSS- Output-Image-TXT:

1. MOVE 'A' TO WSS-Output-Image-TXT (17)
2. MOVE 17 TO WSS-OI-SUB
MOVE 'A' TO WSS-Output-Image-TXT (WSS-OI-SUB)
3. SET WSS-OI-IDX TO 17
MOVE 'A' TO WSS-Output-Image-TXT (WSS-OI-IDX)

Examples 1 and 2 are referred to as subscripting while example 3 is known as indexing. The distinction is fairly simple – INDEXING is the process of referencing an element of a table utilizing a data item with an explicitly or implicitly defined USAGE of INDEX to select the desired occurrence, while SUBSCRIPTING is the process of referencing an element of a table utilizing either a numeric constant or an unedited numeric data item to select the desired occurrence.

Various implementations of COBOL generate object code that is quite different in each of these three situations, and GNU COBOL is no exception. In general, table references such as example #1 (constant subscript) generate the smallest, simplest and fastest object code while table references such as example #2 (numeric data item subscript) generate the largest, most-complicated and slowest object code. Table references such as example #3 (table indexing) generate object code that falls in the middle of the other two but is far closer in efficiency to example #1 than #2.

Some COBOL statements (SEARCH, SEARCH ALL and table-based SORT) require you to index the affected table and to utilize that index with those statements. With any other references to tables, the choice is left to the programmer as to which approach should be used. In general, follow these rules:

1. Use constant subscripts (example #1) wherever possible/practical.
2. If references to table elements are going to be performed many, many times (tens or hundreds of thousands of times or more) during program execution, you will probably see a noticeable improvement in program execution time if you use indexing versus subscripting.

Since it's impossible to perform any arithmetic operation against an index data item directly (other than a simple incrementation or decrementation operation), situations where any non-trivial computations are required to calculate the effective occurrence number for a table reference will require you to use a numeric data item to serve as the receiving field for the calculation. That calculated value would then need to be saved into the index data item via a SET statement.

If you only need to use the computed occurrence number once, you might as well just use the computed occurrence number data item as a subscript. If, however, you will need to use a computed "subscript" more than once, the run-time overhead of converting that occurrence value to an index (via SET) will be worth the coding effort.

Whew!

3. If references to table elements are not going to be performed many, many times it probably won't make much difference whether you use indexing or subscripting.

If you are comfortable with the "C" programming language, you might find the following simple GNU COBOL program useful in exploring the differences between subscripting and indexing:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBVINDEX.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-TABLE-SUB                BINARY-LONG.
01 WS-TABLE.
   05 WS-TABLE-ENTRY          OCCURS 20 TIMES
                               INDEXED BY WS-TABLE-IDX
                               PIC X(1).

PROCEDURE DIVISION.
000-Main SECTION.
E1. MOVE 'A' TO WS-TABLE-ENTRY (17)
.
E2. MOVE 17 TO WS-TABLE-SUB
   MOVE 'A' TO WS-TABLE-ENTRY (WS-TABLE-SUB)
.
E3. SET WS-TABLE-IDX TO 17
   MOVE 'A' TO WS-TABLE-ENTRY (WS-TABLE-SUB)
.
```

Compile this program as follows (the assumption is made that you are executing the cobc command from the directory in which the above program source code (**subvsindex.cbl**) exists).

```
cobc -C -save-temps subvsindex.cbl
```

After this command is executed, the files "**subvsindex.c**" will contain the "PROCEDURE DIVISION" C code and "**subvsindex.c.1.h**" will contain the "WORKING-STORAGE" C code.

See Also...

Giving a Data Item a Compile-Time VALUE	5.2.1.12	The SEARCH ALL Statement	6.4.38.2
Referencing Table Entries	6.1.1	The SORT Statement (Table Sort)	6.4.40.2
The SEARCH Statement	6.4.38.1	The SET (Index) Statement	6.2.39.4
The SET UP/DOWN Statement	6.2.39.5		

9.4. Copybook Naming Conventions and Usage

Since the intent of a copybook is to introduce COBOL code into a particular spot in a program via the COPY statement, it is always a good idea to prefix copybook names with a two-character sequence that identifies where in a program it's contents are intended to be COPYed.

For example:

- IDxxxxxxx** copybooks containing code intended for the **IDENTIFICATION DIVISION**. These will be rare as you almost never encounter COPYed code in the **IDENTIFICATION DIVISION**.
- EDxxxxxxx** copybooks containing code intended for use in the **ENVIRONMENT DIVISION**. These copybooks are generally used for predefined **SPECIAL-NAMES** or **FILE-CONTROL** syntax,
- DDxxxxxxx** copybooks that contain data definitions.
- PDxxxxxxx** copybooks that contain executable instructions.

9.5. PROCEDURE DIVISION Sections Versus Paragraphs

The issue of whether to use section and/or paragraph names (collectively referred to as *procedure names*) within the PROCEDURE DIVISION is one approaching religious significance with many COBOL programmers.

COBOL programming standards used by many organizations that use the language generally call for procedure names to:

1. Contain a leading numeric component
2. Be defined in the PROCEDURE DIVISION in non-decreasing sequence of that numeric component.

When you are looking at or editing any large COBOL program that has been created with programming standards that include these two rules, it is always a simple thing to know whether a reference to a procedure is being made to code that exists before or after your current location in the program!

Technically, GNU COBOL does not require ANY procedure names be defined unless:

1. You are using the **ALTER** statement (the use of which should be avoided at all costs)
2. You are using Format 1 of the **PERFORM** Statement
3. You are using a **GO TO** Statement
4. You are using a SORT or MERGE statement with either (or both) an INPUT PROCEDURE or OUTPUT PROCEDURE

Since it is difficult to write any non-trivial COBOL program that uses none of the above, lets assume you will be including at least one section or paragraph in your GNU COBOL programs.

I like to use PROCEDURE DIVISION paragraphs and sections as follows.

1. The very first procedure defined in the PROCEDURE DIVISION of my programs, assuming no DECLARATIVES are defined, will be a SECTION named "**000-Main**". The declaration of this procedure will immediately follow the PROCEDURE DIVISION header (or **END DECLARATIVES** if DECLARATIVES are used).
2. Any procedures referenced by **MERGE**, **PERFORM**, or **SORT** statements will be defined as their own sections.
3. Any procedures referenced by **GO TO** statements will be defined as paragraphs, and those paragraphs will be defined in the same section as the **GO TO** statements that reference them. In other words, GO TO

statements may not be used to transfer control to a point in a different section. This is NOT a GNU COBOL rule – this is my own personal rule intended to improve the readability of my programs.

4. I always include a numeric prefix to all procedure names I define, and those numbers are assigned in non-decreasing sequence of their value. Thus it is always possible, provided you know in what procedure the GO TO, MERGE, PERFORM or SORT statement you are looking at is located, to know whether you should look forward or backward in the program to find the procedure the statement is referencing.
5. I do not use THRU on any **MERGE**, **PERFORM** or **SORT** statement unless the programming standards of the shop in which I am working requires it.. My reasoning for this is that it is too easy to accidentally introduce a new procedure into the scope of a PERFORM.

See Also...

The USE Statement and DECLARATIVES	6.1.4	The GO TO Statement	6.2.20
The ALTER Statement	6.2.4	The MERGE Statement	6.2.25
PERFORM Format 1 - Procedural	6.2.30.1	SORT Format 1 – File-based SORT	6.2.40.1

9.6. COMPUTE Versus ADD, SUBTRACT, MULTIPLY and DIVIDE

Over the years, there has been much debate over the effectiveness, appropriateness and arithmetic accuracy of using the **COMPUTE** statement rather than the four basic arithmetic operation statements (**ADD**, **SUBTRACT**, **MULTIPLY**, **DIVIDE**).

Here are the facts. Draw your own conclusions as to which approach is more appropriate under which circumstances.

1. The **COMPUTE** statement supports exponentiation (via the “**” operator) – there is no equivalent basic arithmetic statement. Although you could simulate integral exponentiation (raising a value to the third power, for example) using **MULTIPLY** statements, and you may use the **SQRT** built-in intrinsic function to find a square root, there’s just no (easy) way to find the ¼ root of a value without using **COMPUTE**.
2. For non-trivial computations, **COMPUTE** statements “read” better. Take this, for example:

```
COMPUTE R = (A + B * C) / D
```

As compared to:

```
MULTIPLY B BY C GIVING TEMP
ADD A TO TEMP
DIVIDE TEMP BY D GIVING R
```

3. For non-trivial computations, **COMPUTE** statements may execute faster than the equivalent chain of basic arithmetic statements. For example, the **COMPUTE** statement shown in #2 above executes about 25% faster on than does the **MULTIPLY-ADD-DIVIDE** sequence.
4. For trivial computations, on the other hand, I prefer the inherent readability of a statement such as this:

```
ADD 1 TO WSS-Input-Trans-QTY
```

to this:

```
COMPUTE WSS-Input-Trans-QTY = WSS-Input-Trans-QTY + 1
```

See Also...

The ADD Statement	6.2.2	The MULTIPLY Statement	6.2.27
The COMPUTE Statement	6.2.9	The SUBTRACT Statement	6.2.44
The DIVIDE Statement	6.2.13		

10. Sample Programs

This chapter contains some sample GNU COBOL programs, subroutines, functions and copybooks. All code shown here is included in release-appropriate form within the “**samples**” directory of GNU COBOL distributions that I prepare. They are also available upon request using the email address on the cover.

All program listings were created by the **GCic GNU COBOL Interactive Compiler** (itself a sample program listed in section [10.4](#)).

10.1. FileStat-Msgs.cpy – File Status Values

The **FileStat-Msgs.cpy** copybook contains an **EVALUATE** statement to translate the two-digit file status codes that may be generated by file I/O statements.

The copybook assumes that the file status data item name is “**STATUS**” and the error message data item is named “**MSG**”. By using the **COPY** statement’s **REPLACING** clause, however, you may use the data names you wish, as follows:

```
COPY FileStat-Msgs
  REPLACING STATUS BY file-status-data-item-name
  MSG BY error-message-data-item-name
```

Here’s the FileStat-Msgs.cpy copybook:

```
EVALUATE STATUS
  WHEN 00 MOVE 'SUCCESS                ' TO MSG
  WHEN 02 MOVE 'SUCCESS DUPLICATE      ' TO MSG
  WHEN 04 MOVE 'SUCCESS INCOMPLETE     ' TO MSG
  WHEN 05 MOVE 'SUCCESS OPTIONAL       ' TO MSG
  WHEN 07 MOVE 'SUCCESS NO UNIT        ' TO MSG
  WHEN 10 MOVE 'END OF FILE             ' TO MSG
  WHEN 14 MOVE 'OUT OF KEY RANGE       ' TO MSG
  WHEN 21 MOVE 'KEY INVALID             ' TO MSG
  WHEN 22 MOVE 'KEY EXISTS              ' TO MSG
  WHEN 23 MOVE 'KEY NOT EXISTS         ' TO MSG
  WHEN 30 MOVE 'PERMANENT ERROR         ' TO MSG
  WHEN 31 MOVE 'INCONSISTENT FILENAME  ' TO MSG
  WHEN 34 MOVE 'BOUNDARY VIOLATION     ' TO MSG
  WHEN 35 MOVE 'FILE NOT FOUND         ' TO MSG
  WHEN 37 MOVE 'PERMISSION DENIED      ' TO MSG
  WHEN 38 MOVE 'CLOSED WITH LOCK       ' TO MSG
  WHEN 39 MOVE 'CONFLICT ATTRIBUTE     ' TO MSG
  WHEN 41 MOVE 'ALREADY OPEN           ' TO MSG
  WHEN 42 MOVE 'NOT OPEN                ' TO MSG
  WHEN 43 MOVE 'READ NOT DONE          ' TO MSG
  WHEN 44 MOVE 'RECORD OVERFLOW        ' TO MSG
  WHEN 46 MOVE 'READ ERROR              ' TO MSG
  WHEN 47 MOVE 'INPUT DENIED           ' TO MSG
  WHEN 48 MOVE 'OUTPUT DENIED          ' TO MSG
  WHEN 49 MOVE 'I/O DENIED             ' TO MSG
  WHEN 51 MOVE 'RECORD LOCKED          ' TO MSG
  WHEN 52 MOVE 'END-OF-PAGE            ' TO MSG
  WHEN 57 MOVE 'I/O LINAGE             ' TO MSG
  WHEN 61 MOVE 'FILE SHARING FAILURE   ' TO MSG
  WHEN 91 MOVE 'FILE NOT AVAILABLE     ' TO MSG
END-EVALUATE .
```

10.2. COBDUMP – A Hex/ASCII Data Dump Subroutine

COBDUMP is a useful little utility subroutine to produce a formatted hexadecimal and character dump of the data area passed to it.

If you follow the GNU COBOL forums, you've undoubtedly heard about the CBL_OC_DUMP subroutine that was the winning entry in a GNU COBOL programming contest. It's a great tool for producing data dumps, and it's now included in the official GNU COBOL distributions.

For now though, I'll keep using my good ol' "COBDUMP" routine. It's been my travelling companion from COBOL job to COBOL job since 1971. Here it is, all tuned up for GNU COBOL, with new tires and a fresh coat of paint.

```

=====
1      >>SOURCE FORMAT IS FIXED
2      IDENTIFICATION DIVISION.
3      PROGRAM-ID.      COBDUMP.
4      *>*****
5      *> This is an OpenCOBOL subroutine that will generate a      **
6      *> formatted Hex/Char dump of a storage area.  To use this  **
7      *> subroutine, simply CALL it as follows:                    **
8      *>                                                            **
9      *> CALL "COBDUMP" USING <data-item>                          **
10     *> [ <length> ]                                             **
11     *>                                                            **
12     *> If specified, the <length> argument specifies how many   **
13     *> bytes of <data-item> are to be dumped.  If absent, all of **
14     *> <data-item> will be dumped (i.e. LENGTH(<data-item>) will **
15     *> be assumed for <length>).                                  **
16     *>                                                            **
17     *> >>> Note that the subroutine name MUST be specified in  <<< **
18     *> >>> UPPERCASE                                           <<< **
19     *>                                                            **
20     *> The dump is generated to STDERR, so you may pipe it to a  **
21     *> file when you execute your program using "2> file".      **
22     *>                                                            **
23     *> AUTHOR:          GARY L. CUTLER                            **
24     *>                 CutlerGL@gmail.com                       **
25     *>                                                            **
26     *> NOTE:           The author has a sentimental attachment to **
27     *>                 this subroutine - it's been around since 1971 **
28     *>                 and it's been converted to and run on 10 dif- **
29     *>                 ferent operating system/compiler environments **
30     *>                                                            **
31     *> DATE-WRITTEN:  October 14, 1971                          **
32     *>                                                            **
33     *>*****
34     *> DATE CHANGE DESCRIPTION                                    **
35     *> ===== **
36     *> GC1071 Initial coding - Univac Dept. of Defense COBOL '68  **
37     *> GC0577 Converted to Univac ASCII COBOL (ACOB) - COBOL '74  **
38     *> GC1182 Converted to Univac UTS4000 COBOL - COBOL '74 w/   **
39     *> SCREEN SECTION enhancements                               **
40     *> GC0883 Converted to Honeywell/Bull COBOL - COBOL '74     **
41     *> GC0983 Converted to IBM VS COBOL - COBOL '74             **
42     *> GC0887 Converted to IBM VS COBOL II - COBOL '85          **
43     *> GC1294 Converted to Micro Focus COBOL V3.0 - COBOL '85 w/ **
44     *> extensions                                               **
45     *> GC0703 Converted to Unisys Universal Compiling System (UCS) **
46     *> COBOL (UCOB) - COBOL '85                                 **
47     *> GC1204 Converted to Unisys Object COBOL (OCOB) - COBOL 2002 **
48     *> GC0609 Converted to OpenCOBOL 1.1 - COBOL '85 w/ some COBOL **
49     *> 2002 features                                             **
50     *> GC0410 Enhanced to make 2nd argument (buffer length)    **
51     *> optional                                                 **
52     *> GC0211 Ported to IBM Enterprise COBOL                    **
53     *> GC0612 Updated for OpenCOBOL 2.0                        **
54     *>*****
=====

```

```

Line  Statement
=====
55  ENVIRONMENT DIVISION.
56  CONFIGURATION SECTION.
57  REPOSITORY.
58  FUNCTION ALL INTRINSIC.
59  DATA DIVISION.
60  WORKING-STORAGE SECTION.
61  01 WS-Addr-PTR          USAGE POINTER.
62  01 WS-Addr-NUM REDEFINES WS-Addr-PTR
63                               USAGE BINARY-LONG.
64
65  01 WS-Addr-SUB          USAGE BINARY-CHAR.
66
67  01 WS-Addr-Value-NUM    USAGE BINARY-LONG.
68
69  01 WS-Buffer-Byte-CHR.
70  05 WS-Buffer-Byte-NUM    USAGE BINARY-CHAR.
71
72  01 WS-Buffer-Length-NUM  USAGE BINARY-LONG.
73
74  01 WS-Buffer-SUB        PIC 9(4) COMP-5.
75
76  01 WS-Hex-Digit-TXT VALUE '0123456789ABCDEF'.
77  05 WS-Hex-Digit-CHR      OCCURS 16 TIMES
78                               PIC X(1).
79
80  01 WS-Nibble-SUB        PIC 9(1) COMP-5.
81
82  01 WS-Nibble-Left-SUB    PIC 9(1) COMP-5.
83
84  01 WS-Nibble-Right-SUB   PIC 9(1) COMP-5.
85
86  01 WS-Output-Detail-TXT.
87  05 WS-OD-Addr-TXT.
88  10 WS-OD-Addr-Hex-CHR    OCCURS 8 TIMES PIC X.
89  05 FILLER                PIC X(1).
90  05 WS-OD-Relative-Byte-NUM PIC Z(3)9.
91  05 FILLER                PIC X(1).
92  05 WS-OD-Hex-TXT         OCCURS 16 TIMES.
93  10 WS-OD-Hex-1-CHR       PIC X.
94  10 WS-OD-Hex-2-CHR       PIC X.
95  10 FILLER                PIC X.
96  05 WS-OD-ASCII-Data-TXT.
97  10 WS-OD-ASCII-CHR       OCCURS 16 TIMES
98                               PIC X.
99
100 01 WS-Output-SUB         PIC 9(2) COMP-5.
101
102 >>SOURCE FORMAT IS FREE
103 01 WS-Output-Header-1-TXT.
104 05 VALUE '<-Addr-> Byte <----- Hexadecimal ''-----> <---- Char ---->' PIC X(80).
105
106 01 WS-Output-Header-2-TXT.
107 05 VALUE '======' PIC X(80).
108 >>SOURCE FORMAT IS FIXED

```

Line Statement

```

=====
109
110 LINKAGE SECTION.
111 01 L-Buffer-TXT PIC X ANY LENGTH.
112
113 01 L-Buffer-Length-NUM USAGE BINARY-LONG.
114
115 PROCEDURE DIVISION USING L-Buffer-TXT,
116 OPTIONAL L-Buffer-Length-NUM.
117 000-Main SECTION.
118 IF NUMBER-OF-CALL-PARAMETERS = 1
119 MOVE LENGTH(L-Buffer-TXT) TO WS-Buffer-Length-NUM
120 ELSE
121 MOVE L-Buffer-Length-NUM TO WS-Buffer-Length-NUM
122 END-IF
123 MOVE SPACES TO WS-Output-Detail-TXT
124 SET WS-Addr-PTR TO ADDRESS OF L-Buffer-TXT
125 PERFORM 100-Generate-Address
126 MOVE 0 TO WS-Output-SUB
127 DISPLAY WS-Output-Header-1-TXT UPON SYSERR
128 DISPLAY WS-Output-Header-2-TXT UPON SYSERR
129 PERFORM VARYING WS-Buffer-SUB FROM 1 BY 1
130 UNTIL WS-Buffer-SUB > WS-Buffer-Length-NUM
131 ADD 1 TO WS-Output-SUB
132 IF WS-Output-SUB = 1
133 MOVE WS-Buffer-SUB TO WS-OD-Relative-Byte-NUM
134 END-IF
135 MOVE L-Buffer-TXT (WS-Buffer-SUB : 1)
136 TO WS-OD-ASCII-CHR (WS-Output-SUB)
137 WS-Buffer-Byte-CHR
138 DIVIDE WS-Buffer-Byte-NUM BY 16
139 GIVING WS-Nibble-Left-SUB
140 REMAINDER WS-Nibble-Right-SUB
141 ADD 1 TO WS-Nibble-Left-SUB
142 WS-Nibble-Right-SUB
143 MOVE WS-Hex-Digit-CHR (WS-Nibble-Left-SUB)
144 TO WS-OD-Hex-1-CHR (WS-Output-SUB)
145 MOVE WS-Hex-Digit-CHR (WS-Nibble-Right-SUB)
146 TO WS-OD-Hex-2-CHR (WS-Output-SUB)
147 IF WS-Output-SUB = 16
148 CALL "C$PRINTABLE" USING WS-OD-ASCII-Data-TXT
149 DISPLAY WS-Output-Detail-TXT UPON SYSERR
150 MOVE SPACES TO WS-Output-Detail-TXT
151 MOVE 0 TO WS-Output-SUB
152 SET WS-Addr-PTR UP BY 16
153 PERFORM 100-Generate-Address
154 END-IF
155 END-PERFORM
156 IF WS-Output-SUB > 0
157 CALL "C$PRINTABLE" USING WS-OD-ASCII-Data-TXT
158 DISPLAY WS-Output-Detail-TXT UPON SYSERR
159 END-IF
160 EXIT PROGRAM
161 .
162 100-Generate-Address SECTION.

```

Line Statement

Page: 4

```
=====
163      MOVE 8 TO WS-Addr-SUB
164      MOVE WS-Addr-NUM TO WS-Addr-Value-NUM
165      MOVE ALL '0' TO WS-OD-Addr-TXT
166      PERFORM WITH TEST BEFORE UNTIL WS-Addr-Value-NUM = 0
167          DIVIDE WS-Addr-Value-NUM BY 16
168              GIVING WS-Addr-Value-NUM
169              REMAINDER WS-Nibble-SUB
170      ADD 1 TO WS-Nibble-SUB
171      MOVE WS-Hex-Digit-CHR (WS-Nibble-SUB)
172          TO WS-OD-Addr-Hex-CHR (WS-Addr-SUB)
173      SUBTRACT 1 FROM WS-Addr-SUB
174      END-PERFORM
175      .
```


GNU COBOL V2.0 11FEB2012 Cross-Reference Listing - GCic for Windows/MinGW Copyright (C) 2009 - 2013, Gary L. Cutler, GPL 2013/11/21
 E:/GNU-COBOL/samples/COBDUMP.cbl Page: 5

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)							
COBDUMP	000-Main	117	PROCEDURE								
COBDUMP	100-Generate-Address	162	PROCEDURE	125	153						
COBDUMP	L-Buffer-Length-NUM	113	LINKAGE	116	121						
COBDUMP	L-Buffer-TXT	111	LINKAGE	115	119	124	135				
COBDUMP	LENGTH		PROCEDURE	119							
COBDUMP	NUMBER-OF-CALL-PARAMETERS		PROCEDURE	118							
COBDUMP	WS-Addr-NUM	62	WORKING-STORAGE	164							
COBDUMP	WS-Addr-PTR	61	WORKING-STORAGE	62	124*	152*					
COBDUMP	WS-Addr-SUB	65	WORKING-STORAGE	163*	172	173					
COBDUMP	WS-Addr-Value-NUM	67	WORKING-STORAGE	164*	166	167	168*				
COBDUMP	WS-Buffer-Byte-CHR	69	WORKING-STORAGE	137							
COBDUMP	WS-Buffer-Byte-NUM	70	WORKING-STORAGE	138							
COBDUMP	WS-Buffer-Length-NUM	72	WORKING-STORAGE	119*	121*	130					
COBDUMP	WS-Buffer-SUB	74	WORKING-STORAGE	129*	130	133	135				
COBDUMP	WS-Hex-Digit-CHR	77	WORKING-STORAGE	143	145	171					
COBDUMP	WS-Hex-Digit-TXT	76	WORKING-STORAGE								
COBDUMP	WS-Nibble-Left-SUB	82	WORKING-STORAGE	139*	141*	143					
COBDUMP	WS-Nibble-Right-SUB	84	WORKING-STORAGE	140*	142*	145					
COBDUMP	WS-Nibble-SUB	80	WORKING-STORAGE	169*	170*	171					
COBDUMP	WS-OD-Addr-Hex-CHR	88	WORKING-STORAGE	172*							
COBDUMP	WS-OD-Addr-TXT	87	WORKING-STORAGE	165*							
COBDUMP	WS-OD-ASCII-CHR	97	WORKING-STORAGE	136*							
COBDUMP	WS-OD-ASCII-Data-TXT	96	WORKING-STORAGE	148*	157*						
COBDUMP	WS-OD-Hex-1-CHR	93	WORKING-STORAGE	144*							
COBDUMP	WS-OD-Hex-2-CHR	94	WORKING-STORAGE	146*							
COBDUMP	WS-OD-Hex-TXT	92	WORKING-STORAGE								
COBDUMP	WS-OD-Relative-Byte-NUM	90	WORKING-STORAGE	133*							
COBDUMP	WS-Output-Detail-TXT	86	WORKING-STORAGE	123*	149	150*	158				
COBDUMP	WS-Output-Header-1-TXT	103	WORKING-STORAGE	127							
COBDUMP	WS-Output-Header-2-TXT	106	WORKING-STORAGE	128							
COBDUMP	WS-Output-SUB	100	WORKING-STORAGE	126*	131*	132	136	144	146	147	151*
				156							

10.3. DAY-FROM-DATE – A Function to Determine Day of Week From a Date

DAY-FROM-DATE is a user-defined function that accepts a single argument – either a 7-digit Julian date in the form “yyyyddd” or an 8-digit Gregorian date in the form “yyyymmdd”. This argument may be supplied either as a PIC 9(n) USAGE DISPLAY data item (n=7 or 8) or as a 7- or 8-digit numeric literal.

The subroutine will determine if the supplied date is a valid date in the year range 0000 thru 9999 and what day of the week that date fell on.

The value returned will be zero if the date argument was invalid or an integer in the range 1-7, representing Sunday thru Saturday.

```

=====
 1      >>SOURCE FORMAT IS FIXED
 2      IDENTIFICATION DIVISION.
 3      FUNCTION-ID. DAY-FROM-DATE.
 4      *>*****
 5      *> This GNU COBOL user-defined function converts a Gregorian or **
 6      *> Julian date into a numeric day of the week. **
 7      *>*****
 8      *> Arguments: **
 9      *> **
10      *> Calendar-Date A PIC 9 data item or numeric literal which **
11      *> will be treated as a calendar date as fol- **
12      *> lows: **
13      *> **
14      *> 7-digit value: Interpreted as a Julian date **
15      *> in the form yyyyddd **
16      *> 8-digit value: Interpreted as a Gregorian **
17      *> date in the form yyyymmdd **
18      *> **
19      *> The result returned will be one of the following: **
20      *> **
21      *> 0: The supplied date is invalid **
22      *> 1: The supplied date is a Sunday **
23      *> 2: The supplied date is a Monday **
24      *> . **
25      *> . **
26      *> . **
27      *> 7: The supplied date is a Saturday **
28      *>*****
29      ENVIRONMENT DIVISION.
30      CONFIGURATION SECTION.
31      REPOSITORY.
32      FUNCTION ALL INTRINSIC.
33      DATA DIVISION.
34      WORKING-STORAGE SECTION.
35      01 WS-Input-Date-DT.
36      05 WS-ID-YYYY-NUM PIC 9(4).
37      05 WS-ID-MM-NUM PIC 9(2).
38      05 WS-ID-DD-NUM PIC 9(2).
39      01 WS-Y-NUM BINARY-LONG.
40      01 WS-M-NUM BINARY-LONG.
41      01 WS-Temp-NUM BINARY-LONG.
42      LINKAGE SECTION.
43      01 L-Input-Date-DT PIC 9 ANY LENGTH.
44      01 L-Output-Day-NUM USAGE BINARY-LONG
45      SIGNED.
46      PROCEDURE DIVISION USING L-Input-Date-DT
47      RETURNING L-Output-Day-NUM.
48      000-Main SECTION.
49      CALL "C$PARAMSIZE" USING 1
50      EVALUATE RETURN-CODE
51      WHEN 7
52      IF TEST-DAY-YYYYDDD(L-Input-Date-DT) > 0
53      MOVE 0 TO L-Output-Day-NUM
54      GOBACK
=====

```

Line Statement

Page: 2

```

=====
55         END-IF
56         MOVE DATE-OF-INTEGGER(INTEGER-OF-DAY(L-Input-Date-DT))
57         TO WS-Input-Date-DT
58     WHEN 8
59         IF TEST-DATE-YYYYMMDD(L-Input-Date-DT) > 0
60             MOVE 0 TO L-Output-Day-NUM
61             GOBACK
62         END-IF
63         MOVE L-Input-Date-DT TO WS-Input-Date-DT
64     WHEN OTHER
65         MOVE 0 TO L-Output-Day-NUM
66         GOBACK
67     END-EVALUATE
68     *> IF january OR february
69     *>     y = year - 1
70     *>     m = month + 10
71     *> ELSE
72     *>     y = year
73     *>     m = month - 2
74     *> END-IF
75     *> For Gregorian calendar:
76     *>     result = (day + y + y/4 - y/100 + y/400 + (31*m)/12) mod 7
77     *> (All divisions are integer divisions, discarding any remainder)
78     IF WS-ID-MM-NUM = 1 OR 2
79         SUBTRACT 1 FROM WS-ID-YYYY-NUM GIVING WS-Y-NUM
80         ADD WS-ID-MM-NUM, 10 GIVING WS-M-NUM
81     ELSE
82         MOVE WS-ID-YYYY-NUM TO WS-Y-NUM
83         SUBTRACT 2 FROM WS-ID-MM-NUM GIVING WS-M-NUM
84     END-IF
85     COMPUTE L-Output-Day-NUM =
86         WS-ID-DD-NUM
87         + WS-Y-NUM
88         + INTEGER(WS-Y-NUM/4)
89         - INTEGER(WS-Y-NUM/100)
90         + INTEGER(WS-Y-NUM/400)
91         + INTEGER((31*WS-M-NUM)/12)
92     DIVIDE L-Output-Day-NUM BY 7
93         GIVING WS-Temp-NUM
94         REMAINDER L-Output-Day-NUM
95     ADD 1 TO L-Output-Day-NUM
96     GOBACK
97     .
98
=====

```

GNU COBOL V2.0 11FEB2012 Cross-Reference Listing - GCic for Windows/MinGW Copyright (C) 2009 - 2013, Gary L. Cutler, GPL 2013/11/21

E:/GNU-COBOL/samples/DAY-FROM-DATE.cbl

Page: 3

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)						
DAY-FROM-DATE	000-Main	48	PROCEDURE							
DAY-FROM-DATE	DATE-OF-INTEG		PROCEDURE	56						
DAY-FROM-DATE	INTEGER		PROCEDURE	88	89	90	91			
DAY-FROM-DATE	INTEGER-OF-DAY		PROCEDURE	56						
DAY-FROM-DATE	L-Input-Date-DT	43	LINKAGE	46	52	56	59	63		
DAY-FROM-DATE	L-Output-Day-NUM	44	LINKAGE	47	53*	60*	65*	85*	92	94*
DAY-FROM-DATE	RETURN-CODE		PROCEDURE	50						95*
DAY-FROM-DATE	TEST-DATE-YYYYMMDD		PROCEDURE	59						
DAY-FROM-DATE	TEST-DAY-YYYYDDD		PROCEDURE	52						
DAY-FROM-DATE	WS-ID-DD-NUM	38	WORKING-STORAGE	86						
DAY-FROM-DATE	WS-ID-MM-NUM	37	WORKING-STORAGE	78	80	83				
DAY-FROM-DATE	WS-ID-YYYY-NUM	36	WORKING-STORAGE	79	82					
DAY-FROM-DATE	WS-Input-Date-DT	35	WORKING-STORAGE	57*	63*					
DAY-FROM-DATE	WS-M-NUM	40	WORKING-STORAGE	80*	83*	91				
DAY-FROM-DATE	WS-Temp-NUM	41	WORKING-STORAGE	93*						
DAY-FROM-DATE	WS-Y-NUM	39	WORKING-STORAGE	79*	82*	87	88	89	90	

10.4. GCic – an Interactive GNU COBOL Full-Screen Compiler Front-End

This is MUCH more than a mere demonstration program – it’s also a very practical utility! The “GCic” (GNU COBOL Interactive Compiler) is a TUI (Textual User Interface) program that may be used as a full-screen interface to the “cobc” compiler. In addition, GCic can produce neat, concise and useful cross-reference listings of GNU COBOL programs, showing not only where user-defined names and built-in registers and intrinsic functions are referenced, but also where user-defined data items ARE MODIFIED by program code! The program is well documented (IMHO) and you should find it fairly easy to follow. The GCic.cbl program was written to work with a native Windows or Windows/MinGW build of GNU COBOL as well as a Windows/Cygwin, UNIX or OS X build.

Source listings generated by GCic will show the original source code of your programs, with all indentation and comments preserved. Additionally, any COPYed code will be included in the listing immediately (in compressed form) following the COPY statement that triggered its inclusion into your program.

Cross-reference listings will show all user-defined data items and procedures as well as intrinsic function and special register references. In addition to showing the line numbers at which items were defined and referenced, those references that MODIFY the contents of the data item will have an asterisk appended to them.

Line Statement

Page: 1

```

=====
1      >>SOURCE FORMAT IS FIXED
2      *> CONFIGURATION SETTINGS: Set these switches before compiling:
3      *>
4      *> LINEDRAW Set to:
5      *>   0   To use spaces (no lines)
6      *>   1   To use the line-drawing character set (PC codepage 437)
7      *>   2   To use conventional ASCII characters (+, -, |)
8      *>
9      *>       OSX USERS - To use the linedrawing character set,
10     *>           set your 'terminal' font to 'Lucida Console'
11     *>
12     *> OS       Set to one of the following:
13     *>       'CYGWIN'   For a Windows/Cygwin version
14     *>       'MINGW'    For a Windows/MinGW version
15     *>       'OSX'      For a Macintosh OSX version
16     *>       'UNIX'     For a Unix/Linux version
17     *>       'WINDOWS'  For a Native Windows version
18     *>
19     *> SELCHAR  Set to the desired single character to be used as the red
20     *>           'feature selected' character on the screen.
21     *>           SUGGESTIONS: '>', '*', '=', '+'
22     *>
23 GC0712 >>DEFINE CONSTANT LINEDRAW AS 1
24 GC0712 >>DEFINE CONSTANT OS AS 'MINGW'
25 GC0712 >>DEFINE CONSTANT SELCHAR AS '>'
26     *> -----
27     *> Now set these switches to establish initial (default) settings
28     *> for the various on-screen options. Set them to a value of
29     *> 0 if they are to be 'OFF' and 1 if they are to be 'ON'
30     *>
31 GC0712 >>DEFINE CONSTANT F1 AS 0 *> Assume WITH DEBUGGING MODE
32 GC0712 >>DEFINE CONSTANT F2 AS 0 *> Procedure+Statement Trace
33 GC0712 >>DEFINE CONSTANT F3 AS 0 *> Make A Library (DLL)
34 GC0712 >>DEFINE CONSTANT F4 AS 0 *> Execute If Compilation OK
35 GC0712 >>DEFINE CONSTANT F5 AS 0 *> Generate Listings
36 GC0712 >>DEFINE CONSTANT F6 AS 1 *> "FUNCTION" Is Optional
37 GC0712 >>DEFINE CONSTANT F7 AS 1 *> Enable All Warnings
38 GC0712 >>DEFINE CONSTANT F8 AS 0 *> Source Is Free-Format
39 GC0712 >>DEFINE CONSTANT F9 AS 1 *> No COMP/BINARY Truncation
40 GC0712 >>DEFINE CONSTANT F12 AS 4 *> Default config file (1-7):
41     *>           1 = BS2000
42     *>           2 = COBOL85
43     *>           3 = COBOL2002
44     *>           4 = DEFAULT
45     *>           5 = IBM
46     *>           6 = MF (i.e. Microfocus)
47     *>           7 = MVS
48     *> -----
49     *> END CONFIGURATION SETTINGS
=====

```

Line Statement

Page: 2

```

=====
50 /
51 IDENTIFICATION DIVISION.
52 PROGRAM-ID. GCic.
53 * *****
54 * >NOTE< >NOTE< >NOTE< >NOTE< >NOTE< >NOTE< **
55 * **
56 *> If this program is compiled with '-fdebugging-line', you **
57 *> will need to pipe SYSERR to a text file when executing GCic **
58 *> (by adding the text '2> filename' to the end of the GCic **
59 *> command). You may also need to press the ENTER key when **
60 *> GCic is finished. **
61 * *****
62 *> This program provides a Textual User Interface (TUI) to the **
63 *> process of compiling and (optionally) executing a GNU COBOL **
64 *> program. **
65 * **
66 *> This programs execution syntax is as follows: **
67 * **
68 *> GCic <program-path-and-filename> [ <switch>... ] **
69 * **
70 *> Once executed, a display screen will be presented showing **
71 *> the compilation options that will be used. The user will **
72 *> have the opportunity to change options, specify new ones **
73 *> and specify any program execution arguments to be used if **
74 *> you select the 'Execute' option. When you press the Enter **
75 *> key the program will be compiled. **
76 * **
77 *> The SCREEN SECTION contains an image of the screen. **
78 * **
79 *> The '010-Parse-Args' section in the PROCEDURE DIVISION has **
80 *> documentation on switches and their function. **
81 * *****
82 * **
83 *> AUTHOR: GARY L. CUTLER **
84 *> CutlerGL@gmail.com **
85 *> Copyright (C) 2009-2013, Gary L. Cutler, GPL **
86 * **
87 *> DATE-WRITTEN: June 14, 2009 **
88 * **
89 * *****
90 *> DATE CHANGE DESCRIPTION **
91 *> ===== **
92 *> GC0609 Don't display compiler messages file if compilation **
93 *> Is successful. Also don't display messages if the **
94 *> output file is busy (just put a message on the **
95 *> screen, leave the OC screen up & let the user fix **
96 *> the problem & resubmit. **
97 *> GC0709 When 'EXECUTE' is selected, a 'FILE BUSY' error will **
98 *> still cause the (old) executable to be launched. **
99 *> Also, the 'EXTRA SWITCHES' field is being ignored. **
100 *> Changed the title bar to lowlighted reverse video & **
101 *> the message area to highlighted reverse-video. **
102 *> GC0809 Add a SPACE in front of command-line args when **
103 *> executing users program. Add a SPACE after the **

```


Line Statement

Page: 3

```

=====
104      *>      -ftraceall switch when building cobc command.      **
105      *> GC0909 Convert to work on Cygwin/Linux as well as MinGW  **
106      *> GC0310 Virtualized the key codes for S-F1 thru S-F7 as they **
107      *>      differ depending upon whether PDCurses or NCurses is  **
108      *>      being used.                                          **
109      *> GC0410 Introduced the cross-reference and source listing  **
110      *>      features. Also fixed a bug in @EXTRA switch proces-  **
111      *>      sing where garbage will result if more than the     **
112      *>      @EXTRA switch is specified.                          **
113      *> GC1010 Corrected several problems reported by Vince Coen: **
114      *>      1) Listing/Xref wouldn't work if '-I' additional   **
115      *>      cobc switch specified.                               **
116      *>      2) Programs coded with lowercase reserved words did **
117      *>      not get parsed properly when generating listing     **
118      *>      and/or xref reports.                                 **
119      *>      3) Reliance on a TEMP environment variable caused  **
120      *>      non-recoverable errors when generating listing     **
121      *>      and/or xref reports in a session that lacks a     **
122      *>      TEMP variable.                                        **
123      *>      As a result of this change, GCic no longer runs a  **
124      *>      second 'cobc' when generating listing and/or xref   **
125      *>      reports. A '-save-temps' (without '=dir') specified **
126      *>      in the @EXTRA options field will be ignored. A     **
127      *>      '-save-temps=dir' specified in the @EXTRA options  **
128      *>      field will negate both the @XREF and @SOURCE opts,  **
129      *>      if specified.                                         **
130      *> GC0711 Tailored for 29APR2011 version of GNU COBOL 2.0  **
131      *> GC0712 Replaced all switches with configuration settings; **
132      *>      Tailored for 11FEB2012 version of GNU COBOL 2.0;   **
133      *>      Reformatted screen layout to fit a 24x80 screen    **
134      *>      rather than a 25x81 screen and to accommodate shell  **
135      *>      environments having only F1-F12 (like 'terminal' in  **
136      *>      OSX); Fully tested under OSX (required a few altera-  **
137      *>      tions); Expanded both extra-options and runtime-     **
138      *>      arguments areas to TWO lines (152 chars total) each; **
139      *>      Added support for MF/IBM/BS2000 listing-control     **
140      *>      directives EJECT,SKIP1,SKIP2,SKIP3 (any of these in **
141      *>      copybooks will be ignored)                             **
142      *> GC0313 Expand the source code record from 80 chars to 256 **
143      *>      to facilitate looking for "LINKAGE SECTION" in a   **
144      *>      free-format file.                                       **
145      *> GC1113 Edited to support the change of "OpenCOBOL" to "GNU **
146      *>      COBOL"                                                **
147      *> *****
148      ENVIRONMENT DIVISION.
149      CONFIGURATION SECTION.
150      REPOSITORY.
151          FUNCTION ALL INTRINSIC.
152      INPUT-OUTPUT SECTION.
153      FILE-CONTROL.
154 GC1010      SELECT F-Cobc-Output-FILE      ASSIGN TO WS-Listing-Filename-TXT
155                                     ORGANIZATION IS LINE SEQUENTIAL.
156
157      SELECT F-Source-Code-FILE      ASSIGN TO WS-File-Name-TXT
=====

```

GNU COBOL V2.0 11FEB2012 Source Listing - GCic for Windows/MinGW Copyright (C) 2009 - 2013, Gary L. Cutler, GPL 2013/11/21
E:/GNU-COBOL/samples/GCic.cbl Page: 4

Line Statement

```
=====
158 ORGANIZATION IS LINE SEQUENTIAL
159 FILE STATUS IS WS-FSM-Status-CD.
=====
```

Line Statement

Page: 5

```
=====
160      /
161      DATA DIVISION.
162      FILE SECTION.
163      FD F-Cobc-Output-FILE.
164      01 F-Cobc-Output-REC          PIC X(256).
165
166      FD F-Source-Code-FILE.
167 GC0313 01 F-Source-Code-REC      PIC X(256).
168
169      WORKING-STORAGE SECTION.
170      COPY screenio.
171
172      78 COB-COLOR-BLACK VALUE 0.
173      78 COB-COLOR-BLUE VALUE 1.
174      78 COB-COLOR-GREEN VALUE 2.
175      78 COB-COLOR-CYAN VALUE 3.
176      78 COB-COLOR-RED VALUE 4.
177      78 COB-COLOR-MAGENTA VALUE 5.
178      78 COB-COLOR-YELLOW VALUE 6.
179      78 COB-COLOR-WHITE VALUE 7.
180      78 COB-SCR-OK VALUE 0.
181      78 COB-SCR-F1 VALUE 1001.
182      78 COB-SCR-F2 VALUE 1002.
183      78 COB-SCR-F3 VALUE 1003.
184      78 COB-SCR-F4 VALUE 1004.
185      78 COB-SCR-F5 VALUE 1005.
186      78 COB-SCR-F6 VALUE 1006.
187      78 COB-SCR-F7 VALUE 1007.
188      78 COB-SCR-F8 VALUE 1008.
189      78 COB-SCR-F9 VALUE 1009.
190      78 COB-SCR-F10 VALUE 1010.
191      78 COB-SCR-F11 VALUE 1011.
192      78 COB-SCR-F12 VALUE 1012.
193      78 COB-SCR-F13 VALUE 1013.
194      78 COB-SCR-F14 VALUE 1014.
195      78 COB-SCR-F15 VALUE 1015.
196      78 COB-SCR-F16 VALUE 1016.
197      78 COB-SCR-F17 VALUE 1017.
198      78 COB-SCR-F18 VALUE 1018.
199      78 COB-SCR-F19 VALUE 1019.
200      78 COB-SCR-F20 VALUE 1020.
201      78 COB-SCR-F21 VALUE 1021.
202      78 COB-SCR-F22 VALUE 1022.
203      78 COB-SCR-F23 VALUE 1023.
204      78 COB-SCR-F24 VALUE 1024.
205      78 COB-SCR-F25 VALUE 1025.
206      78 COB-SCR-F26 VALUE 1026.
207      78 COB-SCR-F27 VALUE 1027.
208      78 COB-SCR-F28 VALUE 1028.
209      78 COB-SCR-F29 VALUE 1029.
210      78 COB-SCR-F30 VALUE 1030.
211      78 COB-SCR-F31 VALUE 1031.
212      78 COB-SCR-F32 VALUE 1032.
213      78 COB-SCR-F33 VALUE 1033.
214      78 COB-SCR-F34 VALUE 1034.
```

Line Statement

Page: 6

```

=====
 78 COB-SCR-F35 VALUE 1035.
 78 COB-SCR-F36 VALUE 1036.
 78 COB-SCR-F37 VALUE 1037.
 78 COB-SCR-F38 VALUE 1038.
 78 COB-SCR-F39 VALUE 1039.
 78 COB-SCR-F40 VALUE 1040.
 78 COB-SCR-F41 VALUE 1041.
 78 COB-SCR-F42 VALUE 1042.
 78 COB-SCR-F43 VALUE 1043.
 78 COB-SCR-F44 VALUE 1044.
 78 COB-SCR-F45 VALUE 1045.
 78 COB-SCR-F46 VALUE 1046.
 78 COB-SCR-F47 VALUE 1047.
 78 COB-SCR-F48 VALUE 1048.
 78 COB-SCR-F49 VALUE 1049.
 78 COB-SCR-F50 VALUE 1050.
 78 COB-SCR-F51 VALUE 1051.
 78 COB-SCR-F52 VALUE 1052.
 78 COB-SCR-F53 VALUE 1053.
 78 COB-SCR-F54 VALUE 1054.
 78 COB-SCR-F55 VALUE 1055.
 78 COB-SCR-F56 VALUE 1056.
 78 COB-SCR-F57 VALUE 1057.
 78 COB-SCR-F58 VALUE 1058.
 78 COB-SCR-F59 VALUE 1059.
 78 COB-SCR-F60 VALUE 1060.
 78 COB-SCR-F61 VALUE 1061.
 78 COB-SCR-F62 VALUE 1062.
 78 COB-SCR-F63 VALUE 1063.
 78 COB-SCR-F64 VALUE 1064.
 78 COB-SCR-PAGE_UP VALUE 2001.
 78 COB-SCR-PAGE_DOWN VALUE 2002.
 78 COB-SCR-KEY-UP VALUE 2003.
 78 COB-SCR-KEY-DOWN VALUE 2004.
 78 COB-SCR-ESC VALUE 2005.
 78 COB-SCR-PRINT VALUE 2006.
 78 COB-SCR-NO-FIELD VALUE 8000.
 78 COB-SCR-TIME-OUT VALUE 8001.
 78 COB-SCR-FATAL VALUE 9000.
 78 COB-SCR-MAX-FIELD VALUE 9001.
171
172 GC0712 01 WS-Compilation-Switches-TXT.
173 GC0712 05 WS-CS-Args-TXT VALUE SPACES.
174 GC0712 10 WS-CS-Arg-H1-TXT PIC X(76).
175 GC0712 10 WS-CS-Arg-H2-TXT PIC X(76).
176 GC0712 05 WS-CS-Filenames-TXT.
177 GC0712 10 VALUE 'BS2000' PIC X(9).
178 GC0712 10 VALUE 'COBOL85' PIC X(9).
179 GC0712 10 VALUE 'COBOL2002' PIC X(9).
180 GC0712 10 VALUE 'DEFAULT' PIC X(9).
181 GC0712 10 VALUE 'IBM' PIC X(9).
182 GC0712 10 VALUE 'MF' PIC X(9).
183 GC0712 10 VALUE 'MVS' PIC X(9).
184 GC0712 05 WS-CS-Filenames-Table-TXT REDEFINES WS-CS-Filenames-TXT.

```

Line Statement

Page: 7

```

=====
185 GC0712      10 WS-CS-Filename-TXT          OCCURS 7 TIMES
186 GC0712          PIC X(9).
187 GC0712 >>IF F12 < 1
188 GC0712      05 WS-CS-Config-NUM      VALUE 4      PIC 9(1).
189 GC0712 >>ELIF F12 > 7
190 GC0712      05 WS-CS-Config-NUM      VALUE 4      PIC 9(1).
191 GC0712 >>ELSE
192 GC0712      05 WS-CS-Config-NUM      VALUE F12    PIC 9(1).
193 GC0712 >>END-IF
194 GC0712      05 WS-CS-Extra-TXT VALUE SPACES.
195 GC0712      10 WS-CS-Extra-H1-TXT          PIC X(76).
196 GC0712      10 WS-CS-Extra-H2-TXT          PIC X(76).
197 GC0712      05 WS-CS-Switch-Defaults-TXT.
198 GC0712      10 VALUE F1          PIC 9(1). *> WS-CS-DEBUG-CHR
199 GC0712      10 VALUE F4          PIC 9(1). *> WS-CS-EXECUTE-CHR
200 GC0712      10 VALUE F8          PIC 9(1). *> WS-CS-FREE-CHR
201 GC0712      10 VALUE F3          PIC 9(1). *> WS-CS-LIBRARY-CHR
202 GC0712      10 VALUE F5          PIC 9(1). *> WS-CS-LISTING-CHR
203 GC0712      10 VALUE F6          PIC 9(1). *> WS-CS-NOFUNC-CHR
204 GC0712      10 VALUE F9          PIC 9(1). *> WS-CS-NOTRUNC-CHR
205 GC0712      10 VALUE F2          PIC 9(1). *> WS-CS-TRACEALL-CHR
206 GC0712      10 VALUE F7          PIC 9(1). *> WS-CS-WARNALL-CHR
207 GC0712      05 WS-CS-All-Switches-TXT REDEFINES
208 GC0712          WS-CS-Switch-Defaults-TXT.
209 GC0712      10 WS-CS-DEBUG-CHR          PIC X(1).
210 GC0712      10 WS-CS-EXECUTE-CHR          PIC X(1).
211 GC0712      10 WS-CS-FREE-CHR           PIC X(1).
212 GC0712      10 WS-CS-LIBRARY-CHR          PIC X(1).
213 GC0712      10 WS-CS-LISTING-CHR          PIC X(1).
214 GC0712      10 WS-CS-NOFUNC-CHR          PIC X(1).
215 GC0712      10 WS-CS-NOTRUNC-CHR          PIC X(1).
216 GC0712      10 WS-CS-TRACEALL-CHR          PIC X(1).
217 GC0712      10 WS-CS-WARNALL-CHR          PIC X(1).
218
219 GC0909 01 WS-Cmd-TXT          PIC X(512).
220
221 GC0712 01 WS-Cmd-Args-TXT      PIC X(256).
222
223 GC0712 01 WS-Cmd-End-Quote-CHR  PIC X(1).
224
225 GC0712 01 WS-Cmd-SUB          USAGE BINARY-LONG.
226
227      01 WS-Cobc-Cmd-TXT          PIC X(256).
228
229      01 WS-Config-Fn-TXT          PIC X(12).
230
231 GC1113 01 WS-Delete-Fn-TXT      PIC X(256).
232
233      01 WS-File-Name-TXT.
234      05 WS-FN-CHR          OCCURS 256 TIMES
235          PIC X(1).
236
237      01 WS-File-Status-Message-TXT.
238      05 VALUE 'Status Code: '    PIC X(13).

```

Line Statement

Page: 8

```

=====
239          05 WS-FSM-Status-CD          PIC 9(2).
240          05 VALUE ', Meaning: '      PIC X(11).
241          05 WS-FSM-Msg-TXT          PIC X(25).
242
243 GC0909 01 WS-Horizontal-Line-TXT     PIC X(80).
244 GC0909
245          01 WS-I-SUB                  USAGE BINARY-LONG.
246
247          01 WS-J-SUB                  USAGE BINARY-LONG.
248
249 GC0712 01 WS-Listing-Filename-TXT    PIC X(256).
250
251          01 WS-OC-Compile-DT          PIC XXXX/XX/XXBXX/XX.
252
253 GC0712 >>IF OS = 'CYGWIN'
254 GC0712 01 WS-OS-Dir-CHR              VALUE '/'          PIC X(1).
255 GC0712 78 WS-OS-Exe-Ext-CONST       VALUE '.exe'.
256 GC0712 78 WS-OS-Lib-Ext-CONST       VALUE '.dll'.
257 GC0712 78 WS-OS-Lib-Type-CONST     VALUE 'DLL)'.
258 GC0712 01 WS-OS-Type-CD             VALUE 2            PIC 9(1).
259 GC0712 >>ELIF OS = 'MINGW'
260 GC0712 01 WS-OS-Dir-CHR              VALUE '\\'         PIC X(1).
261 GC0712 78 WS-OS-Exe-Ext-CONST       VALUE '.exe'.
262 GC0712 78 WS-OS-Lib-Ext-CONST       VALUE '.dll'.
263 GC0712 78 WS-OS-Lib-Type-CONST     VALUE 'DLL)'.
264 GC0712 01 WS-OS-Type-CD             VALUE 5            PIC 9(1).
265 GC0712 >>ELIF OS = 'OSX'
266 GC0712 01 WS-OS-Dir-CHR              VALUE '/'          PIC X(1).
267 GC0712 78 WS-OS-Exe-Ext-CONST       VALUE '.'.
268 GC0712 78 WS-OS-Lib-Ext-CONST       VALUE '.dylib'.
269 GC0712 78 WS-OS-Lib-Type-CONST     VALUE 'DYLIB)'.
270 GC0712 01 WS-OS-Type-CD             VALUE 4            PIC 9(1).
271 GC0712 >>ELIF OS = 'UNIX'
272 GC0712 01 WS-OS-Dir-CHR              VALUE '/'          PIC X(1).
273 GC0712 78 WS-OS-Exe-Ext-CONST       VALUE '.'.
274 GC0712 78 WS-OS-Lib-Ext-CONST       VALUE '.so'.
275 GC0712 78 WS-OS-Lib-Type-CONST     VALUE 'SO)'.
276 GC0712 01 WS-OS-Type-CD             VALUE 3            PIC 9(1).
277 GC0712 >>ELIF OS = 'WINDOWS'
278 GC0712 01 WS-OS-Dir-CHR              VALUE '\\'         PIC X(1).
279 GC0712 78 WS-OS-Exe-Ext-CONST       VALUE '.exe'.
280 GC0712 78 WS-OS-Lib-Ext-CONST       VALUE '.dll'.
281 GC0712 78 WS-OS-Lib-Type-CONST     VALUE 'DLL)'.
282 GC0712 01 WS-OS-Type-CD             VALUE 1            PIC 9(1).
283 GC0712 >>END-IF
284 GC0909          88 WS-OS-Windows-BOOL VALUE 1, 5.
285 GC0909          88 WS-OS-Cygwin-BOOL  VALUE 2.
286 GC0712          88 WS-OS-UNIX-BOOL    VALUE 3, 4.
287 GC0712          88 WS-OS-OSX-BOOL     VALUE 4.
288
289          01 WS-OS-Type-FILLER-TXT.
290          05 VALUE 'Windows'           PIC X(14).
291          05 VALUE 'Windows/Cygwin'    PIC X(14).
292          05 VALUE 'UNIX/Linux'       PIC X(14).
=====

```

Line Statement

Page: 9

```

=====
293          05 VALUE 'OSX'                PIC X(14).
294          05 VALUE 'Windows/MinGW'     PIC X(14).
295      01  WS-OS-Types-TXT REDEFINES WS-OS-Type-FILLER-TXT.
296          05 WS-OS-Type-TXT            OCCURS 5 TIMES
297                                          PIC X(14).
298
299      01  WS-Output-Msg-TXT              PIC X(80).
300
301      01  WS-Path-Delimiter-CHR         PIC X(1).
302
303      01  WS-Prog-Extension-TXT         PIC X(256).
304
305      01  WS-Prog-Folder-TXT            PIC X(256).
306
307 GC0712 01  WS-Prog-File-Name-TXT.
308 GC0712 05  WS-PFN-CHR                  OCCURS 256 TIMES
309 GC0712 PIC X(1).
310
311 GC0712 01  WS-Pgm-Nm-TXT               PIC X(31).
312
313      01  WS-Runtime-Switches-TXT.
314          05 WS-RS-Compile-OK-CHR       PIC X(1).
315          88 WS-RS-Compile-OK-BOOL      VALUE 'Y'.
316 GC0909 88 WS-RS-Compile-OK-Warn-BOOL  VALUE 'W'.
317          88 WS-RS-Compile-Failed-BOOL  VALUE 'N'.
318 GC0609 05  WS-RS-Complete-CHR         PIC X(1).
319 GC0609 88  WS-RS-Complete-BOOL        VALUE 'Y'.
320 GC0609 88  WS-RS-Not-Complete-BOOL    VALUE 'N'.
321 GC0712 05  WS-RS-Quote-CHR            PIC X(1).
322 GC0712 88  WS-RS-Double-Quote-Used-BOOL VALUE 'Y' FALSE 'N'.
323 GC0809 05  WS-RS-IDENT-DIV-CHR        PIC X(1).
324 GC0809 88  WS-RS-1st-Prog-Complete-BOOL VALUE 'Y'.
325 GC0809 88  WS-RS-More-To-1st-Prog-BOOL VALUE 'N'.
326          05  WS-RS-No-Switch-Chgs-CHR  PIC X(1).
327          88  WS-RS-No-Switch-Changes-BOOL VALUE 'Y'.
328          88  WS-RS-Switch-Changes-BOOL VALUE 'N'.
329 GC0709 05  WS-RS-Output-File-Busy-CHR  PIC X(1).
330 GC0709 88  WS-RS-Output-File-Busy-BOOL VALUE 'Y'.
331 GC0709 88  WS-RS-Output-File-Avail-BOOL VALUE 'N'.
332 GC0809 05  WS-RS-Source-Record-Type-CHR PIC X(1).
333 GC0809 88  WS-RS-Source-Rec-Linkage-BOOL VALUE 'L'.
334 GC0809 88  WS-RS-Source-Rec-Ident-BOOL VALUE 'I'.
335 GC0712 88  WS-RS-Source-Rec-Ignored-BOOL VALUE ' '.
336          05  WS-RS-Switch-Error-CHR    PIC X(1).
337          88  WS-RS-Switch-Is-Bad-BOOL  VALUE 'Y'.
338          88  WS-RS-Switch-Is-Good-BOOL VALUE 'N'.
339
340      01  WS-Tally-QTY                    USAGE BINARY-LONG.
=====

```

Line Statement

Page: 10

```

=====
341 /
342 SCREEN SECTION.
343 *>
344 *> Here is the layout of the GCic screen.
345 *>
346 *> The sample screen below shows how the screen would look if the LINEDRAW
347 *> configuration setting is set to a value of 2
348 *>
349 *> The following sample screen layout shows how the screen looks with line-drawing
350 *> characters disabled.
351 *>
352 *>
353 *> 1 2 3 4 5 6 7 8
354 *> 1234567890123456789012345678901234567890123456789012345678901234567890
355 GC0712*> GCic (2011/07/11 08:52) - GNU COBOL V2.0 11FEB2012 Interactive Compilation 01
356 GC0712*>+-----+ 02
357 GC0712*> | Folder: E:\GNU COBOL\Samples | 03
358 GC0712*> | Filename: GCic.cbl | 04
359 GC0712*>+-----+ 05
360 GC0712*> Set/Clr Switches Via F1-F9; Set Config Via F12; ENTER Key Compiles; ESC Quits 06
361 GC0712*>+-----+ 07
362 GC0712*> | F1 Assume WITH DEBUGGING MODE F6 "FUNCTION" Is Optional | Current | 08
363 GC0712*> | F2 Procedure+Statement Trace F7 Enable All Warnings | Config: | 09
364 GC0712*> | F3 Make A Library (DLL) F8 Source Is Free-Format | @@@@ | 10
365 GC0712*> | F4 Execute If Compilation OK F9 No COMP/BINARY Truncation | | 11
366 GC0712*> | F5 >Produce Full Listing | | 12
367 GC0712*>+-----+ 13
368 GC0712*> Extra "cobc" Switches, If Any ("-save-temps=xxx" Prevents Listings): 14
369 GC0712*>+-----+ 15
370 GC0712*> | _____ | 16
371 GC0712*> | _____ | 17
372 GC0712*>+-----+ 18
373 GC0712*> Program Execution Arguments, If Any: 19
374 GC0712*>+-----+ 20
375 GC0712*> | _____ | 21
376 GC0712*> | _____ | 22
377 GC0712*>+-----+ 23
378 GC0712*> GCic Copyright (C) 2009 - 2013, Gary L. Cutler, GPL 24
379 *>
380 *> 1234567890123456789012345678901234567890123456789012345678901234567890
381 *>
382 *> 1 2 3 4 5 6 7 8
383 *>
384 *> If this program is run on Windows, it must run with codepage 437 activated to
385 *> display the line-drawing characters. With a native Windows build or a
386 *> Windows/MinGW build, one could use the command 'chcp 437' to set that codepage
387 *> for display within a Windows console window (that should be the default, though).
388 *> With a Windows/Cygwin build, set the environment variable CYGWIN to a value of
389 *> 'codepage:oem' (this cannot be done from within the program though - you will
390 *> have to use the 'Computer/Advanced System Settings/Environment Variables' (Vista or
391 *> Windows 7) function to define the variable. XP Users: use 'My Computer/Properties/
392 *> Advanced/Environment Variables'.
393 *>
394 *> OSX users may use line drawing characters in this and any GNU COBOL program simply
395 *> by setting their 'terminal' application's font to "Lucida Console".

```


Line Statement

```

=====
395      *>
396      >>IF LINEDRAW IS EQUAL TO 0
397          78 LD-UL-Corner          VALUE ' '.
398          78 LD-LL-Corner          VALUE ' '.
399          78 LD-UR-Corner          VALUE ' '.
400          78 LD-LR-Corner          VALUE ' '.
401          78 LD-Upper-T           VALUE ' '.
402          78 LD-Lower-T           VALUE ' '.
403          78 LD-Horiz-Line        VALUE ' '.
404          78 LD-Vert-Line         VALUE ' '.
405      >>ELIF LINEDRAW IS EQUAL TO 1
406          78 LD-UL-Corner          VALUE X'DA'.
407          78 LD-LL-Corner          VALUE X'C0'.
408          78 LD-UR-Corner          VALUE X'BF'.
409          78 LD-LR-Corner          VALUE X'D9'.
410          78 LD-Upper-T           VALUE X'C2'.
411          78 LD-Lower-T           VALUE X'C1'.
412          78 LD-Horiz-Line        VALUE X'C4'.
413          78 LD-Vert-Line         VALUE X'B3'.
414      >>ELSE
415          78 LD-UL-Corner          VALUE '+'.
416          78 LD-LL-Corner          VALUE '+'.
417          78 LD-UR-Corner          VALUE '+'.
418          78 LD-LR-Corner          VALUE '+'.
419          78 LD-Upper-T           VALUE '+'.
420          78 LD-Lower-T           VALUE '+'.
421          78 LD-Horiz-Line        VALUE '-'.
422          78 LD-Vert-Line         VALUE '|'.
423      >>END-IF
424
425      01 S-Blank-SCR LINE 1 COLUMN 1 BLANK SCREEN.
426
427      01 S-Switches-SCR BACKGROUND-COLOR COB-COLOR-BLACK
428                          FOREGROUND-COLOR COB-COLOR-WHITE AUTO.
429      *>
430      *> GENERAL SCREEN FRAMEWORK
431      *>
432          03 BACKGROUND-COLOR COB-COLOR-BLACK
433          FOREGROUND-COLOR COB-COLOR-GREEN HIGHLIGHT.
434 GC0712      05 LINE 02 COL 01          VALUE LD-UL-Corner.
435 GC0712      05          COL 02 PIC X(78) FROM WS-Horizontal-Line-TXT.
436          05          COL 80          VALUE LD-UR-Corner.
437
438 GC0712      05 LINE 03 COL 01          VALUE LD-Vert-Line.
439          05          COL 80          VALUE LD-Vert-Line.
440
441 GC0712      05 LINE 04 COL 01          VALUE LD-Vert-Line.
442          05          COL 80          VALUE LD-Vert-Line.
443
444 GC0712      05 LINE 05 COL 01          VALUE LD-LL-Corner.
445 GC0712      05          COL 02 PIC X(78) FROM WS-Horizontal-Line-TXT.
446          05          COL 80          VALUE LD-LR-Corner.
447
448 GC0712      05 LINE 07 COL 01          VALUE LD-UL-Corner.

```

Line Statement

Page: 12

```

=====
449 GC0712      05          COL 02 PIC X(65) FROM WS-Horizontal-Line-TXT.
450 GC0712      05          COL 67          VALUE LD-Upper-T.
451 GC0712      05          COL 68 PIC X(12) FROM WS-Horizontal-Line-TXT.
452          05          COL 80          VALUE LD-UR-Corner.
453
454 GC0712      05 LINE 08 COL 01          VALUE LD-Vert-Line.
455 GC0712      05          COL 67          VALUE LD-Vert-Line.
456          05          COL 80          VALUE LD-Vert-Line.
457
458 GC0712      05 LINE 09 COL 01          VALUE LD-Vert-Line.
459 GC0712      05          COL 67          VALUE LD-Vert-Line.
460          05          COL 80          VALUE LD-Vert-Line.
461
462 GC0712      05 LINE 10 COL 01          VALUE LD-Vert-Line.
463 GC0712      05          COL 67          VALUE LD-Vert-Line.
464          05          COL 80          VALUE LD-Vert-Line.
465
466 GC0712      05 LINE 11 COL 01          VALUE LD-Vert-Line.
467 GC0712      05          COL 67          VALUE LD-Vert-Line.
468          05          COL 80          VALUE LD-Vert-Line.
469
470 GC0712      05 LINE 12 COL 01          VALUE LD-Vert-Line.
471 GC0712      05          COL 67          VALUE LD-Vert-Line.
472          05          COL 80          VALUE LD-Vert-Line.
473
474 GC0712      05 LINE 13 COL 01          VALUE LD-LL-Corner.
475 GC0712      05          COL 02 PIC X(65) FROM WS-Horizontal-Line-TXT.
476 GC0712      05          COL 67          VALUE LD-Lower-T.
477 GC0712      05          COL 68 PIC X(12) FROM WS-Horizontal-Line-TXT.
478          05          COL 80          VALUE LD-LR-Corner.
479
480 GC0712      05 LINE 15 COL 01          VALUE LD-UL-Corner.
481 GC0712      05          COL 02 PIC X(78) FROM WS-Horizontal-Line-TXT.
482          05          COL 80          VALUE LD-UR-Corner.
483
484 GC0712      05 LINE 16 COL 01          VALUE LD-Vert-Line.
485          05          COL 80          VALUE LD-Vert-Line.
486
487 GC0712      05 LINE 17 COL 01          VALUE LD-Vert-Line.
488          05          COL 80          VALUE LD-Vert-Line.
489
490 GC0712      05 LINE 18 COL 01          VALUE LD-LL-Corner.
491 GC0712      05          COL 02 PIC X(78) FROM WS-Horizontal-Line-TXT.
492          05          COL 80          VALUE LD-LR-Corner.
493
494 GC0712      05 LINE 20 COL 01          VALUE LD-UL-Corner.
495 GC0712      05          COL 02 PIC X(78) FROM WS-Horizontal-Line-TXT.
496          05          COL 80          VALUE LD-UR-Corner.
497
498 GC0712      05 LINE 21 COL 01          VALUE LD-Vert-Line.
499          05          COL 80          VALUE LD-Vert-Line.
500
501 GC0712      05 LINE 22 COL 01          VALUE LD-Vert-Line.
502          05          COL 80          VALUE LD-Vert-Line.
=====

```

Line Statement

Page: 13

```

=====
503
504 GC0712      05 LINE 23 COL 01          VALUE LD-LL-Corner.
505 GC0712      05          COL 02 PIC X(78) FROM WS-Horizontal-Line-TXT.
506            05          COL 80          VALUE LD-LR-Corner.
507            *>
508            *> TOP AND BOTTOM LINES
509            *>
510 GC0712      03 BACKGROUND-COLOR COB-COLOR-BLUE
511            FOREGROUND-COLOR COB-COLOR-WHITE HIGHLIGHT.
512 GC0410      05 LINE 01 COL 01 VALUE ' GCic ('.
513 GC0410      05          COL 08 PIC X(16) FROM WS-OC-Compile-DT.
514 GC0711      05          COL 24 VALUE ') GNU COBOL 2.0 11FEB2012 ' &
515 GC0410            'Interactive Compilation ' '.
516 GC0712      03 BACKGROUND-COLOR COB-COLOR-RED BLINK
517 GC0712      FOREGROUND-COLOR COB-COLOR-WHITE HIGHLIGHT.
518 GC0712      05 LINE 24 COL 01 PIC X(80) FROM WS-Output-Msg-TXT.
519            *>
520            *> LABELS
521            *>
522            03 BACKGROUND-COLOR COB-COLOR-BLACK
523            FOREGROUND-COLOR COB-COLOR-CYAN HIGHLIGHT.
524 GC0712      05 LINE 06 COL 02 VALUE 'Set/Clr Switches Via F1-F9; ' &
525 GC0712            'Set Config Via F12; Enter Key ' &
526 GC0712            'Compiles; Esc Quits'.
527 GC0712      05 LINE 14 COL 02 VALUE 'Extra "cobc" Switches, If Any ' &
528 GC0712            '("-save-temps=xxx" Prevents ' &
529 GC0712            'Listings):'.
530 GC0712      05 LINE 19 COL 02 VALUE 'Program Execution Arguments, ' &
531 GC0712            'If Any:'.
532 GC0712      03 BACKGROUND-COLOR COB-COLOR-BLACK
533 GC0712      FOREGROUND-COLOR COB-COLOR-WHITE HIGHLIGHT.
534 GC0712      05 LINE 06 COL 23 VALUE 'F1'.
535 GC0712      05          COL 26 VALUE 'F9'.
536 GC0712      05          COL 45 VALUE 'F12'.
537 GC0712      05          COL 50 VALUE 'ENTER'.
538 GC0712      05          COL 70 VALUE 'ESC'.
539            *>
540            *> TOP SECTION BACKGROUND
541            *>
542            03 BACKGROUND-COLOR COB-COLOR-BLACK
543            FOREGROUND-COLOR COB-COLOR-WHITE HIGHLIGHT.
544 GC0712      05 LINE 03 COL 62 VALUE 'Enter'.
545 GC0712      05 LINE 04 COL 62 VALUE 'Esc'.
546
547            03 BACKGROUND-COLOR COB-COLOR-BLACK
548            FOREGROUND-COLOR COB-COLOR-GREEN HIGHLIGHT.
549 GC0712      05 LINE 04 COL 03 VALUE 'Folder: '.
550 GC0712      05 LINE 03 COL 03 VALUE 'Filename: '.
551
552 GC0712      05 LINE 03 COL 67 VALUE ': Compile '.
553 GC0712      05 LINE 04 COL 65 VALUE ': Quit '.
554            *>
555            *> TOP SECTION PROGRAM INFO
556            *>

```

Line Statement

```

=====
557          03 BACKGROUND-COLOR COB-COLOR-BLACK
558          FOREGROUND-COLOR COB-COLOR-WHITE HIGHLIGHT.
559 GC0712    05 LINE 03 COL 13 PIC X(66) FROM WS-Prog-File-Name-TXT.
560 GC0712    05 LINE 04 COL 13 PIC X(66) FROM WS-Prog-Folder-TXT.
561          *>
562          *> MIDDLE LEFT SECTION F-KEYS
563          *>
564          03 BACKGROUND-COLOR COB-COLOR-BLACK
565          FOREGROUND-COLOR COB-COLOR-WHITE HIGHLIGHT.
566 GC0712    05 LINE 08 COL 03 VALUE 'F1'.
567 GC0712    05 LINE 09 COL 03 VALUE 'F2'.
568 GC0712    05 LINE 10 COL 03 VALUE 'F3'.
569 GC0712    05 LINE 11 COL 03 VALUE 'F4'.
570 GC0712    05 LINE 12 COL 03 VALUE 'F5'.
571
572 GC0712    05 LINE 08 COL 35 VALUE 'F6'.
573 GC0712    05 LINE 09 COL 35 VALUE 'F7'.
574 GC0712    05 LINE 10 COL 35 VALUE 'F8'.
575 GC0712    05 LINE 11 COL 35 VALUE 'F9'.
576          *>
577          *> MIDDLE LEFT SECTION SWITCHES
578          *>
579          03 BACKGROUND-COLOR COB-COLOR-BLACK
580          FOREGROUND-COLOR COB-COLOR-RED HIGHLIGHT.
581 GC0712    05 LINE 08 COL 06 PIC X(1) FROM WS-CS-DEBUG-CHR.
582 GC0712    05 LINE 09 COL 06 PIC X(1) FROM WS-CS-TRACEALL-CHR.
583 GC0712    05 LINE 10 COL 06 PIC X(1) FROM WS-CS-LIBRARY-CHR.
584 GC0712    05 LINE 11 COL 06 PIC X(1) FROM WS-CS-EXECUTE-CHR.
585 GC0712    05 LINE 12 COL 06 PIC X(1) FROM WS-CS-LISTING-CHR.
586
587 GC0712    05 LINE 08 COL 38 PIC X(1) FROM WS-CS-NOFUNC-CHR.
588 GC0712    05 LINE 09 COL 38 PIC X(1) FROM WS-CS-WARNALL-CHR.
589 GC0712    05 LINE 10 COL 38 PIC X(1) FROM WS-CS-FREE-CHR.
590 GC0712    05 LINE 11 COL 38 PIC X(1) FROM WS-CS-NOTRUNC-CHR.
591          *>
592          *> MIDDLE LEFT SECTION BACKGROUND
593          *>
594          03 BACKGROUND-COLOR COB-COLOR-BLACK
595          FOREGROUND-COLOR COB-COLOR-GREEN HIGHLIGHT.
596 GC0712    05 LINE 08 COL 07 VALUE 'Assume WITH DEBUGGING MODE'.
597 GC0712    05 LINE 09 COL 07 VALUE 'Procedure+Statement Trace '.
598 GC0712    05 LINE 10 COL 07 VALUE 'Make a Library ('.
599 GC0712    05          COL 23 VALUE WS-OS-Lib-Type-CONST.
600 GC0712    05 LINE 11 COL 07 VALUE 'Execute If Compilation OK '.
601 GC0712    05 LINE 12 COL 07 VALUE 'Produce Full Listing '.
602
603 GC0712    05 LINE 08 COL 39 VALUE '"FUNCTION" Is Optional '.
604 GC0712    05 LINE 09 COL 39 VALUE 'Enable All Warnings '.
605 GC0712    05 LINE 10 COL 39 VALUE 'Source Is Free-Format '.
606 GC0712    05 LINE 11 COL 39 VALUE 'No COMP/BINARY Truncation '.
607          *>
608          *> MIDDLE RIGHT SECTION Text
609          *>
610          03 BACKGROUND-COLOR COB-COLOR-BLACK

```

Line Statement

Page: 15

```
=====
611          FOREGROUND-COLOR COB-COLOR-GREEN HIGHLIGHT.
612 GC0712    05 LINE 08 COL 69 VALUE 'Current'.
613 GC0712    05 LINE 09 COL 69 VALUE 'Config:'.
614          *>
615          *> MIDDLE RIGHT SECTION CONFIG FILE
616          *>
617          03 BACKGROUND-COLOR COB-COLOR-BLACK
618          FOREGROUND-COLOR COB-COLOR-WHITE HIGHLIGHT.
619 GC0712    05 LINE 10 COL 69 PIC X(10)
620 GC0712    FROM WS-CS-Filename-TXT (WS-CS-Config-NUM).
621          *>
622          *> FREE-FORM OPTIONS FIELDS
623          *>
624          03 BACKGROUND-COLOR COB-COLOR-BLACK
625          FOREGROUND-COLOR COB-COLOR-WHITE HIGHLIGHT.
626 GC0712    05 LINE 16 COL 03 PIC X(76) USING WS-CS-Extra-H1-TXT.
627 GC0712    05 LINE 17 COL 03 PIC X(76) USING WS-CS-Extra-H2-TXT.
628 GC0712    05 LINE 21 COL 03 PIC X(76) USING WS-CS-Arg-H1-TXT.
629 GC0712    05 LINE 22 COL 03 PIC X(76) USING WS-CS-Arg-H2-TXT.
```

Line Statement

Page: 16

```

=====
630 /
631 PROCEDURE DIVISION.
632 *>*****
633 *> Legend to procedure names: **
634 *> **
635 *> 00x-xxx All MAIN driver procedures **
636 *> 0xx-xxx All GLOBAL UTILITY procedures **
637 *> 1xx-xxx All INITIALIZATION procedures **
638 *> 2xx-xxx All CORE PROCESSING procedures **
639 *> 9xx-xxx All TERMINATION procedures **
640 *>*****
641 DECLARATIVES.
642 000-File-Error SECTION.
643 USE AFTER STANDARD ERROR PROCEDURE ON F-Source-Code-FILE.
644 COPY FileStat-Msgs
645 REPLACING STATUS BY WS-FSM-Status-CD
646 MSG BY WS-FSM-Msg-TXT.
EVALUATE WS-FSM-Status-CD
WHEN 00 MOVE 'SUCCESS ' TO WS-FSM-Msg-TXT
WHEN 02 MOVE 'SUCCESS DUPLICATE ' TO WS-FSM-Msg-TXT
WHEN 04 MOVE 'SUCCESS INCOMPLETE ' TO WS-FSM-Msg-TXT
WHEN 05 MOVE 'SUCCESS OPTIONAL ' TO WS-FSM-Msg-TXT
WHEN 07 MOVE 'SUCCESS NO UNIT ' TO WS-FSM-Msg-TXT
WHEN 10 MOVE 'END OF FILE ' TO WS-FSM-Msg-TXT
WHEN 14 MOVE 'OUT OF KEY RANGE ' TO WS-FSM-Msg-TXT
WHEN 21 MOVE 'KEY INVALID ' TO WS-FSM-Msg-TXT
WHEN 22 MOVE 'KEY EXISTS ' TO WS-FSM-Msg-TXT
WHEN 23 MOVE 'KEY NOT EXISTS ' TO WS-FSM-Msg-TXT
WHEN 30 MOVE 'PERMANENT ERROR ' TO WS-FSM-Msg-TXT
WHEN 31 MOVE 'INCONSISTENT FILENAME ' TO WS-FSM-Msg-TXT
WHEN 34 MOVE 'BOUNDARY VIOLATION ' TO WS-FSM-Msg-TXT
WHEN 35 MOVE 'FILE NOT FOUND ' TO WS-FSM-Msg-TXT
WHEN 37 MOVE 'PERMISSION DENIED ' TO WS-FSM-Msg-TXT
WHEN 38 MOVE 'CLOSED WITH LOCK ' TO WS-FSM-Msg-TXT
WHEN 39 MOVE 'CONFLICT ATTRIBUTE ' TO WS-FSM-Msg-TXT
WHEN 41 MOVE 'ALREADY OPEN ' TO WS-FSM-Msg-TXT
WHEN 42 MOVE 'NOT OPEN ' TO WS-FSM-Msg-TXT
WHEN 43 MOVE 'READ NOT DONE ' TO WS-FSM-Msg-TXT
WHEN 44 MOVE 'RECORD OVERFLOW ' TO WS-FSM-Msg-TXT
WHEN 46 MOVE 'READ ERROR ' TO WS-FSM-Msg-TXT
WHEN 47 MOVE 'INPUT DENIED ' TO WS-FSM-Msg-TXT
WHEN 48 MOVE 'OUTPUT DENIED ' TO WS-FSM-Msg-TXT
WHEN 49 MOVE 'I/O DENIED ' TO WS-FSM-Msg-TXT
WHEN 51 MOVE 'RECORD LOCKED ' TO WS-FSM-Msg-TXT
WHEN 52 MOVE 'END-OF-PAGE ' TO WS-FSM-Msg-TXT
WHEN 57 MOVE 'I/O LINAGE ' TO WS-FSM-Msg-TXT
WHEN 61 MOVE 'FILE SHARING FAILURE ' TO WS-FSM-Msg-TXT
WHEN 91 MOVE 'FILE NOT AVAILABLE ' TO WS-FSM-Msg-TXT
END-EVALUATE.
647 MOVE SPACES TO WS-Output-Msg-TXT
648 IF WS-FSM-Status-CD = 35
649 DISPLAY
650 'File not found: ''
651 TRIM(WS-File-Name-TXT,TRAILING)

```

Line Statement

Page: 17

```
=====
652          '""
653          ELSE
654              DISPLAY
655              'Error accessing file: ""
656              TRIM(WS-File-Name-TXT,TRAILING)
657              '""
658          END-IF
659          GOBACK
660          .
661      END DECLARATIVES.
662
663      000-Main SECTION.
664          PERFORM 100-Initialization
665      GC0609      SET WS-RS-Not-Complete-BOOL TO TRUE
666      GC0609      PERFORM UNTIL WS-RS-Complete-BOOL
667      GC0609          PERFORM 200-Let-User-Set-Switches
668      GC0609          PERFORM 210-Run-Compiler
669      GC0410          IF (WS-RS-Compile-OK-BOOL OR WS-RS-Compile-OK-Warn-BOOL)
670      GC0712          AND (WS-CS-LISTING-CHR > SPACE)
671      GC0712              DISPLAY S-Blank-SCR
672      GC0410              PERFORM 220-Make-Listing
673      GC0410          END-IF
674      GC0709          IF (WS-CS-EXECUTE-CHR NOT = SPACES)
675      GC0709          AND (WS-RS-Output-File-Avail-BOOL)
676      GC0609              PERFORM 230-Run-Program
677      GC0609          END-IF
678      GC0712          PERFORM 250-Autoload-Listing
679      GC0609      END-PERFORM
680          PERFORM 900-Terminate
681      * -- Control will NOT return
682          .
683      * -- Control will NOT return
```

Line Statement

Page: 18

```

=====
684 /
685 *>*****
686 *> Perform all program-wide initialization operations **
687 *>*****
688 100-Initialization SECTION.
689 *>*****
690 *> Make sure full screen-handling is in effect **
691 *>*****
692 SET ENVIRONMENT 'COB_SCREEN_EXCEPTIONS' TO 'Y'
693 SET ENVIRONMENT 'COB_SCREEN_ESC' TO 'Y'
694 *>*****
695 *> Get GCic Compilation Date/Time **
696 *>*****
697 MOVE WHEN-COMPILED (1:12) TO WS-OC-Compile-DT
698 INSPECT WS-OC-Compile-DT
699 REPLACING ALL '/' BY ':'
700 AFTER INITIAL SPACE
701 *>*****
702 *> Convert WS-CS-All-Switches-TXT to Needed Alphanumeric Values **
703 *>*****
704 INSPECT WS-CS-All-Switches-TXT
705 REPLACING ALL '0' BY SPACE
706 ALL '1' BY SELCHAR
707 *>*****
708 *> Process filename (the only command-line argument) **
709 *>*****
710 GC0712 ACCEPT WS-Cmd-Args-TXT FROM COMMAND-LINE
711 GC0712 MOVE 1 TO WS-Cmd-SUB
712 GC0712 IF WS-Cmd-Args-TXT(WS-Cmd-SUB:1) = ''' OR '''
713 GC0712 MOVE WS-Cmd-Args-TXT(WS-Cmd-SUB:1)
714 GC0712 TO WS-Cmd-End-Quote-CHR
715 GC0712 ADD 1 TO WS-Cmd-SUB
716 GC0712 UNSTRING WS-Cmd-Args-TXT
717 GC0712 DELIMITED BY WS-Cmd-End-Quote-CHR
718 GC0712 INTO WS-File-Name-TXT
719 GC0712 WITH POINTER WS-Cmd-SUB
720 GC0712 ELSE
721 GC0712 UNSTRING WS-Cmd-Args-TXT
722 GC0712 DELIMITED BY ALL SPACES
723 GC0712 INTO WS-File-Name-TXT
724 GC0712 WITH POINTER WS-Cmd-SUB
725 GC0712 END-IF
726 IF WS-File-Name-TXT = SPACES
727 GC0712 DISPLAY 'No program filename was specified'
728 PERFORM 900-Terminate
729 * ----- Control will NOT return
730 END-IF
731 *>*****
732 *> Determine if 'Make A Library' feature should be forced 'ON' **
733 *>*****
734 PERFORM 240-Find-LINKAGE-SECTION
735 *>*****
736 *> Split 'WS-File-Name-TXT' into 'WS-Prog-Folder-TXT' and **
737 *> 'WS-Prog-File-Name-TXT' **

```


Line Statement

Page: 19

```

=====
738      *>*****
739 GC0909      IF WS-OS-Cygwin-BOOL AND WS-File-Name-TXT (2:1) = ':'
740 GC0712          MOVE '\' TO WS-OS-Dir-CHR
741 GC0909      END-IF
742 GC0712          MOVE LENGTH(WS-File-Name-TXT) TO WS-I-SUB
743 GC0712          PERFORM UNTIL WS-I-SUB = 0
744 GC0712              OR WS-FN-CHR (WS-I-SUB) = WS-OS-Dir-CHR
745                  SUBTRACT 1 FROM WS-I-SUB
746          END-PERFORM
747          IF WS-I-SUB = 0
748              MOVE SPACES          TO WS-Prog-Folder-TXT
749              MOVE WS-File-Name-TXT TO WS-Prog-File-Name-TXT
750          ELSE
751              MOVE '*' TO WS-FN-CHR (WS-I-SUB)
752              UNSTRING WS-File-Name-TXT DELIMITED BY '*'
753                  INTO WS-Prog-Folder-TXT
754                      WS-Prog-File-Name-TXT
755 GC0712          MOVE WS-OS-Dir-CHR TO WS-FN-CHR (WS-I-SUB)
756          END-IF
757          IF WS-Prog-Folder-TXT = SPACES
758              ACCEPT WS-Prog-Folder-TXT FROM ENVIRONMENT 'CD'
759 GC0909          ELSE
760 GC0909              CALL 'CBL_CHANGE_DIR'
761 GC0909                  USING TRIM(WS-Prog-Folder-TXT,TRAILING)
762          END-IF
763 GC0909          IF WS-OS-Cygwin-BOOL AND WS-File-Name-TXT (2:1) = ':'
764 GC0712              MOVE '/' TO WS-OS-Dir-CHR
765 GC0909          END-IF
766      *>*****
767      *> Split 'WS-Prog-File-Name-TXT' into 'WS-Pgm-Nm-TXT' &      **
768      *> 'WS-Prog-Extension-TXT'                                     **
769      *>*****
770 GC0712          MOVE LENGTH(WS-Prog-File-Name-TXT) TO WS-I-SUB
771 GC0712          PERFORM UNTIL WS-I-SUB = 0
772 GC0712              OR WS-PFN-CHR (WS-I-SUB) = '.'
773 GC0712                  SUBTRACT 1 FROM WS-I-SUB
774 GC0712          END-PERFORM
775 GC0712          IF WS-I-SUB = 0
776 GC0712              MOVE WS-Prog-File-Name-TXT TO WS-Pgm-Nm-TXT
777 GC0712              MOVE SPACES          TO WS-Prog-Extension-TXT
778 GC0712          ELSE
779 GC0712              MOVE '*' TO WS-PFN-CHR (WS-I-SUB)
780 GC0712              UNSTRING WS-Prog-File-Name-TXT DELIMITED BY '*'
781 GC0712                  INTO WS-Pgm-Nm-TXT
782 GC0712                      WS-Prog-Extension-TXT
783 GC0712              MOVE '.' TO WS-PFN-CHR (WS-I-SUB)
784 GC0712          END-IF
785      *>*****
786      *> Build initial Line 24 Message                                **
787      *>*****
788 GC0909          MOVE ALL LD-Horiz-Line TO WS-Horizontal-Line-TXT.
789 GC0410          MOVE CONCATENATE(' GCic for '
790 GC0410              TRIM(WS-OS-Type-TXT(WS-OS-Type-CD),Trailing)
791 GC0712              ' Copyright (C) 2009 - 2013, Gary L. '

```

Line Statement

```
=====
792 GC0410                                'Cutler, GPL')
793 GC0410      TO WS-Output-Msg-TXT.
794 GC0909      .
```

Line Statement

Page: 21

```

=====
795 /
796 *>*****
797 *> Show the user the current switch settings and allow them to **
798 *> be changed. **
799 *>*****
800 200-Let-User-Set-Switches SECTION.
801     SET WS-RS-Switch-Changes-BOOL TO TRUE
802     PERFORM UNTIL WS-RS-No-Switch-Changes-BOOL
803         ACCEPT S-Switches-SCR
804         IF COB-CRT-STATUS > 0
805             EVALUATE COB-CRT-STATUS
806                 WHEN COB-SCR-F1
807                     IF WS-CS-DEBUG-CHR = SPACE
808 GC0712                     MOVE SELCHAR TO WS-CS-DEBUG-CHR
809                     ELSE
810                         MOVE ' ' TO WS-CS-DEBUG-CHR
811                     END-IF
812 GC0712                 WHEN COB-SCR-F2
813 GC0712                     IF WS-CS-TRACEALL-CHR = SPACE
814 GC0712                     MOVE SELCHAR TO WS-CS-TRACEALL-CHR
815 GC0712                     ELSE
816 GC0712                     MOVE ' ' TO WS-CS-TRACEALL-CHR
817 GC0712                     END-IF
818                 WHEN COB-SCR-F3
819 GC0712                     IF WS-CS-LIBRARY-CHR = SPACE
820 GC0712                     MOVE SELCHAR TO WS-CS-LIBRARY-CHR
821                     ELSE
822 GC0712                     MOVE ' ' TO WS-CS-LIBRARY-CHR
823                     END-IF
824                 WHEN COB-SCR-F4
825                     IF WS-CS-EXECUTE-CHR = SPACE
826 GC0712                     MOVE SELCHAR TO WS-CS-EXECUTE-CHR
827                     ELSE
828                         MOVE ' ' TO WS-CS-EXECUTE-CHR
829                     END-IF
830 GC0712                 WHEN COB-SCR-F5
831 GC0712                     IF WS-CS-LISTING-CHR = SPACE
832 GC0712                     MOVE SELCHAR TO WS-CS-LISTING-CHR
833 GC0712                     ELSE
834 GC0712                     MOVE ' ' TO WS-CS-LISTING-CHR
835 GC0712                     END-IF
836 GC0712                 WHEN COB-SCR-F6
837 GC0712                     IF WS-CS-NOFUNC-CHR = SPACE
838 GC0712                     MOVE SELCHAR TO WS-CS-NOFUNC-CHR
839 GC0712                     ELSE
840 GC0712                     MOVE ' ' TO WS-CS-NOFUNC-CHR
841 GC0712                     END-IF
842 GC0712                 WHEN COB-SCR-F7
843 GC0712                     IF WS-CS-WARNALL-CHR = SPACE
844 GC0712                     MOVE SELCHAR TO WS-CS-WARNALL-CHR
845 GC0712                     ELSE
846 GC0712                     MOVE ' ' TO WS-CS-WARNALL-CHR
847 GC0712                     END-IF
848 GC0712                 WHEN COB-SCR-F8

```

Line Statement

Page: 22

```
=====
849 GC0712          IF WS-CS-FREE-CHR = SPACE
850 GC0712          MOVE SELCHAR TO WS-CS-FREE-CHR
851 GC0712          ELSE
852 GC0712          MOVE ' ' TO WS-CS-FREE-CHR
853 GC0712          END-IF
854 GC0712        WHEN COB-SCR-F9
855 GC0712          IF WS-CS-NOTRUNC-CHR = SPACE
856 GC0712          MOVE SELCHAR TO WS-CS-NOTRUNC-CHR
857 GC0712          ELSE
858 GC0712          MOVE ' ' TO WS-CS-NOTRUNC-CHR
859 GC0712          END-IF
860 GC0712        WHEN COB-SCR-ESC
861 GC0712          PERFORM 900-Terminate
862 GC0712 * ----- Control will NOT return
863 GC0712          WHEN COB-SCR-F12
864 GC0712            ADD 1 TO WS-CS-Config-NUM
865 GC0712            IF WS-CS-Config-NUM > 7
866 GC0712            MOVE 1 TO WS-CS-Config-NUM
867 GC0712            END-IF
868 GC0712          WHEN OTHER
869 GC0712            MOVE 'An unsupported key was pressed'
870 GC0712            TO WS-Output-Msg-TXT
871 GC0712          END-EVALUATE
872 GC0712        ELSE
873 GC0712          SET WS-RS-No-Switch-Changes-BOOL TO TRUE
874 GC0712        END-IF
875 GC0712      END-PERFORM
876 GC0712      .
=====
```

Line Statement

```

=====
877 /
878 *>*****
879 *> Run the compiler using the switch settings we've prepared. **
880 *>*****
881 210-Run-Compiler SECTION.
882 MOVE SPACES TO WS-Cmd-TXT
883 WS-Cobc-Cmd-TXT
884 WS-Output-Msg-TXT
885 DISPLAY S-Switches-SCR
886 MOVE 1 TO WS-I-SUB
887 GC0712 MOVE LOWER-CASE(WS-CS-Filename-TXT (WS-CS-Config-NUM))
888 GC0712 TO WS-Config-Fn-TXT
889 *>*****
890 *> Build the 'cobc' command **
891 *>*****
892 GC0909 MOVE SPACES TO WS-Cobc-Cmd-TXT
893 GC0909 STRING 'cobc -v -std='
894 GC0909 TRIM(WS-Config-Fn-TXT,TRAILING)
895 GC0909 ' '
896 GC0909 INTO WS-Cobc-Cmd-TXT
897 GC0909 WITH POINTER WS-I-SUB
898 IF WS-CS-LIBRARY-CHR NOT = ' '
899 STRING '-m '
900 DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
901 WITH POINTER WS-I-SUB
902 ELSE
903 STRING '-x '
904 DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
905 WITH POINTER WS-I-SUB
906 END-IF
907 IF WS-CS-DEBUG-CHR NOT = ' '
908 STRING '-fdebugging-line '
909 DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
910 WITH POINTER WS-I-SUB
911 END-IF
912 IF WS-CS-NOTRUNC-CHR NOT = ' '
913 STRING '-fnotrunc '
914 DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
915 WITH POINTER WS-I-SUB
916 END-IF
917 IF WS-CS-TRACEALL-CHR NOT = ' '
918 GC0809 STRING '-ftraceall '
919 DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
920 WITH POINTER WS-I-SUB
921 END-IF
922 GC0712 IF WS-CS-NOFUNC-CHR NOT = ' '
923 GC0712 STRING '-fintrinsic=all '
924 GC0712 DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
925 GC0712 WITH POINTER WS-I-SUB
926 GC0712 END-IF
927 GC0712 IF WS-CS-WARNALL-CHR NOT = ' '
928 GC0712 STRING '-wall '
929 GC0712 DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
930 GC0712 WITH POINTER WS-I-SUB

```

Line Statement

Page: 24

```

=====
931 GC0712     END-IF
932 GC0712     IF WS-CS-FREE-CHR NOT = ' '
933 GC0712         STRING '-free '
934 GC0712             DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
935 GC0712             WITH POINTER WS-I-SUB
936 GC0712     ELSE
937 GC0712         STRING '-fixed '
938 GC0712             DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
939 GC0712             WITH POINTER WS-I-SUB
940 GC0712     END-IF
941
942 GC0712     MOVE 0 TO WS-Tally-QTY
943 GC0712     INSPECT WS-CS-Extra-TXT
944 GC0712         TALLYING WS-Tally-QTY FOR ALL '-save-temps'
945 GC0712     IF WS-CS-LISTING-CHR > SPACE
946 GC0712     AND WS-Tally-QTY > 0
947 GC0712         MOVE SPACE TO WS-CS-LISTING-CHR *> Can't generate listing if -save-temps used
948 GC0712     END-IF
949 GC0712     IF WS-CS-LISTING-CHR > SPACE
950 GC1010         STRING '-save-temps '
951 GC1010             DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
952 GC1010             WITH POINTER WS-I-SUB
953 GC1010     END-IF
954
955 GC0709     IF WS-CS-Extra-TXT > SPACES
956 GC0709         STRING ' '
957 GC0709             TRIM(WS-CS-Extra-TXT,TRAILING)
958 GC0709             ' '
959 GC0709             DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
960 GC0709             WITH POINTER WS-I-SUB
961 GC0709     END-IF
962 GC0909     STRING TRIM(WS-Prog-File-Name-TXT,TRAILING)
963 GC0909         DELIMITED SIZE INTO WS-Cobc-Cmd-TXT
964 GC0909         WITH POINTER WS-I-SUB
965 *>*****
966 *> Prepare the compilation listing file **
967 *>*****
968 GC1113     MOVE CONCATENATE(TRIM(WS-Pgm-Nm-TXT,Trailing),'.gclst')
969 GC0712         TO WS-Listing-Filename-TXT
970 GC0712     CALL 'CBL_DELETE_FILE' USING WS-Listing-Filename-TXT
971 *>*****
972 *> Now execute the 'cobc' command **
973 *>*****
974 GC0410     MOVE ' Compiling...' TO WS-Output-Msg-TXT
975 GC0410     DISPLAY S-Switches-SCR
976 GC0609     SET WS-RS-Output-File-Avail-BOOL TO TRUE
977     MOVE SPACES TO WS-Cmd-TXT
978     STRING TRIM(WS-Cobc-Cmd-TXT,TRAILING)
979 GC0712         ' >' WS-Listing-Filename-TXT
980 GC0712         ' 2>&1'
981     DELIMITED SIZE
982     INTO WS-Cmd-TXT
983 DEBUG D     DISPLAY WS-Cmd-TXT UPON SYSERR
984     CALL 'SYSTEM' USING TRIM(WS-Cmd-TXT,TRAILING)
=====

```

Line Statement

Page: 25

```

=====
 985 GC0712 OPEN EXTEND F-Cobc-Output-FILE
 986 GC0712 WRITE F-Cobc-Output-REC FROM SPACES
 987 GC0712 IF RETURN-CODE = 0
 988 GC0712     SET WS-RS-Compile-OK-BOOL TO TRUE
 989 GC0712     MOVE ' Compilation Was Successful' TO WS-Output-Msg-TXT
 990 GC0712     MOVE CONCATENATE('GNU COBOL',WS-Output-Msg-TXT)
 991 GC0712         TO F-Cobc-Output-REC
 992 GC0712     WRITE F-Cobc-Output-REC
 993 GC0712     SET WS-RS-Complete-BOOL TO TRUE
 994 GC0712 ELSE
 995 GC0712     SET WS-RS-Compile-Failed-BOOL TO TRUE
 996 GC0712     MOVE CONCATENATE(' Compilation Failed - See ',
 997 GC0712         TRIM(WS-Listing-Filename-TXT,Trailing))
 998 GC0712         TO WS-Output-Msg-TXT
 999 GC0712     MOVE 'GNU COBOL Compilation HAS FAILED - See Above'
1000 GC0712         TO F-Cobc-Output-REC
1001 GC0712     WRITE F-Cobc-Output-REC
1002 GC0712 END-IF
1003 GC0712 CLOSE F-Cobc-Output-FILE
1004 GC0712 DISPLAY S-Switches-SCR
1005 GC0712 CALL 'C$SLEEP' USING 2
1006 GC0712 MOVE SPACES TO WS-Output-Msg-TXT
1007 GC0712 IF WS-RS-Compile-Failed-BOOL
1008 GC0712     PERFORM 250-Autoload-Listing
1009 GC0712     PERFORM 900-Terminate
1010 GC0712     *> ----- Control will not return
1011 GC0712     END-IF
1012 GC0712     .
=====

```

Line Statement

Page: 26

```

=====
1013 /
1014 *>*****
1015 *> Generate a source + xref listing using 'LISTING' subroutine **
1016 *>*****
1017 GC0410 220-Make-Listing SECTION.
1018 GC0410 MOVE ' Generating listing...' TO WS-Output-Msg-TXT
1019 GC0410 DISPLAY S-Switches-SCR
1020 GC0410 MOVE 0 TO RETURN-CODE
1021 *>*****
1022 *> Create the listing **
1023 *>*****
1024 GC0410 MOVE SPACES TO WS-Output-Msg-TXT
1025 GC0410 CALL 'LISTING' USING WS-Listing-Filename-TXT
1026 GC0712 WS-File-Name-TXT
1027 GC0712 WS-OS-Type-CD
1028 GC0410 ON EXCEPTION
1029 GC0410 MOVE ' LISTING module is not available'
1030 GC0410 TO WS-Output-Msg-TXT
1031 GC0410 MOVE 1 TO RETURN-CODE
1032 GC0410 END-CALL
1033 GC0410 IF RETURN-CODE = 0
1034 GC0712 MOVE ' Source+Xref listing generated '
1035 GC0712 TO WS-Output-Msg-TXT
1036 GC0410 END-IF
1037 GC0712 DISPLAY S-Switches-SCR
1038 GC0712 CALL 'C$SLEEP' USING 2
1039 GC0712 PERFORM 250-Autoload-Listing
1040 GC0410 .
=====

```


Line Statement

Page: 27

```

=====
1041 /
1042 *>*****
1043 *> Run the compiled program **
1044 *>*****
1045 230-Run-Program SECTION.
1046 GC0909 MOVE SPACES TO WS-Cmd-TXT
1047 GC0909 MOVE 1 TO WS-I-SUB
1048 *>*****
1049 *> If necessary, start with 'cobcrun' command **
1050 *>*****
1051 GC0712 IF WS-CS-LIBRARY-CHR NOT = ' '
1052         STRING 'cobcrun ' DELIMITED SIZE
1053         INTO WS-Cmd-TXT
1054         WITH POINTER WS-I-SUB
1055     END-IF
1056 *>*****
1057 *> Add any necessary path prefix **
1058 *>*****
1059 GC0712 SET WS-RS-Double-Quote-Used-BOOL TO FALSE
1060 IF WS-Prog-Folder-TXT NOT = SPACES
1061 GC0909 IF WS-OS-Cygwin-BOOL AND WS-Prog-Folder-TXT (2:1) = ':'
1062 GC0909     STRING '/cygdrive/'
1063 GC0909     INTO WS-Cmd-TXT
1064 GC0909     WITH POINTER WS-I-SUB
1065 GC0909     STRING LOWER-CASE(WS-Prog-Folder-TXT (1:1))
1066 GC0909     INTO WS-Cmd-TXT
1067 GC0909     WITH POINTER WS-I-SUB
1068 GC0909     PERFORM
1069 GC0909         VARYING WS-J-SUB FROM 3 BY 1
1070 GC0909         UNTIL WS-J-SUB > LENGTH(TRIM(WS-Prog-Folder-TXT))
1071 GC0909         IF WS-Prog-Folder-TXT (WS-J-SUB:1) = '\'
1072 GC0909             STRING '/'
1073 GC0909             INTO WS-Cmd-TXT
1074 GC0909             WITH POINTER WS-I-SUB
1075 GC0909         ELSE
1076 GC0909             STRING WS-Prog-Folder-TXT (WS-J-SUB:1)
1077 GC0909             INTO WS-Cmd-TXT
1078 GC0909             WITH POINTER WS-I-SUB
1079 GC0909         END-IF
1080 GC0909     END-PERFORM
1081 GC0909 ELSE
1082 GC0410     STRING ''' TRIM(WS-Prog-Folder-TXT,TRAILING)
1083 GC0909     INTO WS-Cmd-TXT
1084 GC0909     WITH POINTER WS-I-SUB
1085 GC0712     SET WS-RS-Double-Quote-Used-BOOL TO TRUE
1086 GC0909     END-IF
1087 GC0712     STRING WS-OS-Dir-CHR
1088 GC0909     INTO WS-Cmd-TXT
1089 GC0909     WITH POINTER WS-I-SUB
1090 GC0909 ELSE
1091 GC0909     IF WS-OS-Cygwin-BOOL OR WS-OS-UNIX-BOOL
1092 GC0909         STRING './'
1093 GC0909         INTO WS-Cmd-TXT
1094 GC0909         WITH POINTER WS-I-SUB

```

Line Statement

Page: 28

```

=====
1095 GC0909      END-IF
1096            END-IF
1097            *>*****
1098            *> Insert program filename                **
1099            *>*****
1100 GC0909      STRING TRIM(WS-Pgm-Nm-TXT,TRAILING)
1101 GC0909      INTO WS-Cmd-TXT
1102 GC0909      WITH POINTER WS-I-SUB
1103            *>*****
1104            *> Insert proper extension                **
1105            *>*****
1106 GC0712      IF WS-CS-LIBRARY-CHR = ' '
1107 GC0712      IF WS-OS-Exe-Ext-CONST > ' '
1108 GC0712      STRING WS-OS-Exe-Ext-CONST DELIMITED SPACE
1109 GC0712      INTO WS-Cmd-TXT
1110 GC0712      WITH POINTER WS-I-SUB
1111 GC0712      END-IF
1112 GC0712      ELSE
1113 GC0712      IF WS-OS-Lib-Ext-CONST > ' '
1114 GC0712      STRING WS-OS-Lib-Ext-CONST DELIMITED SPACE
1115 GC0712      INTO WS-Cmd-TXT
1116 GC0712      WITH POINTER WS-I-SUB
1117 GC0712      END-IF
1118 GC0712      END-IF
1119 GC0712      IF WS-RS-Double-Quote-Used-BOOL
1120 GC0712      STRING '""' DELIMITED SIZE
1121 GC0712      INTO WS-Cmd-TXT
1122 GC0712      WITH POINTER WS-I-SUB
1123 GC0712      END-IF
1124            IF WS-CS-Args-TXT NOT = SPACES
1125 GC0809      STRING ' ' TRIM(WS-CS-Args-TXT,TRAILING)
1126            INTO WS-Cmd-TXT
1127            WITH POINTER WS-I-SUB
1128            END-IF
1129            IF WS-OS-Windows-BOOL
1130 GC0712      STRING '&&pause'
1131            INTO WS-Cmd-TXT
1132            WITH POINTER WS-I-SUB
1133            ELSE
1134 GC0712      STRING ';echo "Press ENTER to close...";read'
1135            INTO WS-Cmd-TXT
1136            WITH POINTER WS-I-SUB
1137            END-IF
1138 DEBUG D    DISPLAY WS-Cmd-TXT UPON SYSERR
1139            *>*****
1140            *> Run the program                        **
1141            *>*****
1142 GC0909      DISPLAY S-Blank-SCR
1143            CALL 'SYSTEM' USING TRIM(WS-Cmd-TXT,TRAILING)
1144 GC0712      MOVE SPACES TO WS-Output-Msg-TXT
1145            PERFORM 900-Terminate
1146            * -- Control will NOT return
1147            .
=====

```

Line Statement

Page: 29

```

=====
1148 /
1149 *>*****
1150 *> Determine if the program being compiled is a MAIN program **
1151 *>*****
1152 240-Find-LINKAGE-SECTION SECTION.
1153 OPEN INPUT F-Source-Code-FILE
1154 GC0712 MOVE ' ' TO WS-CS-LIBRARY-CHR
1155 SET WS-RS-More-To-1st-Prog-BOOL TO TRUE
1156 PERFORM UNTIL WS-RS-1st-Prog-Complete-BOOL
1157 READ F-Source-Code-FILE AT END
1158 CLOSE F-Source-Code-FILE
1159 EXIT SECTION
1160 END-READ
1161 GC0712 CALL 'CHECKSRC'
1162 GC0712 USING BY CONTENT F-Source-Code-REC
1163 GC0712 BY REFERENCE WS-RS-Source-Record-Type-CHR
1164 IF WS-RS-Source-Rec-Ident-BOOL
1165 SET WS-RS-1st-Prog-Complete-BOOL TO TRUE
1166 END-IF
1167 END-PERFORM
1168 GC0712 SET WS-RS-Source-Rec-Ignored-BOOL TO TRUE
1169 PERFORM UNTIL WS-RS-Source-Rec-Linkage-BOOL
1170 OR WS-RS-Source-Rec-Ident-BOOL
1171 READ F-Source-Code-FILE AT END
1172 CLOSE F-Source-Code-FILE
1173 EXIT SECTION
1174 END-READ
1175 GC0712 CALL 'CHECKSRC'
1176 GC0712 USING BY CONTENT F-Source-Code-REC
1177 GC0712 BY REFERENCE WS-RS-Source-Record-Type-CHR
1178 END-PERFORM
1179 CLOSE F-Source-Code-FILE
1180 IF WS-RS-Source-Rec-Linkage-BOOL
1181 GC0712 MOVE SELCHAR TO WS-CS-LIBRARY-CHR
1182 END-IF
1183 .
=====

```

Line Statement

```

=====
1184 /
1185 GC0712*>*****
1186 GC0712*> Attempt to open the listing file as a command. This will - **
1187 GC1113*> if the user has associated filetype/extension 'gclst' with **
1188 GC0712*> an application - invoke the appropriate application to **
1189 GC0712*> allow the user to view the listing. **
1190 GC0712*>*****
1191 GC0712 250-Autoload-Listing SECTION.
1192 GC0712 EVALUATE TRUE
1193 GC0712 WHEN WS-OS-Windows-BOOL OR WS-OS-Cygwin-BOOL
1194 GC0712 MOVE SPACES TO WS-Cmd-TXT
1195 GC0712 STRING
1196 GC0712 'cmd /c '
1197 GC0712 TRIM(WS-Listing-Filename-TXT,TRAILING)
1198 GC0712 DELIMITED SIZE INTO WS-Cmd-TXT
1199 GC0712 CALL 'SYSTEM' USING TRIM(WS-Cmd-TXT,TRAILING)
1200 GC0712 WHEN WS-OS-OSX-BOOL
1201 GC0712 MOVE SPACES TO WS-Cmd-TXT
1202 GC0712 STRING
1203 GC0712 'open -t '
1204 GC0712 TRIM(WS-Listing-Filename-TXT,TRAILING)
1205 GC0712 DELIMITED SIZE INTO WS-Cmd-TXT
1206 GC0712 CALL 'SYSTEM' USING TRIM(WS-Cmd-TXT,TRAILING)
1207 GC0712 END-EVALUATE
1208 GC0712*>*****
1209 GC0712*> ** Since we had to do our own '-save-temps' when we **
1210 GC0712*> ** compiled (in order to generate the cross-reference **
1211 GC0712*> ** listing) we now need to clean up after ourselves. **
1212 GC0712*>*****
1213 GC1112 DISPLAY S-Blank-SCR
1214 GC0712 IF WS-OS-Windows-BOOL
1215 GC0712 MOVE CONCATENATE('del ',TRIM(WS-Pgm-Nm-TXT,TRAILING))
1216 GC0712 TO WS-Cmd-TXT
1217 GC0712 ELSE
1218 GC0712 MOVE CONCATENATE('rm ',TRIM(WS-Pgm-Nm-TXT,TRAILING))
1219 GC0712 TO WS-Cmd-TXT
1220 GC0712 END-IF
1221 GC0712 CALL 'SYSTEM'
1222 GC0712 USING CONCATENATE(TRIM(WS-Cmd-TXT,TRAILING),'.c')
1223 GC0712 CALL 'SYSTEM'
1224 GC0712 USING CONCATENATE(TRIM(WS-Cmd-TXT,TRAILING),'.c.h')
1225 GC0712 CALL 'SYSTEM'
1226 GC0712 USING CONCATENATE(TRIM(WS-Cmd-TXT,TRAILING),'.c.l*.h')
1227 GC0712 CALL 'SYSTEM'
1228 GC0712 USING CONCATENATE(TRIM(WS-Cmd-TXT,TRAILING),'.i')
1229 GC0712 CALL 'SYSTEM'
1230 GC0712 USING CONCATENATE(TRIM(WS-Cmd-TXT,TRAILING),'.o')
1231
1232 GC0712 .
=====

```

Line Statement

Page: 31

```
=====
1233 /
1234 *>*****
1235 *> Display a message and halt the program **
1236 *>*****
1237 900-Terminate SECTION.
1238 GC0909 IF WS-Output-Msg-TXT > SPACES
1239 GC0909 DISPLAY S-Switches-SCR
1240 GC0909 CALL 'C$SLEEP' USING 2
1241 GC0909 END-IF
1242 DISPLAY S-Blank-SCR
1243 STOP RUN
1244 .
1245
1246 END PROGRAM GCic.
```

Line Statement

```

=====
1247 /
1248 IDENTIFICATION DIVISION.
1249 PROGRAM-ID. CHECKSRC.
1250 *>*****
1251 *> This subprogram will scan a line of source code it is given **
1252 *> looking for 'LINKAGE SECTION' or 'IDENTIFICATION DIVISION'. **
1253 *> **
1254 *> ****NOTE**** ****NOTE**** ****NOTE**** ****NOTE*** **
1255 *> **
1256 *> These two strings must be found IN THEIR ENTIRETY within **
1257 *> the 1st 80 columns of program source records, and cannot **
1258 *> follow either a '*>' sequence OR a '*' in col 7. **
1259 *>*****
1260 *> DATE CHANGE DESCRIPTION **
1261 *> ===== **
1262 *> GC0809 Initial coding. **
1263 *>*****
1264 ENVIRONMENT DIVISION.
1265 CONFIGURATION SECTION.
1266 REPOSITORY.
1267 FUNCTION ALL INTRINSIC.
1268 DATA DIVISION.
1269 WORKING-STORAGE SECTION.
1270 01 WS-Compressed-Src-TXT.
1271 05 WS-CS-CHR OCCURS 80 TIMES
1272 PIC X(1).
1273
1274 01 WS-Runtime-Switches-TXT.
1275 05 WS-RS-Found-SPACE-CHR PIC X(1).
1276 88 WS-RS-Skipping-SPACE-BOOL VALUE 'Y'.
1277 88 WS-RS-Not-Skipping-SPACE-BOOL VALUE 'N'.
1278
1279 01 WS-I-SUB USAGE BINARY-CHAR.
1280
1281 01 WS-J-SUB USAGE BINARY-CHAR.
1282 LINKAGE SECTION.
1283 01 L-Argument-1-TXT.
1284 02 L-A1-CHR OCCURS 80 TIMES
1285 PIC X(1).
1286
1287 01 L-Argument-2-CHR PIC X(1).
1288 88 L-A2-LINKAGE-SECTION-BOOL VALUE 'L'.
1289 88 L-A2-IDENT-DIVISION-BOOL VALUE 'I'.
1290 88 L-A2-Nothing-Special-BOOL VALUE ' '.
=====

```

Line Statement

Page: 33

```

=====
1291 /
1292 GC0712 PROCEDURE DIVISION USING BY VALUE L-Argument-1-TXT
1293 GC0712 BY REFERENCE L-Argument-2-CHR.
1294 000-Main SECTION.
1295     SET L-A2-Nothing-Special-BOOL TO TRUE
1296     IF L-A1-CHR (7) = '*'
1297         GOBACK
1298     END-IF
1299     .
1300 *>
1301 *> Compress multiple consecutive spaces
1302 *>
1303     SET WS-RS-Not-Skipping-SPACE-BOOL TO TRUE
1304     MOVE 0 TO WS-J-SUB
1305     MOVE SPACES TO WS-Compressed-Src-TXT
1306     PERFORM VARYING WS-I-SUB FROM 1 BY 1
1307         UNTIL WS-I-SUB > 80
1308         IF L-A1-CHR (WS-I-SUB) = SPACE
1309             IF WS-RS-Not-Skipping-SPACE-BOOL
1310                 ADD 1 TO WS-J-SUB
1311                 MOVE UPPER-CASE(L-A1-CHR (WS-I-SUB))
1312                 TO WS-CS-CHR (WS-J-SUB)
1313                 SET WS-RS-Skipping-SPACE-BOOL TO TRUE
1314             END-IF
1315         ELSE
1316             SET WS-RS-Not-Skipping-SPACE-BOOL TO TRUE
1317             ADD 1 TO WS-J-SUB
1318             MOVE L-A1-CHR (WS-I-SUB) TO WS-CS-CHR (WS-J-SUB)
1319         END-IF
1320     END-PERFORM
1321 *>
1322 *> Scan the compressed source line
1323 *>
1324     PERFORM VARYING WS-I-SUB FROM 1 BY 1
1325         UNTIL WS-I-SUB > 66
1326     EVALUATE TRUE
1327         WHEN WS-CS-CHR (WS-I-SUB) = '*'
1328             IF WS-Compressed-Src-TXT (WS-I-SUB : 2) = '*>'
1329                 GOBACK
1330             END-IF
1331         WHEN (WS-CS-CHR (WS-I-SUB) = 'L') AND (WS-I-SUB < 66)
1332             IF WS-Compressed-Src-TXT (WS-I-SUB : 15)
1333                 = 'LINKAGE SECTION'
1334                 SET L-A2-LINKAGE-SECTION-BOOL TO TRUE
1335                 GOBACK
1336             END-IF
1337         WHEN (WS-CS-CHR (WS-I-SUB) = 'I') AND (WS-I-SUB < 58)
1338             IF WS-Compressed-Src-TXT (WS-I-SUB : 23)
1339                 = 'IDENTIFICATION DIVISION'
1340                 SET L-A2-IDENT-DIVISION-BOOL TO TRUE
1341                 GOBACK
1342             END-IF
1343     END-EVALUATE
1344 END-PERFORM
=====

```

Line Statement

Page: 34

```

=====
1345      *>
1346      *> If we get to here, we never found anything!
1347      *>
1348 +      GOBACK
1349
1350      END PROGRAM CHECKSRC.
1351
1352      IDENTIFICATION DIVISION.
1353      PROGRAM-ID. LISTING.
1354      *>*****
1355      *> This subprogram generates a cross-reference listing of an **
1356      *> GNU COBOL program. **
1357      *>*****
1358      *> **
1359      *> AUTHOR: GARY L. CUTLER **
1360      *> CutlerGL@gmail.com **
1361      *> Copyright (C) 2010, Gary L. Cutler, GPL **
1362      *> **
1363      *> DATE-WRITTEN: April 1, 2010 **
1364      *> **
1365      *>*****
1366      *> DATE CHANGE DESCRIPTION **
1367      *> ===== **
1368      *> GC0410 Initial coding **
1369      *> GC0711 Updates to accommodate the 12MAR2010 version of OC **
1370      *> GC0710 Handle duplicate data names (i.e. 'CORRESPONDING' or **
1371      *> qualified items) better; ignore 'END PROGRAM' recs **
1372      *> so program name doesn't appear in listing. **
1373      *> GC0313 Fix problem where the first procedure name defined **
1374      *> in the PROCEDURE DIVISION lacks a "Defined" line **
1375      *> number. **
1376      *>*****
1377      ENVIRONMENT DIVISION.
1378      CONFIGURATION SECTION.
1379      REPOSITORY.
1380      FUNCTION ALL INTRINSIC.
1381      INPUT-OUTPUT SECTION.
1382      FILE-CONTROL.
1383      SELECT F-Expanded-Src-FILE ASSIGN TO WS-Expanded-Src-Fn-TXT
1384      ORGANIZATION IS LINE SEQUENTIAL.
1385 GC0712 SELECT F-Listing-FILE ASSIGN TO L-Listing-Fn-TXT
1386      ORGANIZATION IS LINE SEQUENTIAL.
1387      SELECT F-Original-Src-FILE ASSIGN TO L-Src-Fn-TXT
1388      ORGANIZATION IS LINE SEQUENTIAL.
1389      SELECT F-Sort-Work-FILE ASSIGN TO DISK.
1390      DATA DIVISION.
1391      FILE SECTION.
1392      FD F-Expanded-Src-FILE.
1393      01 F-Expanded-Src-REC.
1394      05 F-ES-1-CHR PIC X.
1395      05 F-ES-2-256-TXT-256 PIC X(256).
1396 GC0712 01 F-Expanded-Src2-REC.
1397 GC0712 05 F-ES-1-7-TXT PIC X(7).
1398 GC0712 05 F-ES-8-256-TXT PIC X(249).
=====

```


Line Statement

Page: 35

```

=====
1399
1400 GC0712 FD F-Listing-FILE.
1401 GC0712 01 F-Listing-REC PIC X(135).
1402
1403 FD F-Original-Src-FILE.
1404 01 F-Original-Src-REC.
1405 GC0410 05 F-OS-1-128-TXT.
1406 GC0410 10 FILLER PIC X(6).
1407 GC0410 10 F-OS-7-CHR PIC X(1).
1408 GC0712 10 F-OS-8-72-TXT PIC X(65).
1409 GC0712 10 FILLER PIC X(56).
1410 05 F-OS-129-256-TXT PIC X(128).
1411
1412 SD F-Sort-Work-FILE.
1413 01 F-Sort-Work-REC.
1414 05 F-SW-Prog-ID-TXT PIC X(15).
1415 05 F-SW-Token-Uc-TXT PIC X(32).
1416 05 F-SW-Token-TXT PIC X(32).
1417 05 F-SW-Section-TXT PIC X(15).
1418 05 F-SW-Def-Line-NUM PIC 9(6).
1419 05 F-SW-Reference-TXT.
1420 10 F-SW-Ref-Line-NUM PIC 9(6).
1421 10 F-SW-Ref-Flag-CHR PIC X(1).
1422
1423 WORKING-STORAGE SECTION.
1424 78 WS-Lines-Per-Rec-CONST VALUE 8.
1425
1426 01 WS-Curr-CHR PIC X(1).
1427 88 WS-Curr-Char-Is-Punct-BOOL VALUE '=' , '(' , ')' ,
1428 '* , '/' , '&' ,
1429 ':' , ';' , '<' ,
1430 '>' , ':' .
1431 88 WS-Curr-Char-Is-Quote-BOOL VALUE '"' , "'" .
1432 88 WS-Curr-Char-Is-X-BOOL VALUE 'x' , 'X' .
1433 88 WS-Curr-Char-Is-Z-BOOL VALUE 'z' , 'Z' .
1434
1435 01 WS-Curr-Division-TXT PIC X(1).
1436 GC1010 88 WS-CD-In-IDENT-DIV-BOOL VALUE 'i' , 'I' , '?' .
1437 GC1010 88 WS-CD-In-ENV-DIV-BOOL VALUE 'e' , 'E' .
1438 GC1010 88 WS-CD-In-DATA-DIV-BOOL VALUE 'd' , 'D' .
1439 GC1010 88 WS-CD-In-PROC-DIV-BOOL VALUE 'p' , 'P' .
1440
1441 01 WS-Curr-Line-NUM PIC 9(6).
1442
1443 01 WS-Curr-Prog-ID-TXT.
1444 05 FILLER PIC X(12).
1445 05 WS-CPI-13-15-TXT PIC X(3).
1446 GC0712 05 WS-CPI-16-CHR PIC X(1).
1447
1448 01 WS-Curr-Section-TXT.
1449 05 WS-CS-1-CHR PIC X(1).
1450 05 WS-CS-2-14-TXT.
1451 10 FILLER PIC X(10).
1452 10 WS-CS-11-14-TXT PIC X(3).

```

Line Statement

Page: 36

```

=====
1453          05 WS-CS-15-CHR          PIC X(1).
1454
1455          01 WS-Curr-Verb-TXT      PIC X(12).
1456
1457          01 WS-Delim-TXT          PIC X(2).
1458
1459          01 WS-Dummy-TXT          PIC X(1).
1460
1461          01 WS-Expanded-Src-Fn-TXT PIC X(256).
1462
1463          01 WS-Filename-TXT       PIC X(256).
1464
1465          01 WS-Group-Indicators-TXT.
1466             05 WS-GI-Prog-ID-TXT  PIC X(15).
1467             05 WS-GI-Token-TXT    PIC X(32).
1468
1469          01 WS-Held-Reference-TXT  PIC X(100).
1470
1471          01 WS-I-SUB               USAGE BINARY-LONG.
1472
1473          01 WS-J-SUB               USAGE BINARY-LONG.
1474
1475          01 WS-Lines-Left-NUM      USAGE BINARY-LONG.
1476
1477          01 WS-Lines-Per-Page-NUM  USAGE BINARY-LONG.
1478
1479          01 WS-Lines-Per-Page-Env-TXT PIC X(256).
1480
1481 GC1010 01 WS-Main-Module-Name-TXT PIC X(256).
1482
1483          01 WS-Next-CHR            PIC X(1).
1484             88 WS-Next-Char-Is-Quote-BOOL VALUE '"', '''.
1485
1486          01 WS-OS-Type-FILLER-TXT.
1487             05 VALUE 'Windows'     PIC X(14).
1488             05 VALUE 'Windows/Cygwin' PIC X(14).
1489             05 VALUE 'UNIX/Linux'  PIC X(14).
1490             05 VALUE 'OSX'          PIC X(14).
1491             05 VALUE 'Windows/MinGW' PIC X(14).
1492          01 WS-OS-Types-TXT REDEFINES WS-OS-Type-FILLER-TXT.
1493             05 WS-OS-Type-TXT      PIC X(14)
1494                                     OCCURS 5 TIMES .
1495
1496 GC0712 01 WS-Page-NUM             USAGE BINARY-LONG.
1497
1498 GC0712 01 WS-Page-No-TXT.
1499 GC0712 05 WS-PN-Literal-TXT      PIC X(6).
1500 GC0712 05 WS-PN-Page-NUM        PIC Z(3)9.
1501
1502          01 WS-Program-Path-TXT   PIC X(256).
1503
1504          01 WS-Reserved-Words-TXT.
1505             05 VALUE '              ' PIC X(33).
1506             05 VALUE 'IABS          ' PIC X(33).

```

Line Statement

Page: 37

```

=====
1507      05 VALUE 'VACCEPT          ' PIC X(33).
1508      05 VALUE ' ACCESS          ' PIC X(33).
1509      05 VALUE ' IACOS            ' PIC X(33).
1510      05 VALUE ' ACTIVE-CLASS     ' PIC X(33).      UNIMPLEMENTED
1511      05 VALUE ' VADD             ' PIC X(33).
1512      05 VALUE ' ADDRESS          ' PIC X(33).
1513      05 VALUE ' ADVANCING        ' PIC X(33).
1514      05 VALUE ' KAFTER           ' PIC X(33).
1515      05 VALUE ' ALIGNED          ' PIC X(33).      UNIMPLEMENTED
1516      05 VALUE ' ALL              ' PIC X(33).
1517      05 VALUE ' VALLOCATE        ' PIC X(33).
1518      05 VALUE ' ALPHABET         ' PIC X(33).
1519      05 VALUE ' ALPHABETIC       ' PIC X(33).
1520      05 VALUE ' ALPHABETIC-LOWER ' PIC X(33).
1521      05 VALUE ' ALPHABETIC-UPPER ' PIC X(33).
1522      05 VALUE ' ALPHANUMERIC     ' PIC X(33).
1523      05 VALUE ' ALPHANUMERIC-EDITED ' PIC X(33).
1524      05 VALUE ' ALSO            ' PIC X(33).
1525      05 VALUE ' VALTER           ' PIC X(33).
1526      05 VALUE ' ALTERNATE        ' PIC X(33).
1527      05 VALUE ' AND              ' PIC X(33).
1528      05 VALUE ' IANNUITY         ' PIC X(33).
1529      05 VALUE ' ANY              ' PIC X(33).
1530      05 VALUE ' ANYCASE          ' PIC X(33).      UNIMPLEMENTED
1531      05 VALUE ' ARE              ' PIC X(33).
1532      05 VALUE ' AREA             ' PIC X(33).
1533      05 VALUE ' AREAS            ' PIC X(33).
1534      05 VALUE ' ARGUMENT-NUMBER  ' PIC X(33).
1535      05 VALUE ' ARGUMENT-VALUE   ' PIC X(33).
1536      05 VALUE ' ARITHMETIC       ' PIC X(33).      UNIMPLEMENTED
1537      05 VALUE ' AS               ' PIC X(33).
1538      05 VALUE ' ASCENDING        ' PIC X(33).
1539      05 VALUE ' ASCII            ' PIC X(33).
1540      05 VALUE ' IASIN            ' PIC X(33).
1541      05 VALUE ' ASSIGN           ' PIC X(33).
1542      05 VALUE ' AT               ' PIC X(33).
1543      05 VALUE ' IATAN            ' PIC X(33).
1544 GC0711 05 VALUE ' ATTRIBUTE        ' PIC X(33).
1545      05 VALUE ' AUTHOR           ' PIC X(33).      OBSOLETE
1546      05 VALUE ' AUTO            ' PIC X(33).
1547      05 VALUE ' AUTO-SKIP        ' PIC X(33).
1548      05 VALUE ' AUTOMATIC        ' PIC X(33).
1549      05 VALUE ' AUTOTERMINATE    ' PIC X(33).
1550      05 VALUE ' AWAY-FROM-ZERO   ' PIC X(33).
1551      05 VALUE ' B-AND            ' PIC X(33).      UNIMPLEMENTED
1552      05 VALUE ' B-NOT            ' PIC X(33).      UNIMPLEMENTED
1553      05 VALUE ' B-OR             ' PIC X(33).      UNIMPLEMENTED
1554      05 VALUE ' B-XOR            ' PIC X(33).      UNIMPLEMENTED
1555      05 VALUE ' BACKGROUND-COLOR ' PIC X(33).
1556      05 VALUE ' BACKGROUND-COLOUR ' PIC X(33).
1557      05 VALUE ' BASED           ' PIC X(33).
1558      05 VALUE ' BEEP            ' PIC X(33).
1559      05 VALUE ' BEFORE          ' PIC X(33).
1560      05 VALUE ' BELL             ' PIC X(33).
=====

```

Line Statement

Page: 38

```

=====
1561      05 VALUE ' BINARY          ' PIC X(33).
1562      05 VALUE ' BINARY-C-LONG   ' PIC X(33).
1563      05 VALUE ' BINARY-CHAR     ' PIC X(33).
1564      05 VALUE ' BINARY-DOUBLE   ' PIC X(33).
1565      05 VALUE ' BINARY-INT      ' PIC X(33).
1566      05 VALUE ' BINARY-LONG     ' PIC X(33).
1567      05 VALUE ' BINARY-LONG-LONG ' PIC X(33).
1568      05 VALUE ' BINARY-SHORT    ' PIC X(33).
1569      05 VALUE ' BIT              ' PIC X(33).      UNIMPLEMENTED
1570      05 VALUE ' BLANK           ' PIC X(33).
1571      05 VALUE ' BLINK           ' PIC X(33).
1572      05 VALUE ' BLOCK           ' PIC X(33).
1573      05 VALUE ' BOOLEAN         ' PIC X(33).      UNIMPLEMENTED
1574      05 VALUE ' IBOOLEAN-OF-INTEGER ' PIC X(33).      UNIMPLEMENTED
1575      05 VALUE ' BOTTOM          ' PIC X(33).
1576      05 VALUE ' YBY             ' PIC X(33).
1577      05 VALUE ' IBYTE-LENGTH    ' PIC X(33).
1578      05 VALUE ' MC01            ' PIC X(33).
1579      05 VALUE ' MC02            ' PIC X(33).
1580      05 VALUE ' MC03            ' PIC X(33).
1581      05 VALUE ' MC04            ' PIC X(33).
1582      05 VALUE ' MC05            ' PIC X(33).
1583      05 VALUE ' MC06            ' PIC X(33).
1584      05 VALUE ' MC07            ' PIC X(33).
1585      05 VALUE ' MC08            ' PIC X(33).
1586      05 VALUE ' MC09            ' PIC X(33).
1587      05 VALUE ' MC10            ' PIC X(33).
1588      05 VALUE ' MC11            ' PIC X(33).
1589      05 VALUE ' MC12            ' PIC X(33).
1590      05 VALUE ' VCALL           ' PIC X(33).
1591      05 VALUE ' MCALL-CONVENTION ' PIC X(33).
1592      05 VALUE ' VCANCEL         ' PIC X(33).
1593      05 VALUE ' CAPACITY        ' PIC X(33).      UNIMPLEMENTED
1594      05 VALUE ' CD              ' PIC X(33).      OBSOLETE
1595      05 VALUE ' CENTER         ' PIC X(33).      UNIMPLEMENTED
1596      05 VALUE ' CF             ' PIC X(33).
1597      05 VALUE ' CH             ' PIC X(33).
1598      05 VALUE ' CHAIN          ' PIC X(33).      UNIMPLEMENTED
1599      05 VALUE ' CHAINING       ' PIC X(33).
1600      05 VALUE ' ICHAR          ' PIC X(33).
1601      05 VALUE ' ICHAR-NATIONAL  ' PIC X(33).      UNIMPLEMENTED
1602      05 VALUE ' CHARACTER      ' PIC X(33).
1603      05 VALUE ' CHARACTERS     ' PIC X(33).
1604      05 VALUE ' CLASS          ' PIC X(33).
1605      05 VALUE ' CLASS-ID       ' PIC X(33).      UNIMPLEMENTED
1606 GC0711 05 VALUE ' CLASSIFICATION ' PIC X(33).
1607      05 VALUE ' VCLOSE         ' PIC X(33).
1608      05 VALUE ' ICOB-CRT-STATUS ' PIC X(33).
1609      05 VALUE ' CODE           ' PIC X(33).
1610      05 VALUE ' CODE-SET       ' PIC X(33).
1611      05 VALUE ' COL            ' PIC X(33).
1612      05 VALUE ' COLLATING     ' PIC X(33).
1613      05 VALUE ' COLS          ' PIC X(33).
1614      05 VALUE ' COLUMN        ' PIC X(33).
=====

```

Line Statement

Page: 39

```

=====
1615      05 VALUE ' COLUMNS                ' PIC X(33).
1616      05 VALUE ' ICOMBINED-DATETIME      ' PIC X(33).
1617      05 VALUE ' COMMA                    ' PIC X(33).
1618      05 VALUE ' COMMAND-LINE             ' PIC X(33).
1619      05 VALUE ' VCOMMIT                   ' PIC X(33).
1620      05 VALUE ' COMMON                     ' PIC X(33).
1621      05 VALUE ' COMMUNICATION             ' PIC X(33).      OBSOLETE
1622      05 VALUE ' COMP                       ' PIC X(33).
1623      05 VALUE ' COMP-1                     ' PIC X(33).
1624      05 VALUE ' COMP-2                     ' PIC X(33).
1625      05 VALUE ' COMP-3                     ' PIC X(33).
1626      05 VALUE ' COMP-4                     ' PIC X(33).
1627      05 VALUE ' COMP-5                     ' PIC X(33).
1628      05 VALUE ' COMP-6                     ' PIC X(33).
1629      05 VALUE ' COMP-X                     ' PIC X(33).
1630      05 VALUE ' COMPUTATIONAL              ' PIC X(33).
1631      05 VALUE ' COMPUTATIONAL-1            ' PIC X(33).
1632      05 VALUE ' COMPUTATIONAL-2            ' PIC X(33).
1633      05 VALUE ' COMPUTATIONAL-3            ' PIC X(33).
1634      05 VALUE ' COMPUTATIONAL-4            ' PIC X(33).
1635      05 VALUE ' COMPUTATIONAL-5            ' PIC X(33).
1636      05 VALUE ' COMPUTATIONAL-X            ' PIC X(33).
1637      05 VALUE ' VCOMPUTE                   ' PIC X(33).
1638      05 VALUE ' ICONCATENATE               ' PIC X(33).
1639 GC0712 05 VALUE ' CONDITION                 ' PIC X(33).
1640      05 VALUE ' KCONFIGURATION              ' PIC X(33).
1641      05 VALUE ' MCONSOLE                    ' PIC X(33).
1642      05 VALUE ' CONSTANT                    ' PIC X(33).
1643      05 VALUE ' CONTAINS                     ' PIC X(33).
1644 GC0712 05 VALUE ' ACONTENT                  ' PIC X(33).
1645      05 VALUE ' VCONTINUE                   ' PIC X(33).
1646      05 VALUE ' CONTROL                      ' PIC X(33).
1647      05 VALUE ' CONTROLS                     ' PIC X(33).
1648 GC0711 05 VALUE ' CONVERSION                 ' PIC X(33).
1649      05 VALUE ' KCONVERTING                 ' PIC X(33).
1650      05 VALUE ' COPY                         ' PIC X(33).
1651      05 VALUE ' CORR                         ' PIC X(33).
1652      05 VALUE ' CORRESPONDING               ' PIC X(33).
1653      05 VALUE ' ICOS                        ' PIC X(33).
1654      05 VALUE ' KCOUNT                      ' PIC X(33).
1655      05 VALUE ' CRT                          ' PIC X(33).
1656      05 VALUE ' CRT-UNDER                    ' PIC X(33).
1657      05 VALUE ' MCSP                         ' PIC X(33).
1658      05 VALUE ' CURRENCY                     ' PIC X(33).
1659 GC0711 05 VALUE ' ICURRENCY-SYMBOL           ' PIC X(33).
1660      05 VALUE ' ICURRENT-DATE                ' PIC X(33).
1661      05 VALUE ' CURSOR                       ' PIC X(33).
1662      05 VALUE ' CYCLE                        ' PIC X(33).
1663      05 VALUE ' KDATA                        ' PIC X(33).
1664      05 VALUE ' DATA-POINTER                 ' PIC X(33).      UNIMPLEMENTED
1665      05 VALUE ' DATE                         ' PIC X(33).
1666      05 VALUE ' DATE-COMPILED                ' PIC X(33).      OBSOLETE
1667      05 VALUE ' DATE-MODIFIED                ' PIC X(33).      OBSOLETE
1668      05 VALUE ' IDATE-OF-INTEGGER            ' PIC X(33).
=====

```

Line Statement

Page: 40

```

=====
1669          05 VALUE 'IDATE-TO-YYYYMMDD          ' PIC X(33).
1670          05 VALUE ' DATE-WRITTEN              ' PIC X(33).      OBSOLETE
1671          05 VALUE ' DAY                        ' PIC X(33).
1672          05 VALUE 'IDAY-OF-INTEGGER           ' PIC X(33).
1673          05 VALUE ' DAY-OF-WEEK               ' PIC X(33).
1674          05 VALUE 'IDAY-TO-YYYYDDD           ' PIC X(33).
1675          05 VALUE ' DE                         ' PIC X(33).
1676 GC0712    05 VALUE 'IDEBUG-CONTENTS            ' PIC X(33).
1677 GC0712    05 VALUE 'IDEBUG-ITEM                ' PIC X(33).
1678 GC0712    05 VALUE 'IDEBUG-LINE                ' PIC X(33).
1679 GC0712    05 VALUE 'IDEBUG-NAME                ' PIC X(33).
1680 GC0712    05 VALUE 'IDEBUG-SUB-1              ' PIC X(33).
1681 GC0712    05 VALUE 'IDEBUG-SUB-2              ' PIC X(33).
1682 GC0712    05 VALUE 'IDEBUG-SUB-3              ' PIC X(33).
1683          05 VALUE ' DEBUGGING                  ' PIC X(33).
1684          05 VALUE ' DECIMAL-POINT              ' PIC X(33).
1685          05 VALUE ' DECLARATIVES                ' PIC X(33).
1686          05 VALUE ' DEFAULT                     ' PIC X(33).
1687          05 VALUE 'VDELETE                      ' PIC X(33).
1688          05 VALUE ' DELIMITED                   ' PIC X(33).
1689          05 VALUE 'KDELIMITER                   ' PIC X(33).
1690          05 VALUE ' DEPENDING                    ' PIC X(33).
1691          05 VALUE ' DESCENDING                   ' PIC X(33).
1692          05 VALUE ' DESTINATION                  ' PIC X(33).      UNIMPLEMENTED
1693          05 VALUE ' DETAIL                       ' PIC X(33).
1694 GC0711    05 VALUE ' DISC                        ' PIC X(33).
1695          05 VALUE ' DISK                         ' PIC X(33).
1696          05 VALUE 'VDISPLAY                      ' PIC X(33).
1697          05 VALUE 'IDISPLAY-OF                   ' PIC X(33).      UNIMPLEMENTED
1698          05 VALUE 'VDIVIDE                       ' PIC X(33).
1699          05 VALUE 'KDIVISION                     ' PIC X(33).
1700          05 VALUE 'KDOWN                          ' PIC X(33).
1701          05 VALUE ' DUPLICATES                   ' PIC X(33).
1702          05 VALUE ' DYNAMIC                      ' PIC X(33).
1703          05 VALUE ' IE                           ' PIC X(33).
1704          05 VALUE ' EBCDIC                       ' PIC X(33).
1705 GC0712    05 VALUE ' EC                          ' PIC X(33).
1706          05 VALUE ' EGI                          ' PIC X(33).      OBSOLETE
1707          05 VALUE 'VELSE                          ' PIC X(33).
1708          05 VALUE ' EMI                          ' PIC X(33).      OBSOLETE
1709          05 VALUE ' EMPTY-CHECK                  ' PIC X(33).
1710          05 VALUE 'VENABLE                       ' PIC X(33).      OBSOLETE
1711 GC0710    05 VALUE 'KEND                          ' PIC X(33).
1712          05 VALUE ' END-ACCEPT                    ' PIC X(33).
1713          05 VALUE ' END-ADD                       ' PIC X(33).
1714          05 VALUE ' END-CALL                      ' PIC X(33).
1715          05 VALUE ' END-CHAIN                     ' PIC X(33).      UNIMPLEMENTED
1716          05 VALUE ' END-COMPUTE                  ' PIC X(33).
1717          05 VALUE ' END-DELETE                    ' PIC X(33).
1718          05 VALUE ' END-DISPLAY                   ' PIC X(33).
1719          05 VALUE ' END-DIVIDE                    ' PIC X(33).
1720          05 VALUE ' END-EVALUATE                  ' PIC X(33).
1721          05 VALUE ' END-IF                        ' PIC X(33).
1722          05 VALUE ' END-MULTIPLY                  ' PIC X(33).
=====

```

Line Statement

Page: 41

```

=====
1723      05 VALUE ' END-OF-PAGE          ' PIC X(33).
1724      05 VALUE ' END-PERFORM        ' PIC X(33).
1725      05 VALUE ' END-READ           ' PIC X(33).
1726      05 VALUE ' END-RECEIVE        ' PIC X(33).      OBSOLETE
1727      05 VALUE ' END-RETURN          ' PIC X(33).
1728      05 VALUE ' END-REWRITE         ' PIC X(33).
1729      05 VALUE ' END-SEARCH         ' PIC X(33).
1730      05 VALUE ' END-START           ' PIC X(33).
1731      05 VALUE ' END-STRING          ' PIC X(33).
1732      05 VALUE ' END-SUBTRACT        ' PIC X(33).
1733      05 VALUE ' END-UNSTRING        ' PIC X(33).
1734      05 VALUE ' END-WRITE           ' PIC X(33).
1735      05 VALUE ' VENTRY               ' PIC X(33).
1736      05 VALUE ' ENTRY-CONVENTION    ' PIC X(33).      UNIMPLEMENTED
1737      05 VALUE ' KENVIRONMENT        ' PIC X(33).
1738      05 VALUE ' ENVIRONMENT-NAME    ' PIC X(33).
1739      05 VALUE ' ENVIRONMENT-VALUE   ' PIC X(33).
1740      05 VALUE ' EO                  ' PIC X(33).      UNIMPLEMENTED
1741      05 VALUE ' EOL                 ' PIC X(33).
1742      05 VALUE ' EOP                 ' PIC X(33).
1743      05 VALUE ' EOS                 ' PIC X(33).
1744      05 VALUE ' EQUAL               ' PIC X(33).
1745      05 VALUE ' KEQUALS            ' PIC X(33).
1746      05 VALUE ' ERASE               ' PIC X(33).
1747      05 VALUE ' ERROR               ' PIC X(33).
1748      05 VALUE ' ESCAPE              ' PIC X(33).
1749      05 VALUE ' ESI                ' PIC X(33).      OBSOLETE
1750      05 VALUE ' VEVALUATE          ' PIC X(33).
1751      05 VALUE ' EXCEPTION           ' PIC X(33).
1752      05 VALUE ' IEXCEPTION-FILE     ' PIC X(33).
1753      05 VALUE ' IEXCEPTION-FILE-N   ' PIC X(33).      UNIMPLEMENTED
1754      05 VALUE ' IEXCEPTION-LOCATION   ' PIC X(33).
1755      05 VALUE ' IEXCEPTION-LOCATION-N ' PIC X(33).      UNIMPLEMENTED
1756      05 VALUE ' EXCEPTION-OBJECT    ' PIC X(33).      UNIMPLEMENTED
1757      05 VALUE ' IEXCEPTION-STATEMENT ' PIC X(33).
1758      05 VALUE ' IEXCEPTION-STATUS   ' PIC X(33).
1759      05 VALUE ' EXCLUSIVE            ' PIC X(33).
1760      05 VALUE ' VEXIT               ' PIC X(33).
1761      05 VALUE ' IEXP                ' PIC X(33).
1762      05 VALUE ' IEXP10              ' PIC X(33).
1763      05 VALUE ' EXPANDS             ' PIC X(33).      UNIMPLEMENTED
1764      05 VALUE ' EXTEND              ' PIC X(33).
1765      05 VALUE ' EXTERNAL            ' PIC X(33).
1766      05 VALUE ' IFACTORIAL          ' PIC X(33).
1767      05 VALUE ' FACTORY             ' PIC X(33).      UNIMPLEMENTED
1768      05 VALUE ' FALSE               ' PIC X(33).
1769      05 VALUE ' KFD                 ' PIC X(33).
1770      05 VALUE ' KFILE               ' PIC X(33).
1771      05 VALUE ' FILE-CONTROL        ' PIC X(33).
1772      05 VALUE ' FILE-ID             ' PIC X(33).
1773 GC1113 05 VALUE ' FILLER            ' PIC X(33).
1774      05 VALUE ' FINAL               ' PIC X(33).
1775      05 VALUE ' FIRST               ' PIC X(33).
1776 GC0712 05 VALUE ' FLOAT-BINARY-128 ' PIC X(33).      UNIMPLEMENTED
=====

```

Line Statement

Page: 42

```

=====
1777 GC0712 05 VALUE ' FLOAT-BINARY-32 ' PIC X(33). UNIMPLEMENTED
1778 GC0712 05 VALUE ' FLOAT-BINARY-64 ' PIC X(33). UNIMPLEMENTED
1779 05 VALUE ' FLOAT-DECIMAL-16 ' PIC X(33).
1780 05 VALUE ' FLOAT-DECIMAL-34 ' PIC X(33).
1781 05 VALUE ' FLOAT-EXTENDED ' PIC X(33). UNIMPLEMENTED
1782 GC0712 05 VALUE ' FLOAT-INFINITY ' PIC X(33). UNIMPLEMENTED
1783 05 VALUE ' FLOAT-LONG ' PIC X(33).
1784 GC0712 05 VALUE ' FLOAT-NOT-A-NUMBER ' PIC X(33). UNIMPLEMENTED
1785 05 VALUE ' FLOAT-SHORT ' PIC X(33).
1786 05 VALUE ' FOOTING ' PIC X(33).
1787 05 VALUE ' FOR ' PIC X(33).
1788 05 VALUE ' FOREGROUND-COLOR ' PIC X(33).
1789 05 VALUE ' FOREGROUND-COLOUR ' PIC X(33).
1790 GC0711 05 VALUE ' FOREVER ' PIC X(33).
1791 05 VALUE ' FORMAT ' PIC X(33). UNIMPLEMENTED
1792 GC0711 05 VALUE ' IFORMATTED-CURRENT-DATE ' PIC X(33). UNIMPLEMENTED
1793 GC0711 05 VALUE ' IFORMATTED-DATE ' PIC X(33). UNIMPLEMENTED
1794 GC0711 05 VALUE ' IFORMATTED-DATETIME ' PIC X(33). UNIMPLEMENTED
1795 GC0711 05 VALUE ' IFORMATTED-TIME ' PIC X(33). UNIMPLEMENTED
1796 05 VALUE ' MFORMFEED ' PIC X(33).
1797 05 VALUE ' IFRACTION-PART ' PIC X(33).
1798 05 VALUE ' VFREE ' PIC X(33).
1799 05 VALUE ' FROM ' PIC X(33).
1800 05 VALUE ' FULL ' PIC X(33).
1801 05 VALUE ' FUNCTION ' PIC X(33).
1802 GC0712 05 VALUE ' KFUNCTION-ID ' PIC X(33).
1803 05 VALUE ' FUNCTION-POINTER ' PIC X(33). UNIMPLEMENTED
1804 05 VALUE ' VGENERATE ' PIC X(33).
1805 05 VALUE ' GET ' PIC X(33). UNIMPLEMENTED
1806 05 VALUE ' KGIVING ' PIC X(33).
1807 05 VALUE ' GLOBAL ' PIC X(33).
1808 05 VALUE ' VGO ' PIC X(33).
1809 05 VALUE ' VGOBACK ' PIC X(33).
1810 05 VALUE ' GREATER ' PIC X(33).
1811 05 VALUE ' GROUP ' PIC X(33).
1812 05 VALUE ' GROUP-USAGE ' PIC X(33). UNIMPLEMENTED
1813 05 VALUE ' HEADING ' PIC X(33).
1814 05 VALUE ' HIGH-VALUE ' PIC X(33).
1815 05 VALUE ' HIGH-VALUES ' PIC X(33).
1816 GC0711 05 VALUE ' IHIGHEST-ALGEBRAIC ' PIC X(33).
1817 05 VALUE ' HIGHLIGHT ' PIC X(33).
1818 05 VALUE ' I-O ' PIC X(33).
1819 05 VALUE ' I-O-CONTROL ' PIC X(33).
1820 05 VALUE ' KID ' PIC X(33).
1821 05 VALUE ' KIDENTIFICATION ' PIC X(33).
1822 05 VALUE ' VIF ' PIC X(33).
1823 05 VALUE ' IGNORE ' PIC X(33).
1824 05 VALUE ' IGNORING ' PIC X(33).
1825 05 VALUE ' IMPLEMENTS ' PIC X(33). UNIMPLEMENTED
1826 05 VALUE ' IN ' PIC X(33).
1827 05 VALUE ' INDEX ' PIC X(33).
1828 05 VALUE ' KINDEXED ' PIC X(33).
1829 05 VALUE ' INDICATE ' PIC X(33).
1830 05 VALUE ' INDIRECT ' PIC X(33). UNIMPLEMENTED
=====

```


Line Statement

Page: 43

```

=====
1831      05 VALUE ' INHERITS          ' PIC X(33).      UNIMPLEMENTED
1832      05 VALUE ' INITIAL          ' PIC X(33).
1833      05 VALUE ' VINITIALISE      ' PIC X(33).
1834      05 VALUE ' INITIALISED     ' PIC X(33).
1835      05 VALUE ' VINITIALIZE      ' PIC X(33).
1836      05 VALUE ' INITIALIZED      ' PIC X(33).
1837      05 VALUE ' VINITIATE        ' PIC X(33).
1838      05 VALUE ' INPUT             ' PIC X(33).
1839      05 VALUE ' KINPUT-OUTPUT     ' PIC X(33).
1840      05 VALUE ' VINSPECT          ' PIC X(33).
1841      05 VALUE ' INSTALLATION      ' PIC X(33).      OBSOLETE
1842      05 VALUE ' IINTEGER          ' PIC X(33).
1843      05 VALUE ' IINTEGER-OF-BOOLEAN ' PIC X(33).      UNIMPLEMENTED
1844      05 VALUE ' IINTEGER-OF-DATE   ' PIC X(33).
1845      05 VALUE ' IINTEGER-OF-DAY   ' PIC X(33).
1846 GC0711 05 VALUE ' IINTEGER-OF-FORMATTED-DATE ' PIC X(33).      UNIMPLEMENTED
1847      05 VALUE ' IINTEGER-PART     ' PIC X(33).
1848      05 VALUE ' INTERFACE          ' PIC X(33).      UNIMPLEMENTED
1849      05 VALUE ' INTERFACE-ID      ' PIC X(33).      UNIMPLEMENTED
1850      05 VALUE ' INTERMEDIATE      ' PIC X(33).      UNIMPLEMENTED
1851      05 VALUE ' KINTO             ' PIC X(33).
1852      05 VALUE ' INTRINSIC         ' PIC X(33).
1853      05 VALUE ' INVALID          ' PIC X(33).
1854      05 VALUE ' INVOKE           ' PIC X(33).      UNIMPLEMENTED
1855      05 VALUE ' IS                ' PIC X(33).
1856      05 VALUE ' JUST              ' PIC X(33).
1857      05 VALUE ' JUSTIFIED        ' PIC X(33).
1858      05 VALUE ' KEPT              ' PIC X(33).
1859      05 VALUE ' KEY               ' PIC X(33).
1860      05 VALUE ' KEYBOARD          ' PIC X(33).
1861      05 VALUE ' LABEL             ' PIC X(33).
1862      05 VALUE ' LAST              ' PIC X(33).
1863      05 VALUE ' LC_ALL             ' PIC X(33).      UNIMPLEMENTED
1864      05 VALUE ' LC_COLLATE        ' PIC X(33).      UNIMPLEMENTED
1865      05 VALUE ' LC_CTYPE          ' PIC X(33).      UNIMPLEMENTED
1866      05 VALUE ' LC_MESSAGES       ' PIC X(33).      UNIMPLEMENTED
1867      05 VALUE ' LC_MONETARY       ' PIC X(33).      UNIMPLEMENTED
1868      05 VALUE ' LC_NUMERIC        ' PIC X(33).      UNIMPLEMENTED
1869      05 VALUE ' LC_TIME           ' PIC X(33).      UNIMPLEMENTED
1870      05 VALUE ' LEADING           ' PIC X(33).
1871      05 VALUE ' LEFT              ' PIC X(33).
1872      05 VALUE ' LEFT-JUSTIFY      ' PIC X(33).      UNIMPLEMENTED
1873      05 VALUE ' LEFTLINE          ' PIC X(33).
1874 GC0712 05 VALUE ' LENGTH           ' PIC X(33).
1875 GC0711 05 VALUE ' ILENGTH-AN      ' PIC X(33).
1876      05 VALUE ' LENGTH-CHECK     ' PIC X(33).
1877      05 VALUE ' LESS              ' PIC X(33).
1878      05 VALUE ' LIMIT             ' PIC X(33).
1879      05 VALUE ' LIMITS            ' PIC X(33).
1880      05 VALUE ' LINAGE            ' PIC X(33).
1881      05 VALUE ' ILINAGE-COUNTER    ' PIC X(33).
1882      05 VALUE ' LINE              ' PIC X(33).
1883      05 VALUE ' LINE-COUNTER      ' PIC X(33).
1884      05 VALUE ' LINES             ' PIC X(33).
=====

```

Line Statement

Page: 44

```

=====
1885          05 VALUE 'KLINKAGE          ' PIC X(33).
1886          05 VALUE 'KLOCAL-STORAGE   ' PIC X(33).
1887          05 VALUE ' LOCALE          ' PIC X(33).
1888 GC0711    05 VALUE 'ILOCALE-COMPARE   ' PIC X(33).
1889          05 VALUE 'ILOCALE-DATE     ' PIC X(33).
1890          05 VALUE 'ILOCALE-TIME     ' PIC X(33).
1891          05 VALUE 'ILOCALE-TIME-FROM-SECONDS ' PIC X(33).
1892          05 VALUE ' LOCK            ' PIC X(33).
1893          05 VALUE ' ILOG             ' PIC X(33).
1894          05 VALUE ' ILOG10          ' PIC X(33).
1895          05 VALUE ' LOW-VALUE       ' PIC X(33).
1896          05 VALUE ' LOW-VALUES     ' PIC X(33).
1897          05 VALUE ' LOWER           ' PIC X(33).
1898          05 VALUE ' ILOWER-CASE     ' PIC X(33).
1899 GC0711    05 VALUE ' ILOWEST-ALGEBRAIC ' PIC X(33).
1900          05 VALUE ' LOWLIGHT       ' PIC X(33).
1901          05 VALUE ' MANUAL          ' PIC X(33).
1902          05 VALUE ' IMAX            ' PIC X(33).
1903          05 VALUE ' IMEAN           ' PIC X(33).
1904          05 VALUE ' IMEDIAN         ' PIC X(33).
1905          05 VALUE ' MEMORY          ' PIC X(33).
1906          05 VALUE ' VMERGE          ' PIC X(33).
1907          05 VALUE ' MESSAGE        ' PIC X(33).
1908          05 VALUE ' METHOD           ' PIC X(33).
1909          05 VALUE ' METHOD-ID        ' PIC X(33).
1910          05 VALUE ' IMIDRANGE       ' PIC X(33).
1911          05 VALUE ' IMIN            ' PIC X(33).
1912          05 VALUE ' MINUS           ' PIC X(33).
1913          05 VALUE ' IMOD            ' PIC X(33).
1914          05 VALUE ' MODE            ' PIC X(33).
1915          05 VALUE ' IMODULE-CALLER-ID ' PIC X(33).
1916          05 VALUE ' IMODULE-DATE    ' PIC X(33).
1917          05 VALUE ' IMODULE-FORMATTED-DATE ' PIC X(33).
1918          05 VALUE ' IMODULE-ID      ' PIC X(33).
1919          05 VALUE ' IMODULE-PATH    ' PIC X(33).
1920          05 VALUE ' IMODULE-SOURCE  ' PIC X(33).
1921          05 VALUE ' IMODULE-TIME   ' PIC X(33).
1922 GC0711    05 VALUE ' IMONETARY-DECIMAL-POINT ' PIC X(33).
1923 GC0711    05 VALUE ' IMONETARY-THOUSANDS-SEPARATOR ' PIC X(33).
1924          05 VALUE ' VMOVE          ' PIC X(33).
1925          05 VALUE ' MULTIPLE       ' PIC X(33).
1926          05 VALUE ' VMULTIPLY      ' PIC X(33).
1927 GC0711    05 VALUE ' NAME            ' PIC X(33).
1928          05 VALUE ' NATIONAL       ' PIC X(33).
1929          05 VALUE ' NATIONAL-EDITED ' PIC X(33).
1930          05 VALUE ' INATIONAL-OF    ' PIC X(33).
1931          05 VALUE ' NATIVE          ' PIC X(33).
1932          05 VALUE ' NEAREST-AWAY-FROM-ZERO ' PIC X(33).
1933          05 VALUE ' NEAREST-EVEN   ' PIC X(33).
1934          05 VALUE ' NEAREST-TOWARD-ZERO ' PIC X(33).
1935          05 VALUE ' NEGATIVE       ' PIC X(33).
1936          05 VALUE ' NESTED         ' PIC X(33).
1937          05 VALUE ' VNEXT          ' PIC X(33).
1938          05 VALUE ' NO              ' PIC X(33).
=====

```

```

OBSOLETE
UNIMPLEMENTED
UNIMPLEMENTED

```

```

UNIMPLEMENTED

```

```

UNIMPLEMENTED

```

Line Statement

Page: 45

```

=====
1939      05 VALUE ' NO-ECHO          ' PIC X(33).
1940      05 VALUE ' NONE            ' PIC X(33).      UNIMPLEMENTED
1941      05 VALUE ' NORMAL          ' PIC X(33).
1942      05 VALUE ' NOT              ' PIC X(33).
1943      05 VALUE ' NULL            ' PIC X(33).
1944      05 VALUE ' NULLS           ' PIC X(33).
1945      05 VALUE ' NUMBER          ' PIC X(33).
1946      05 VALUE ' INUMBER-OF-CALL-PARAMETERS ' PIC X(33).
1947      05 VALUE ' NUMBERS         ' PIC X(33).
1948      05 VALUE ' NUMERIC         ' PIC X(33).
1949 GC0711 05 VALUE ' INUMERIC-DECIMAL-POINT ' PIC X(33).
1950      05 VALUE ' NUMERIC-EDITED  ' PIC X(33).
1951 GC0711 05 VALUE ' INUMERIC-THOUSANDS-SEPARATOR ' PIC X(33).
1952      05 VALUE ' INUMVAL         ' PIC X(33).
1953      05 VALUE ' INUMVAL-C       ' PIC X(33).
1954 GC0711 05 VALUE ' INUMVAL-F      ' PIC X(33).
1955      05 VALUE ' OBJECT          ' PIC X(33).      UNIMPLEMENTED
1956      05 VALUE ' OBJECT-COMPUTER ' PIC X(33).
1957      05 VALUE ' OBJECT-REFERENCE ' PIC X(33).      UNIMPLEMENTED
1958      05 VALUE ' OCCURS          ' PIC X(33).
1959      05 VALUE ' OF              ' PIC X(33).
1960      05 VALUE ' OFF             ' PIC X(33).
1961      05 VALUE ' OMITTED        ' PIC X(33).
1962      05 VALUE ' ON              ' PIC X(33).
1963      05 VALUE ' ONLY           ' PIC X(33).
1964      05 VALUE ' VOPEN          ' PIC X(33).
1965      05 VALUE ' OPTIONAL       ' PIC X(33).
1966      05 VALUE ' OPTIONS        ' PIC X(33).      UNIMPLEMENTED
1967      05 VALUE ' OR             ' PIC X(33).
1968      05 VALUE ' IORD           ' PIC X(33).
1969      05 VALUE ' IORD-MAX       ' PIC X(33).
1970      05 VALUE ' IORD-MIN      ' PIC X(33).
1971      05 VALUE ' ORDER          ' PIC X(33).
1972      05 VALUE ' ORGANISATION    ' PIC X(33).
1973      05 VALUE ' ORGANIZATION   ' PIC X(33).
1974      05 VALUE ' OTHER         ' PIC X(33).
1975      05 VALUE ' OUTPUT         ' PIC X(33).
1976      05 VALUE ' OVERFLOW       ' PIC X(33).
1977      05 VALUE ' OVERLINE      ' PIC X(33).
1978      05 VALUE ' OVERRIDE      ' PIC X(33).
1979      05 VALUE ' PACKED-DECIMAL ' PIC X(33).
1980      05 VALUE ' PADDING        ' PIC X(33).
1981      05 VALUE ' PAGE           ' PIC X(33).
1982      05 VALUE ' PAGE-COUNTER   ' PIC X(33).
1983      05 VALUE ' PARAGRAPH      ' PIC X(33).
1984      05 VALUE ' VPERFORM       ' PIC X(33).
1985      05 VALUE ' PF            ' PIC X(33).
1986      05 VALUE ' PH            ' PIC X(33).
1987      05 VALUE ' IPI           ' PIC X(33).
1988      05 VALUE ' KPIC          ' PIC X(33).
1989      05 VALUE ' KPICTURE      ' PIC X(33).
1990      05 VALUE ' PLUS          ' PIC X(33).
1991      05 VALUE ' KPOINTER      ' PIC X(33).
1992      05 VALUE ' POSITION        ' PIC X(33).
=====

```

Line Statement

Page: 46

```

=====
1993      05 VALUE ' POSITIVE          ' PIC X(33).
1994      05 VALUE ' PREFIXED         ' PIC X(33).      UNIMPLEMENTED
1995      05 VALUE ' PRESENT          ' PIC X(33).
1996      05 VALUE ' IPRESENT-VALUE   ' PIC X(33).
1997      05 VALUE ' PREVIOUS         ' PIC X(33).
1998      05 VALUE ' MPRINTER         ' PIC X(33).
1999      05 VALUE ' PRINTING         ' PIC X(33).
2000      05 VALUE ' KPROCEDURE       ' PIC X(33).
2001      05 VALUE ' PROCEDURE-POINTER ' PIC X(33).
2002      05 VALUE ' PROCEDURES      ' PIC X(33).
2003      05 VALUE ' PROCEED         ' PIC X(33).
2004      05 VALUE ' PROGRAM         ' PIC X(33).
2005      05 VALUE ' KPROGRAM-ID      ' PIC X(33).
2006      05 VALUE ' PROGRAM-POINTER ' PIC X(33).
2007      05 VALUE ' PROHIBITED      ' PIC X(33).
2008      05 VALUE ' PROMPT          ' PIC X(33).
2009      05 VALUE ' PROPERTY         ' PIC X(33).      UNIMPLEMENTED
2010      05 VALUE ' PROTOTYPE       ' PIC X(33).      UNIMPLEMENTED
2011      05 VALUE ' PURGE           ' PIC X(33).      OBSOLETE
2012      05 VALUE ' QUEUE          ' PIC X(33).      OBSOLETE
2013      05 VALUE ' QUOTE          ' PIC X(33).
2014      05 VALUE ' QUOTES         ' PIC X(33).
2015      05 VALUE ' RAISE          ' PIC X(33).      UNIMPLEMENTED
2016      05 VALUE ' RAISING        ' PIC X(33).      UNIMPLEMENTED
2017      05 VALUE ' IRANDOM        ' PIC X(33).
2018      05 VALUE ' IRANGE         ' PIC X(33).
2019      05 VALUE ' RD             ' PIC X(33).
2020      05 VALUE ' VREAD          ' PIC X(33).
2021      05 VALUE ' VREADY         ' PIC X(33).
2022      05 VALUE ' VRECEIVE       ' PIC X(33).      OBSOLETE
2023      05 VALUE ' RECORD         ' PIC X(33).
2024      05 VALUE ' RECORDING      ' PIC X(33).
2025      05 VALUE ' RECORDS        ' PIC X(33).
2026      05 VALUE ' RECURSIVE     ' PIC X(33).
2027      05 VALUE ' KREDEFINES     ' PIC X(33).
2028      05 VALUE ' REEL           ' PIC X(33).
2029 GC0712 05 VALUE ' AREFERENCE       ' PIC X(33).
2030      05 VALUE ' REFERENCES      ' PIC X(33).
2031      05 VALUE ' RELATION        ' PIC X(33).      UNIMPLEMENTED
2032      05 VALUE ' RELATIVE       ' PIC X(33).
2033      05 VALUE ' VRELEASE       ' PIC X(33).
2034      05 VALUE ' IREM           ' PIC X(33).
2035      05 VALUE ' REMAINDER      ' PIC X(33).
2036      05 VALUE ' REMARKS        ' PIC X(33).      OBSOLETE
2037      05 VALUE ' REMOVAL        ' PIC X(33).
2038      05 VALUE ' KRENAMES       ' PIC X(33).
2039      05 VALUE ' REPLACE        ' PIC X(33).
2040      05 VALUE ' KREPLACING     ' PIC X(33).
2041      05 VALUE ' KREPORT        ' PIC X(33).
2042      05 VALUE ' REPORTING      ' PIC X(33).
2043      05 VALUE ' REPORTS        ' PIC X(33).
2044      05 VALUE ' REPOSITORY     ' PIC X(33).
2045      05 VALUE ' REQUIRED        ' PIC X(33).
2046      05 VALUE ' RESERVE       ' PIC X(33).
=====

```

```

=====
2047      05 VALUE 'VRESET                ' PIC X(33).
2048      05 VALUE ' RESUME                ' PIC X(33).      UNIMPLEMENTED
2049      05 VALUE ' RETRY                  ' PIC X(33).      UNIMPLEMENTED
2050      05 VALUE 'VRETURN                ' PIC X(33).
2051      05 VALUE ' IRETURN-CODE          ' PIC X(33).
2052      05 VALUE ' KRETURNING            ' PIC X(33).
2053      05 VALUE ' IREVERSE              ' PIC X(33).
2054      05 VALUE ' REVERSE-VIDEO        ' PIC X(33).
2055      05 VALUE ' REVERSED              ' PIC X(33).
2056      05 VALUE ' REWIND                ' PIC X(33).
2057      05 VALUE 'VREWRITE              ' PIC X(33).
2058      05 VALUE ' RF                    ' PIC X(33).
2059      05 VALUE ' RH                    ' PIC X(33).
2060      05 VALUE ' RIGHT                  ' PIC X(33).
2061      05 VALUE ' RIGHT-JUSTIFY        ' PIC X(33).      UNIMPLEMENTED
2062      05 VALUE 'VROLLBACK              ' PIC X(33).
2063      05 VALUE ' ROUNDED                ' PIC X(33).
2064      05 VALUE ' ROUNDING              ' PIC X(33).      UNIMPLEMENTED
2065      05 VALUE ' RUN                   ' PIC X(33).
2066      05 VALUE ' SAME                   ' PIC X(33).
2067      05 VALUE 'KSCREEN                 ' PIC X(33).
2068      05 VALUE ' SCROLL                 ' PIC X(33).
2069      05 VALUE ' KSD                    ' PIC X(33).
2070      05 VALUE 'VSEARCH                 ' PIC X(33).
2071      05 VALUE ' SECONDS                ' PIC X(33).      UNIMPLEMENTED
2072      05 VALUE ' ISECONDS-FROM-FORMATTED-TIME ' PIC X(33).
2073      05 VALUE ' ISECONDS-PAST-MIDNIGHT ' PIC X(33).
2074      05 VALUE 'KSECTION                ' PIC X(33).
2075      05 VALUE ' SECURE                  ' PIC X(33).
2076      05 VALUE ' SECURITY                ' PIC X(33).      OBSOLETE
2077      05 VALUE ' SEGMENT                 ' PIC X(33).      OBSOLETE
2078      05 VALUE ' SEGMENT-LIMIT          ' PIC X(33).
2079      05 VALUE ' SELECT                  ' PIC X(33).
2080      05 VALUE ' SELF                    ' PIC X(33).      UNIMPLEMENTED
2081      05 VALUE 'VSEND                   ' PIC X(33).      OBSOLETE
2082      05 VALUE ' SENTENCE                ' PIC X(33).
2083      05 VALUE ' SEPARATE                ' PIC X(33).
2084      05 VALUE ' SEQUENCE                ' PIC X(33).
2085      05 VALUE ' SEQUENTIAL             ' PIC X(33).
2086      05 VALUE 'VSET                    ' PIC X(33).
2087      05 VALUE ' SHARING                 ' PIC X(33).
2088      05 VALUE ' ISIGN                   ' PIC X(33).
2089      05 VALUE ' SIGN                    ' PIC X(33).
2090      05 VALUE ' SIGNED                   ' PIC X(33).
2091      05 VALUE ' SIGNED-INT              ' PIC X(33).
2092      05 VALUE ' SIGNED-LONG            ' PIC X(33).
2093      05 VALUE ' SIGNED-SHORT           ' PIC X(33).
2094      05 VALUE ' ISIN                    ' PIC X(33).
2095      05 VALUE ' SIZE                    ' PIC X(33).
2096      05 VALUE 'VSORT                   ' PIC X(33).
2097      05 VALUE ' SORT-MERGE              ' PIC X(33).
2098      05 VALUE ' ISORT-RETURN            ' PIC X(33).
2099      05 VALUE ' SOURCE                  ' PIC X(33).
2100      05 VALUE ' SOURCE-COMPUTER        ' PIC X(33).
=====

```

Line Statement

Page: 48

```

=====
2101      05 VALUE ' SOURCES          ' PIC X(33).      UNIMPLEMENTED
2102      05 VALUE ' SPACE           ' PIC X(33).
2103      05 VALUE ' SPACE-FILL      ' PIC X(33).      UNIMPLEMENTED
2104      05 VALUE ' SPACES          ' PIC X(33).
2105      05 VALUE ' SPECIAL-NAMES   ' PIC X(33).
2106      05 VALUE ' ISQRT           ' PIC X(33).
2107      05 VALUE ' STANDARD        ' PIC X(33).
2108      05 VALUE ' STANDARD-1      ' PIC X(33).
2109      05 VALUE ' STANDARD-2      ' PIC X(33).
2110      05 VALUE ' STANDARD-BINARY ' PIC X(33).      UNIMPLEMENTED
2111      05 VALUE ' ISTDANDARD-COMPARE ' PIC X(33).      UNIMPLEMENTED
2112      05 VALUE ' STANDARD-DECIMAL ' PIC X(33).      UNIMPLEMENTED
2113      05 VALUE ' ISTDANDARD-DEVIATION ' PIC X(33).
2114      05 VALUE ' VSTART          ' PIC X(33).
2115      05 VALUE ' STATEMENT       ' PIC X(33).      UNIMPLEMENTED
2116      05 VALUE ' STATIC          ' PIC X(33).
2117      05 VALUE ' STATUS          ' PIC X(33).
2118      05 VALUE ' STDCALL         ' PIC X(33).
2119      05 VALUE ' MSTDERR         ' PIC X(33).
2120      05 VALUE ' MSTDIN         ' PIC X(33).
2121      05 VALUE ' MSTDOUT        ' PIC X(33).
2122      05 VALUE ' STEP           ' PIC X(33).
2123      05 VALUE ' VSTOP          ' PIC X(33).
2124      05 VALUE ' ISTORED-CHAR-LENGTH ' PIC X(33).
2125      05 VALUE ' VSTRING        ' PIC X(33).
2126      05 VALUE ' STRONG         ' PIC X(33).      UNIMPLEMENTED
2127      05 VALUE ' SUB-QUEUE-1     ' PIC X(33).      OBSOLETE
2128      05 VALUE ' SUB-QUEUE-2     ' PIC X(33).      OBSOLETE
2129      05 VALUE ' SUB-QUEUE-3     ' PIC X(33).      OBSOLETE
2130      05 VALUE ' ISUBSTITUTE     ' PIC X(33).
2131      05 VALUE ' ISUBSTITUTE-CASE ' PIC X(33).
2132      05 VALUE ' VSUBTRACT       ' PIC X(33).
2133      05 VALUE ' ISUM           ' PIC X(33).
2134      05 VALUE ' SUM            ' PIC X(33).
2135      05 VALUE ' SUPER          ' PIC X(33).      UNIMPLEMENTED
2136      05 VALUE ' VSUPPRESS      ' PIC X(33).
2137      05 VALUE ' MSW0          ' PIC X(33).
2138      05 VALUE ' MSW1          ' PIC X(33).
2139      05 VALUE ' MSW10         ' PIC X(33).
2140      05 VALUE ' MSW11         ' PIC X(33).
2141      05 VALUE ' MSW12         ' PIC X(33).
2142      05 VALUE ' MSW13         ' PIC X(33).
2143      05 VALUE ' MSW14         ' PIC X(33).
2144      05 VALUE ' MSW15         ' PIC X(33).
2145      05 VALUE ' MSW2          ' PIC X(33).
2146      05 VALUE ' MSW3          ' PIC X(33).
2147      05 VALUE ' MSW4          ' PIC X(33).
2148      05 VALUE ' MSW5          ' PIC X(33).
2149      05 VALUE ' MSW6          ' PIC X(33).
2150      05 VALUE ' MSW7          ' PIC X(33).
2151      05 VALUE ' MSW8          ' PIC X(33).
2152      05 VALUE ' MSW9          ' PIC X(33).
2153      05 VALUE ' MSWITCH-0     ' PIC X(33).
2154      05 VALUE ' MSWITCH-1     ' PIC X(33).
=====

```

Line Statement

Page: 49

```

=====
2155      05 VALUE 'MSWITCH-10      ' PIC X(33).
2156      05 VALUE 'MSWITCH-11      ' PIC X(33).
2157      05 VALUE 'MSWITCH-12      ' PIC X(33).
2158      05 VALUE 'MSWITCH-13      ' PIC X(33).
2159      05 VALUE 'MSWITCH-14      ' PIC X(33).
2160      05 VALUE 'MSWITCH-15      ' PIC X(33).
2161      05 VALUE 'MSWITCH-2       ' PIC X(33).
2162      05 VALUE 'MSWITCH-3       ' PIC X(33).
2163      05 VALUE 'MSWITCH-4       ' PIC X(33).
2164      05 VALUE 'MSWITCH-5       ' PIC X(33).
2165      05 VALUE 'MSWITCH-6       ' PIC X(33).
2166      05 VALUE 'MSWITCH-7       ' PIC X(33).
2167      05 VALUE 'MSWITCH-8       ' PIC X(33).
2168      05 VALUE 'MSWITCH-9       ' PIC X(33).
2169      05 VALUE 'SYMBOL          ' PIC X(33).      UNIMPLEMENTED
2170      05 VALUE 'SYMBOLIC        ' PIC X(33).
2171      05 VALUE 'SYNC            ' PIC X(33).
2172      05 VALUE 'SYNCHRONISED    ' PIC X(33).
2173      05 VALUE 'SYNCHRONIZED    ' PIC X(33).
2174      05 VALUE 'MSYSERR         ' PIC X(33).
2175      05 VALUE 'MSYSIN          ' PIC X(33).
2176      05 VALUE 'MSYSIPT         ' PIC X(33).
2177      05 VALUE 'MSYSLIST        ' PIC X(33).
2178      05 VALUE 'MSYSLST        ' PIC X(33).
2179      05 VALUE 'MSYSOUT         ' PIC X(33).
2180      05 VALUE 'SYSTEM-DEFAULT   ' PIC X(33).
2181      05 VALUE 'TABLE           ' PIC X(33).      UNIMPLEMENTED
2182      05 VALUE 'KTALLYING       ' PIC X(33).
2183      05 VALUE 'ITAN            ' PIC X(33).
2184      05 VALUE 'TAPE            ' PIC X(33).
2185      05 VALUE 'TERMINAL        ' PIC X(33).      OBSOLETE
2186      05 VALUE 'VTERMINATE      ' PIC X(33).
2187      05 VALUE 'TEST            ' PIC X(33).
2188      05 VALUE 'ITEST-DATE-YYYYMMDD ' PIC X(33).
2189      05 VALUE 'ITEST-DAY-YYYYDDD  ' PIC X(33).
2190 GC0711 05 VALUE 'ITEST-FORMATTED-DATETIME ' PIC X(33).      UNIMPLEMENTED
2191 GC0711 05 VALUE 'ITEST-NUMVAL      ' PIC X(33).
2192 GC0711 05 VALUE 'ITEST-NUMVAL-C   ' PIC X(33).
2193 GC0711 05 VALUE 'ITEST-NUMVAL-F   ' PIC X(33).
2194      05 VALUE 'TEXT            ' PIC X(33).      OBSOLETE
2195      05 VALUE 'THAN            ' PIC X(33).
2196      05 VALUE 'THEN            ' PIC X(33).
2197      05 VALUE 'THROUGH         ' PIC X(33).
2198      05 VALUE 'THRU           ' PIC X(33).
2199      05 VALUE 'TIME            ' PIC X(33).
2200 GC0711 05 VALUE 'TIME-OUT        ' PIC X(33).
2201 GC0711 05 VALUE 'TIMEOUT        ' PIC X(33).
2202      05 VALUE 'TIMES           ' PIC X(33).
2203      05 VALUE 'KTO            ' PIC X(33).
2204      05 VALUE 'TOP            ' PIC X(33).
2205      05 VALUE 'TOWARD-GREATER    ' PIC X(33).
2206      05 VALUE 'TOWARD-LESSER    ' PIC X(33).
2207      05 VALUE 'TRAILING        ' PIC X(33).
2208      05 VALUE 'TRAILING-SIGN    ' PIC X(33).      UNIMPLEMENTED
=====

```

Line Statement

Page: 50

```

=====
2209      05 VALUE 'VTRANSFORM          ' PIC X(33).
2210      05 VALUE 'ITRIM                ' PIC X(33).
2211      05 VALUE ' TRUE                ' PIC X(33).
2212      05 VALUE ' TRUNCATION          ' PIC X(33).
2213      05 VALUE ' TYPE                 ' PIC X(33).
2214      05 VALUE ' TYPEDEF              ' PIC X(33).      UNIMPLEMENTED
2215      05 VALUE ' UCS-4                ' PIC X(33).      UNIMPLEMENTED
2216      05 VALUE ' UNDERLINE           ' PIC X(33).
2217      05 VALUE ' UNIT                 ' PIC X(33).
2218      05 VALUE ' UNIVERSAL            ' PIC X(33).      UNIMPLEMENTED
2219      05 VALUE 'VUNLOCK               ' PIC X(33).
2220      05 VALUE ' UNSIGNED              ' PIC X(33).
2221      05 VALUE ' UNSIGNED-INT         ' PIC X(33).
2222      05 VALUE ' UNSIGNED-LONG        ' PIC X(33).
2223      05 VALUE ' UNSIGNED-SHORT      ' PIC X(33).
2224      05 VALUE 'VUNSTRING             ' PIC X(33).
2225      05 VALUE ' UNTIL                ' PIC X(33).
2226      05 VALUE 'KUP                   ' PIC X(33).
2227      05 VALUE ' UPDATE               ' PIC X(33).
2228      05 VALUE ' UPON                  ' PIC X(33).
2229      05 VALUE ' UPPER                 ' PIC X(33).
2230      05 VALUE 'IUPPER-CASE           ' PIC X(33).
2231      05 VALUE ' USAGE                ' PIC X(33).
2232      05 VALUE 'VUSE                  ' PIC X(33).
2233 GC0711 05 VALUE ' USER                ' PIC X(33).
2234      05 VALUE ' USER-DEFAULT          ' PIC X(33).
2235      05 VALUE 'KUSING                 ' PIC X(33).
2236      05 VALUE ' UTF-16                ' PIC X(33).      UNIMPLEMENTED
2237      05 VALUE ' UTF-8                 ' PIC X(33).      UNIMPLEMENTED
2238      05 VALUE ' VAL-STATUS            ' PIC X(33).      UNIMPLEMENTED
2239      05 VALUE ' VALID                 ' PIC X(33).      UNIMPLEMENTED
2240      05 VALUE ' VALIDATE              ' PIC X(33).      UNIMPLEMENTED
2241      05 VALUE ' VALIDATE-STATUS       ' PIC X(33).      UNIMPLEMENTED
2242 GC0712 05 VALUE 'AVALUE               ' PIC X(33).
2243      05 VALUE ' VALUES                ' PIC X(33).
2244      05 VALUE 'IVARIANCE              ' PIC X(33).
2245      05 VALUE 'KVARYING                ' PIC X(33).
2246      05 VALUE ' VDISABLE              ' PIC X(33).      UNIMPLEMENTED
2247      05 VALUE ' WAIT                  ' PIC X(33).
2248      05 VALUE 'VWHEN                  ' PIC X(33).
2249      05 VALUE 'IWHEN-COMPILED         ' PIC X(33).
2250      05 VALUE ' WITH                  ' PIC X(33).
2251      05 VALUE ' WORDS                  ' PIC X(33).
2252      05 VALUE 'KWORKING-STORAGE       ' PIC X(33).
2253      05 VALUE 'VWRITE                  ' PIC X(33).
2254      05 VALUE 'IYEAR-TO-YYYY         ' PIC X(33).
2255      05 VALUE ' YYYYDDD                ' PIC X(33).
2256      05 VALUE ' YYYYMMDD              ' PIC X(33).
2257      05 VALUE ' ZERO                   ' PIC X(33).
2258      05 VALUE ' ZERO-FILL              ' PIC X(33).      UNIMPLEMENTED
2259      05 VALUE ' ZEROES                 ' PIC X(33).
2260      05 VALUE ' ZEROS                  ' PIC X(33).
2261      01 WS-Reserved-Word-Table-TXT REDEFINES WS-Reserved-Words-TXT.
2262 GC1113 05 WS-Reserved-Word-TXT OCCURS 756 TIMES

```


Line Statement

```

=====
2263                                     ASCENDING KEY
2264                                     WS-RW-Word-TXT
2265                                     INDEXED WS-RW-IDX.
2266             10 WS-RW-Type-CD          PIC X(1).
2267             10 WS-RW-Word-TXT        PIC X(32).
2268
2269 01 WS-Runtime-Switches.
2270 GC0710 05 WS-RS-Duplicate-CHR       PIC X(1).
2271             05 WS-RS-In-Which-Pgm-CHR PIC X(1).
2272             88 WS-RS-In-Main-Module-BOOL VALUE 'M'.
2273             88 WS-RS-In-Copybook-BOOL VALUE 'C'.
2274             05 WS-RS-Last-Token-Ended-Sent-CHR PIC X(1).
2275             05 WS-RS-Processing-PICTURE-CHR PIC X(1).
2276             05 WS-RS-Token-Ended-Sentence-CHR PIC X(1).
2277 GC0710 05 WS-RS-Verb-Has-Been-Found-CHR PIC X(1).
2278
2279 01 WS-Saved-Section-TXT              PIC X(15).
2280
2281 01 WS-Src-Detail-Line-TXT.
2282             05 WS-SDL-Line-NUM        PIC ZZZZZ9.
2283             05 FILLER                  PIC X(1).
2284             05 WS-SDL-Statement-TXT    PIC X(128).
2285
2286 01 WS-Src-Header-1-TXT.
2287 GC0712 05 WS-SH1-Title-TXT          PIC X(125).
2288             05 WS-SH1-DT               PIC 9999/99/99.
2289
2290 01 WS-Src-Header-2-TXT              PIC X(135).
2291
2292 GC0712 01 WS-Src-Header-3-TXT.
2293 GC0712 05 VALUE 'Line Statement'     PIC X(125).
2294 GC0712 05 WS-SH3-Page-No-TXT        PIC X(10).
2295
2296 GC0712 01 WS-Src-Header-4-TXT.
2297 GC0712 05 VALUE '====='             PIC X(7).
2298 GC0712 05 VALUE ALL '='             PIC X(128).
2299
2300 01 WS-Src-Line-NUM                  PIC 9(6).
2301
2302 01 WS-Src-SUB                       USAGE BINARY-LONG.
2303
2304 GC0712 01 WS-Argument-Type-CD       PIC X(1).
2305 GC0712 88 WS-Argument-Is-Updatable-BOOL VALUE 'U' FALSE ' '.
2306
2307 01 WS-Tally-QTY                     USAGE BINARY-LONG.
2308
2309 01 WS-Temp-10-Chars-TXT              PIC X(10).
2310
2311 01 WS-Temp-32-Chars-1-TXT           PIC X(32).
2312
2313 GC0711 01 WS-Temp-32-Chars-2-TXT    PIC X(32).
2314
2315 GC0711 01 WS-Temp-32-Chars-3-TXT    PIC X(32).
2316
=====

```

Line Statement

Page: 52

```

=====
2317 GC0712 01 WS-Temp-65-Chars-TXT          PIC X(65).
2318
2319          01 WS-Temp-256-Chars-TXT      PIC X(256).
2320
2321          01 WS-Today-DT                  PIC 9(8).
2322
2323          01 WS-Token-Curr-TXT           PIC X(32).
2324
2325          01 WS-Token-Curr-Uc-TXT        PIC X(32).
2326
2327          01 WS-Token-Prev-TXT           PIC X(32).
2328
2329          01 WS-Token-Search-TXT         PIC X(32).
2330
2331          01 WS-Token-Type-CD             PIC X(1).
2332 GC0712      88 WS-TT-Token-Is-Argtype-BOOL VALUE 'A'.
2333              88 WS-TT-Token-Is-EOF-BOOL  VALUE HIGH-VALUES.
2334              88 WS-TT-Token-Is-Identifier-BOOL VALUE 'I'.
2335 GC0712      88 WS-TT-Token-Is-Keyword-BOOL VALUE 'K', 'V', 'A'.
2336              88 WS-TT-Token-Is-Lit-Alpha-BOOL VALUE 'L'.
2337              88 WS-TT-Token-Is-Lit-Number-BOOL VALUE 'N'.
2338              88 WS-TT-Token-Is-Verb-BOOL  VALUE 'V'.
2339 GC0710      88 WS-TT-Token-Is-Reserved-Wd-BOOL VALUE ' '.
2340
2341          01 WS-Usernames-QTY             USAGE BINARY-LONG.
2342
2343          01 WS-Xref-Detail-Line-TXT.
2344              05 WS-XDL-Prog-ID-TXT      PIC X(15).
2345              05 FILLER                    PIC X(1).
2346              05 WS-XDL-Token-TXT         PIC X(32).
2347              05 FILLER                    PIC X(1).
2348              05 WS-XDL-Def-Line-NUM     PIC ZZZZZ9.
2349              05 FILLER                    PIC X(1).
2350              05 WS-XDL-Section-TXT      PIC X(15).
2351              05 FILLER                    PIC X(1).
2352              05 WS-XDL-Reference-TXT     OCCURS WS-Lines-Per-Rec-CONST.
2353                  10 WS-XDL-Ref-Line-NUM PIC ZZZZZ9.
2354                  10 WS-XDL-Ref-Flag-CHR PIC X(1).
2355                  10 FILLER                PIC X(1).
2356
2357          01 WS-Xref-Header-1-TXT.
2358 GC0712      05 WS-XH1-Title-TXT         PIC X(125).
2359              05 WS-XH1-DT                 PIC 9999/99/99.
2360
2361          01 WS-Xref-Header-2-TXT         PIC X(135).
2362
2363 GC0712 01 WS-Xref-Header-3-TXT.
2364 GC0712      05 VALUE 'PROGRAM-ID'        PIC X(16).
2365 GC0712      05 VALUE 'Identifier/Register/' PIC X(20).
2366 GC0712      05 VALUE 'Function'          PIC X(13).
2367 GC0712      05 VALUE 'Defn'              PIC X(7).
2368 GC0712      05 VALUE 'Where Defined'     PIC X(16).
2369 GC0712      05 VALUE 'References (* = Updated)' PIC X(53).
2370 GC0712      05 WS-XH3-Page-No-TXT       PIC X(10).
=====

```

Line Statement

Page: 53

```
=====
2371
2372 GC0712 01 WS-Xref-Header-4-TXT.
2373 GC0712 05 VALUE ALL '=' PIC X(15).
2374 GC0712 05 VALUE SPACE PIC X(1).
2375 GC0712 05 VALUE ALL '=' PIC X(32).
2376 GC0712 05 VALUE SPACE PIC X(1).
2377 GC0712 05 VALUE ALL '=' PIC X(6).
2378 GC0712 05 VALUE SPACE PIC X(1).
2379 GC0712 05 VALUE ALL '=' PIC X(15).
2380 GC0712 05 VALUE SPACE PIC X(1).
2381 GC0712 05 VALUE ALL '=' PIC X(63).
2382
2383 LINKAGE SECTION.
2384 GC0712 01 L-Listing-Fn-TXT PIC X(256).
2385
2386 01 L-Src-Fn-TXT PIC X(256).
2387
2388 GC0712 01 L-OS-Type-CD PIC 9(1).
```

Line Statement

Page: 54

```
=====
2389 /
2390 GC0712 PROCEDURE DIVISION USING L-Listing-Fn-TXT
2391 GC0712 L-Src-Fn-TXT
2392 GC0712 L-OS-Type-CD.
2393 000-Main SECTION.
2394 PERFORM 100-Initialization
2395 GC0712 OPEN OUTPUT F-Listing-FILE
2396 GC0712 PERFORM 500-Produce-Source-Listing
2397 GC0712 SORT F-Sort-Work-FILE
2398 GC0712 ASCENDING KEY F-SW-Prog-ID-TXT
2399 GC0712 F-SW-Token-Uc-TXT
2400 GC0712 F-SW-Ref-Line-NUM
2401 GC0712 INPUT PROCEDURE 300-Tokenize-Source
2402 GC0712 OUTPUT PROCEDURE 400-Produce-Xref-Listing
2403 GC0712 CLOSE F-Listing-FILE
2404 GOBACK
2405 .
```

Line Statement

Page: 55

```

=====
2406 /
2407 *>*****
2408 *> Perform all program-wide initialization operations **
2409 *>*****
2410 100-Initialization SECTION.
2411 GC0712 MOVE 0 TO WS-Page-NUM
2412 GC0712 STRING 'GNU COBOL V2.0 11FEB2012 Source Listing - GCic for '
2413 GC0712 DELIMITED SIZE
2414 GC0712 WS-OS-Type-TXT(L-OS-Type-CD) DELIMITED SPACE
2415 GC0712 ' Copyright (C) 2009 - 2013, Gary L. Cutler, GPL'
2416 GC0712 DELIMITED SIZE
2417 GC0712 INTO WS-SH1-Title-TXT
2418 GC0712 STRING 'GNU COBOL V2.0 11FEB2012 Cross-Reference Listing -' &
2419 GC0712 ' GCic for ' DELIMITED SIZE
2420 GC0712 WS-OS-Type-TXT(L-OS-Type-CD) DELIMITED SPACE
2421 GC0712 ' Copyright (C) 2009 - 2013, Gary L. Cutler, GPL'
2422 GC0712 DELIMITED SIZE
2423 GC0712 INTO WS-XH1-Title-TXT
2424 MOVE TRIM(L-Src-Fn-TXT,Leading) TO L-Src-Fn-TXT
2425 GC1010 PERFORM VARYING WS-I-SUB FROM LENGTH(L-Src-Fn-TXT) BY -1 *> Locate last directory delimiter character so that the fil
ename can be extracted
2426 GC1010 UNTIL L-Src-Fn-TXT(WS-I-SUB:1) = '/' OR '\\'
2427 GC1010 OR WS-I-SUB = 0
2428 GC1010 END-PERFORM
2429 GC1010 IF WS-I-SUB = 0
2430 GC1010 MOVE UPPER-CASE(L-Src-Fn-TXT) TO WS-Main-Module-Name-TXT *> No directory delimiter, whole thing is filename
2431 GC1010 ELSE
2432 GC1010 ADD 1 TO WS-I-SUB
2433 GC1010 MOVE UPPER-CASE(L-Src-Fn-TXT(WS-I-SUB:))
2434 GC1010 TO WS-Main-Module-Name-TXT *> Extract filename
2435 GC1010 END-IF
2436 ACCEPT WS-Lines-Per-Page-Env-TXT
2437 FROM ENVIRONMENT 'OCXREF_LINES'
2438 INSPECT L-Src-Fn-TXT REPLACING ALL '\\' BY '/'
2439 MOVE L-Src-Fn-TXT TO WS-Program-Path-TXT
2440 MOVE WS-Program-Path-TXT TO WS-Src-Header-2-TXT
2441 CALL 'C$JUSTIFY' USING WS-Src-Header-2-TXT, 'Right'
2442 MOVE WS-Src-Header-2-TXT TO WS-Xref-Header-2-TXT
2443 MOVE LENGTH(TRIM(L-Src-Fn-TXT,Trailing)) TO WS-I-SUB
2444 MOVE 0 TO WS-J-SUB
2445 PERFORM UNTIL L-Src-Fn-TXT(WS-I-SUB:1) = '/'
2446 OR WS-I-SUB = 0
2447 SUBTRACT 1 FROM WS-I-SUB
2448 ADD 1 TO WS-J-SUB
2449 END-PERFORM
2450 UNSTRING L-Src-Fn-TXT((WS-I-SUB + 1):WS-J-SUB)
2451 DELIMITED BY '.'
2452 INTO WS-Filename-TXT
2453 WS-Dummy-TXT
2454 GC1010 STRING
2455 GC1010 TRIM(WS-Filename-TXT,Trailing)
2456 GC1010 '.i'
2457 GC1010 DELIMITED SIZE
2458 GC1010 INTO WS-Expanded-Src-Fn-TXT
=====

```

Line Statement

```
=====
2459 GC1010 CALL 'CBL_CHECK_FILE_EXIST' USING WS-Expanded-Src-Fn-TXT
2460 GC1010 WS-Temp-256-Chars-TXT
2461 GC1010 IF RETURN-CODE NOT = 0
2462 GC1010 GOBACK
2463 GC1010 END-IF
2464 IF WS-Lines-Per-Page-Env-TXT NOT = SPACES
2465 MOVE NUMVAL(WS-Lines-Per-Page-Env-TXT)
2466 TO WS-Lines-Per-Page-NUM
2467 ELSE
2468 MOVE 58
2469 TO WS-Lines-Per-Page-NUM
2470 END-IF
2471 ACCEPT WS-Today-DT FROM DATE YYYYMMDD
2472 MOVE WS-Today-DT TO WS-XH1-DT
2473 WS-SH1-DT
2474 MOVE '?????????????...' TO WS-Curr-Prog-ID-TXT
2475 MOVE SPACES TO WS-Curr-Verb-TXT
2476 WS-HeId-Reference-TXT
2477 .
```

Line Statement

Page: 57

```

=====
2478      /
2479      300-Tokenize-Source SECTION.
2480          OPEN INPUT F-Expanded-Src-FILE
2481          MOVE SPACES TO F-Expanded-Src-REC
2482          MOVE 256 TO WS-Src-SUB
2483          MOVE 0 TO WS-Usernames-QTY
2484              WS-Curr-Line-NUM
2485          MOVE '?' TO WS-Curr-Division-TXT
2486 GC0710  MOVE 'N' TO WS-RS-Verb-Has-Been-Found-CHR
2487          PERFORM FOREVER
2488              PERFORM 310-Get-Token
2489              IF WS-TT-Token-Is-EOF-BOOL
2490                  EXIT PERFORM
2491              END-IF
2492              MOVE UPPER-CASE(WS-Token-Curr-TXT)
2493                  TO WS-Token-Curr-Uc-TXT
2494 GC1010  IF WS-TT-Token-Is-Keyword-BOOL
2495 GC1010  OR WS-TT-Token-Is-Reserved-Wd-BOOL
2496 GC1010  MOVE WS-Token-Curr-Uc-TXT TO WS-Token-Curr-TXT
2497 GC1010  END-IF
2498          IF WS-TT-Token-Is-Verb-BOOL
2499              MOVE WS-Token-Curr-Uc-TXT TO WS-Curr-Verb-TXT
2500                  WS-Token-Prev-TXT
2501              IF WS-Held-Reference-TXT NOT = SPACES
2502                  MOVE WS-Held-Reference-TXT TO F-Sort-Work-REC
2503                  MOVE SPACES TO WS-Held-Reference-TXT
2504                  RELEASE F-Sort-Work-REC
2505              END-IF
2506          END-IF
2507          EVALUATE TRUE
2508          WHEN WS-CD-In-IDENT-DIV-BOOL
2509              PERFORM 320-IDENTIFICATION-DIVISION
2510          WHEN WS-CD-In-ENV-DIV-BOOL
2511              PERFORM 330-ENVIRONMENT-DIVISION
2512          WHEN WS-CD-In-DATA-DIV-BOOL
2513              PERFORM 340-DATA-DIVISION
2514          WHEN WS-CD-In-PROC-DIV-BOOL
2515              PERFORM 350-PROCEDURE-DIVISION
2516          END-EVALUATE
2517          IF WS-TT-Token-Is-Keyword-BOOL
2518              MOVE WS-Token-Curr-Uc-TXT TO WS-Token-Prev-TXT
2519          END-IF
2520          IF WS-RS-Token-Ended-Sentence-CHR = 'Y'
2521          AND WS-Curr-Division-TXT NOT = 'I'
2522              MOVE SPACES TO WS-Token-Prev-TXT
2523                  WS-Curr-Verb-TXT
2524          END-IF
2525
2526          END-PERFORM
2527          CLOSE F-Expanded-Src-FILE
2528      .
=====

```

Line Statement

Page: 58

```

=====
2529 /
2530 310-Get-Token SECTION.
2531 *>-- Position to 1st non-blank character
2532 MOVE WS-RS-Token-Ended-Sentence-CHR
2533 TO WS-RS-Last-Token-Ended-Sent-CHR
2534 MOVE 'N' TO WS-RS-Token-Ended-Sentence-CHR
2535 PERFORM UNTIL F-Expanded-Src-REC(WS-Src-SUB : 1) NOT = SPACE
2536 IF WS-Src-SUB > 255
2537 READ F-Expanded-Src-FILE AT END
2538 IF WS-Held-Reference-TXT NOT = SPACES
2539 MOVE WS-Held-Reference-TXT TO F-Sort-Work-REC
2540 MOVE SPACES TO WS-Held-Reference-TXT
2541 RELEASE F-Sort-Work-REC
2542 END-IF
2543 SET WS-TT-Token-Is-EOF-BOOL TO TRUE
2544 MOVE 0 TO WS-Curr-Line-NUM
2545 EXIT SECTION
2546 END-READ
2547 GC0712 IF F-ES-1-7-TXT NOT = '#DEFLIT'
2548 GC0712 IF F-ES-1-CHR = '#'
2549 GC0712 PERFORM 311-Control-Record
2550 GC0712 ELSE
2551 GC0712 PERFORM 312-Expanded-Src-Record
2552 GC0712 END-IF
2553 GC0712 END-IF
2554 ELSE
2555 ADD 1 TO WS-Src-SUB
2556 END-IF
2557 END-PERFORM
2558 *>-- Extract token string
2559 MOVE F-Expanded-Src-REC(WS-Src-SUB : 1)
2560 TO WS-Curr-CHR
2561 MOVE F-Expanded-Src-REC(WS-Src-SUB + 1: 1)
2562 TO WS-Next-CHR
2563 IF WS-Curr-CHR = '.'
2564 ADD 1 TO WS-Src-SUB
2565 MOVE WS-Curr-CHR TO WS-Token-Curr-TXT
2566 MOVE SPACE TO WS-Token-Type-CD
2567 MOVE 'Y' TO WS-RS-Token-Ended-Sentence-CHR
2568 EXIT SECTION
2569 END-IF
2570 IF WS-Curr-Char-Is-Punct-BOOL
2571 AND WS-Curr-CHR = '='
2572 AND WS-Curr-Division-TXT = 'P'
2573 ADD 1 TO WS-Src-SUB
2574 MOVE 'EQUALS' TO WS-Token-Curr-TXT
2575 MOVE 'K' TO WS-Token-Type-CD
2576 EXIT SECTION
2577 END-IF
2578 IF WS-Curr-Char-Is-Punct-BOOL *> So subscripts don't get flagged w/ '*'
2579 AND WS-Curr-CHR = '('
2580 AND WS-Curr-Division-TXT = 'P'
2581 MOVE SPACES TO WS-Token-Prev-TXT
2582 END-IF
=====

```


Line Statement

Page: 59

```

=====
2583      IF WS-Curr-Char-Is-Punct-BOOL
2584          ADD 1 TO WS-Src-SUB
2585          MOVE WS-Curr-CHR TO WS-Token-Curr-TXT
2586          MOVE SPACE TO WS-Token-Type-CD
2587          EXIT SECTION
2588      END-IF
2589      IF WS-Curr-Char-Is-Quote-BOOL
2590          ADD 1 TO WS-Src-SUB
2591          UNSTRING F-Expanded-Src-REC
2592              DELIMITED BY WS-Curr-CHR
2593              INTO WS-Token-Curr-TXT
2594              WITH POINTER WS-Src-SUB
2595          IF F-Expanded-Src-REC(WS-Src-SUB : 1) = '.'
2596              MOVE 'Y' TO WS-RS-Token-Ended-Sentence-CHR
2597              ADD 1 TO WS-Src-SUB
2598          END-IF
2599          SET WS-TT-Token-Is-Lit-Alpha-BOOL TO TRUE
2600          EXIT SECTION
2601      END-IF
2602      IF WS-Curr-Char-Is-X-BOOL AND WS-Next-Char-Is-Quote-BOOL
2603          ADD 2 TO WS-Src-SUB
2604          UNSTRING F-Expanded-Src-REC
2605              DELIMITED BY WS-Next-CHR
2606              INTO WS-Token-Curr-TXT
2607              WITH POINTER WS-Src-SUB
2608          IF F-Expanded-Src-REC(WS-Src-SUB : 1) = '.'
2609              MOVE 'Y' TO WS-RS-Token-Ended-Sentence-CHR
2610              ADD 1 TO WS-Src-SUB
2611          END-IF
2612          SET WS-TT-Token-Is-Lit-Number-BOOL TO TRUE
2613          EXIT SECTION
2614      END-IF
2615      IF WS-Curr-Char-Is-Z-BOOL AND WS-Next-Char-Is-Quote-BOOL
2616          ADD 2 TO WS-Src-SUB
2617          UNSTRING F-Expanded-Src-REC
2618              DELIMITED BY WS-Next-CHR
2619              INTO WS-Token-Curr-TXT
2620              WITH POINTER WS-Src-SUB
2621          IF F-Expanded-Src-REC(WS-Src-SUB : 1) = '.'
2622              MOVE 'Y' TO WS-RS-Token-Ended-Sentence-CHR
2623              ADD 1 TO WS-Src-SUB
2624          END-IF
2625          SET WS-TT-Token-Is-Lit-Alpha-BOOL TO TRUE
2626          EXIT SECTION
2627      END-IF
2628      IF WS-RS-Processing-PICTURE-CHR = 'Y'
2629          UNSTRING F-Expanded-Src-REC
2630              DELIMITED BY ' ' OR ','
2631              INTO WS-Token-Curr-TXT
2632              DELIMITER IN WS-Delim-TXT
2633              WITH POINTER WS-Src-SUB
2634          IF WS-Delim-TXT = ' '
2635              MOVE 'Y' TO WS-RS-Token-Ended-Sentence-CHR
2636              ADD 1 TO WS-Src-SUB

```

Line Statement

Page: 60

```

=====
2637         END-IF
2638         IF UPPER-CASE(WS-Token-Curr-TXT) = 'IS'
2639             MOVE SPACE TO WS-Token-Type-CD
2640             EXIT SECTION
2641         ELSE
2642             MOVE 'N' TO WS-RS-Processing-PICTURE-CHR
2643             MOVE SPACE TO WS-Token-Type-CD
2644             EXIT SECTION
2645         END-IF
2646     END-IF
2647     UNSTRING F-Expanded-Src-REC
2648         DELIMITED BY '.' OR ',' OR '=' OR '(' OR ')' OR '*'
2649             OR '/' OR '&' OR ';' OR ',' OR '<'
2650             OR '>' OR ':'
2651         INTO WS-Token-Curr-TXT
2652         DELIMITER IN WS-Delim-TXT
2653         WITH POINTER WS-Src-SUB
2654     IF WS-Delim-TXT = '.'
2655         MOVE 'Y' TO WS-RS-Token-Ended-Sentence-CHR
2656     END-IF
2657     IF WS-Delim-TXT NOT = '.' AND ' '
2658         SUBTRACT 1 FROM WS-Src-SUB
2659     END-IF
2660 *>-- Classify Token
2661     MOVE UPPER-CASE(WS-Token-Curr-TXT) TO WS-Token-Search-TXT
2662     IF WS-Token-Search-TXT = 'EQUAL' OR 'EQUALS'
2663         MOVE 'EQUALS' TO WS-Token-Curr-TXT
2664         MOVE 'K' TO WS-Token-Type-CD
2665         EXIT SECTION
2666     END-IF
2667     SEARCH ALL WS-Reserved-Word-TXT
2668         WHEN WS-RW-Word-TXT (WS-RW-IDX) = WS-Token-Search-TXT
2669             MOVE WS-RW-Type-CD (WS-RW-IDX) TO WS-Token-Type-CD
2670 GC0710     IF WS-TT-Token-Is-Verb-BOOL
2671 GC0710         MOVE 'Y' TO WS-RS-Verb-Has-Been-Found-CHR
2672 GC0710     END-IF
2673     EXIT SECTION
2674     END-SEARCH
2675 *>-- Not a reserved word, must be a user name
2676     SET WS-TT-Token-Is-Identifier-BOOL TO TRUE
2677     PERFORM 313-Check-For-Numeric-Token
2678     IF WS-TT-Token-Is-Lit-Number-BOOL
2679         IF (WS-RS-Last-Token-Ended-Sent-CHR = 'Y')
2680             AND (WS-Curr-Division-TXT = 'D')
2681             MOVE 'LEVEL #' TO WS-Token-Curr-TXT
2682             MOVE 'K' TO WS-Token-Type-CD
2683             EXIT SECTION
2684         ELSE
2685             EXIT SECTION
2686     END-IF
2687     END-IF
2688     .

```

Line Statement

```

=====
2689      /
2690      311-Control-Record SECTION.
2691          UNSTRING F-ES-2-256-TXT-256
2692              DELIMITED BY ''
2693              INTO WS-Temp-10-Chars-TXT
2694                  WS-Temp-256-Chars-TXT
2695                  WS-Dummy-TXT
2696          INSPECT WS-Temp-10-Chars-TXT REPLACING ALL '' BY SPACE
2697 GC0712      IF WS-Temp-10-Chars-TXT(1:4) = 'line'
2698 GC0712          MOVE SPACES TO WS-Temp-10-Chars-TXT(1:4)
2699 GC0712      END-IF
2700      COMPUTE WS-I-SUB = NUMVAL(WS-Temp-10-Chars-TXT) - 1
2701 GC1010      IF UPPER-CASE(TRIM(WS-Temp-256-Chars-TXT,Trailing)) =
2702 GC1010          TRIM(WS-Main-Module-Name-TXT)
2703              MOVE WS-I-SUB TO WS-Curr-Line-NUM
2704              SET WS-RS-In-Main-Module-BOOL TO TRUE
2705              IF WS-Saved-Section-TXT NOT = SPACES
2706                  MOVE WS-Saved-Section-TXT TO WS-Curr-Section-TXT
2707              END-IF
2708          ELSE
2709              SET WS-RS-In-Copybook-BOOL TO TRUE
2710              IF WS-Saved-Section-TXT = SPACES
2711                  MOVE WS-Curr-Section-TXT TO WS-Saved-Section-TXT
2712              END-IF
2713              MOVE LENGTH(TRIM(WS-Temp-256-Chars-TXT,Trailing))
2714                  TO WS-I-SUB
2715              MOVE 0 TO WS-J-SUB
2716              PERFORM UNTIL WS-Temp-256-Chars-TXT(WS-I-SUB:1) = '/'
2717                  OR WS-I-SUB = 0
2718                  SUBTRACT 1 FROM WS-I-SUB
2719                  ADD 1 TO WS-J-SUB
2720              END-PERFORM
2721              UNSTRING WS-Temp-256-Chars-TXT((WS-I-SUB + 1):WS-J-SUB)
2722                  DELIMITED BY '.'
2723                  INTO WS-Filename-TXT
2724                      WS-Dummy-TXT
2725              MOVE '[' TO WS-CS-1-CHR
2726              MOVE WS-Filename-TXT TO WS-CS-2-14-TXT
2727              IF WS-CS-11-14-TXT NOT = SPACES
2728                  MOVE '...' TO WS-CS-11-14-TXT
2729              END-IF
2730              MOVE ']' TO WS-CS-15-CHR
2731          END-IF
2732          MOVE SPACES TO F-Expanded-Src-REC *> Force another READ
2733          MOVE 256 TO WS-Src-SUB
2734          .
=====

```

Line Statement

Page: 62

```
=====
2735      /
2736      312-Expanded-Src-Record SECTION.
2737 GC0711  MOVE 2 TO WS-Src-SUB
2738          IF WS-RS-In-Main-Module-BOOL
2739          ADD 1 To WS-Curr-Line-NUM
2740          END-IF
2741          .
```

Line Statement

Page: 63

```

=====
2742      /
2743      313-Check-For-Numeric-Token SECTION.
2744      MOVE WS-Token-Curr-TXT TO WS-Temp-32-Chars-1-TXT
2745      INSPECT WS-Temp-32-Chars-1-TXT
2746 GC0711      CONVERTING '0123456789' TO SPACES
2747 GC0711      IF WS-Temp-32-Chars-1-TXT = SPACES          *> Simple Unsigned Integer
2748      SET WS-TT-Token-Is-Lit-Number-BOOL TO TRUE
2749      EXIT SECTION
2750      END-IF
2751 GC0711      MOVE SPACES TO WS-Temp-32-Chars-2-TXT
2752 GC0711      WS-Temp-32-Chars-3-TXT
2753 GC0711      WS-Dummy-TXT
2754 GC0711      UNSTRING WS-Temp-32-Chars-1-TXT
2755 GC0711      DELIMITED BY 'e' OR 'E'
2756 GC0711      INTO WS-Temp-32-Chars-2-TXT
2757 GC0711      WS-Temp-32-Chars-3-TXT
2758 GC0711      WS-Dummy-TXT
2759 GC0711      IF WS-Dummy-TXT NOT = SPACES          *> More than one 'E' - Not Numeric
2760 GC0711      EXIT SECTION
2761 GC0711      END-IF
2762 GC0711      IF WS-Temp-32-Chars-2-TXT(1:1) = '+' OR '-'
2763 GC0711      MOVE SPACE TO WS-Temp-32-Chars-2-TXT(1:1)
2764 GC0711      END-IF
2765 GC0711      IF WS-Temp-32-Chars-3-TXT(1:1) = '+' OR '-'
2766 GC0711      MOVE SPACE TO WS-Temp-32-Chars-3-TXT(1:1)
2767 GC0711      END-IF
2768      MOVE 0 TO WS-Tally-QTY
2769 GC0711      INSPECT WS-Temp-32-Chars-2-TXT
2770      TALLYING WS-Tally-QTY FOR ALL '.'
2771      IF WS-Tally-QTY = 1
2772 GC0711      INSPECT WS-Temp-32-Chars-2-TXT REPLACING ALL '.' BY SPACE
2773      END-IF
2774 GC0711      INSPECT WS-Temp-32-Chars-3-TXT
2775 GC0711      TALLYING WS-Tally-QTY FOR ALL '.'
2776 GC0711      IF WS-Tally-QTY = 1
2777 GC0711      INSPECT WS-Temp-32-Chars-3-TXT REPLACING ALL '.' BY SPACE
2778 GC0711      END-IF
2779 GC0711      IF WS-Temp-32-Chars-2-TXT = SPACES AND WS-Temp-32-Chars-3-TXT = SPACES
2780      SET WS-TT-Token-Is-Lit-Number-BOOL TO TRUE
2781      EXIT SECTION
2782      END-IF
2783      .

```

Line Statement

Page: 64

```
=====
2784      /
2785      320-IDENTIFICATION-DIVISION SECTION.
2786 GC0712      IF WS-TT-Token-Is-Argtype-BOOL
2787 GC0712          SET WS-TT-Token-Is-Reserved-Wd-BOOL TO TRUE
2788 GC0712      END-IF
2789 GC0710      MOVE 'N' TO WS-RS-Verb-Has-Been-Found-CHR
2790          IF WS-TT-Token-Is-Keyword-BOOL
2791          AND WS-Token-Curr-TXT = 'DIVISION'
2792          MOVE WS-Token-Prev-TXT TO WS-Curr-Division-TXT
2793          EXIT SECTION
2794      END-IF
2795 GC0712      IF WS-Token-Prev-TXT = 'PROGRAM-ID' OR 'FUNCTION-ID'
2796          MOVE SPACES TO WS-Token-Prev-TXT
2797          MOVE WS-Token-Curr-TXT TO WS-Curr-Prog-ID-TXT
2798 GC0712          IF WS-CPI-16-CHR NOT = SPACES
2799              MOVE '...' TO WS-CPI-13-15-TXT
2800          END-IF
2801 GC0712          SEARCH ALL WS-Reserved-Word-TXT
2802              WHEN WS-RW-Word-TXT (WS-RW-IDX) = 'LENGTH'
2803              MOVE ' ' TO WS-RW-Type-CD (WS-RW-IDX)
2804 GC0712          END-SEARCH
2805          EXIT SECTION
2806      END-IF
2807      .
=====
```

Line Statement

Page: 65

```
=====
2808      /
2809      330-ENVIRONMENT-DIVISION SECTION.
2810 GC0712      IF WS-TT-Token-Is-Argtype-BOOL
2811 GC0712          SET WS-TT-Token-Is-Reserved-Wd-BOOL TO TRUE
2812 GC0712      END-IF
2813          IF WS-TT-Token-Is-Keyword-BOOL
2814          AND WS-Token-Curr-TXT = 'DIVISION'
2815          MOVE WS-Token-Prev-TXT TO WS-Curr-Division-TXT
2816          EXIT SECTION
2817      END-IF
2818          IF WS-TT-Token-Is-Keyword-BOOL
2819          AND WS-Token-Curr-TXT = 'SECTION'
2820          MOVE WS-Token-Prev-TXT TO WS-Curr-Section-TXT
2821          EXIT SECTION
2822      END-IF
2823          IF WS-TT-Token-Is-Identifier-BOOL
2824 GC0712          IF WS-Token-Prev-TXT = 'FUNCTION'
2825 GC0712              PERFORM 360-Release-Def
2826 GC0712          ELSE
2827 GC0712              PERFORM 361-Release-Ref
2828 GC0712          END-IF
2829      END-IF
2830      .
```

Line Statement

Page: 66

```
=====
2831 /
2832 340-DATA-DIVISION SECTION.
2833 GC0712 IF WS-TT-Token-Is-Argtype-BOOL
2834 GC0712 SET WS-TT-Token-Is-Reserved-Wd-BOOL TO TRUE
2835 GC0712 END-IF
2836 IF WS-TT-Token-Is-Keyword-BOOL
2837 AND WS-Token-Curr-TXT = 'DIVISION'
2838 GC0712 SEARCH ALL WS-Reserved-Word-TXT
2839 GC0712 WHEN WS-RW-Word-TXT (WS-RW-IDX) = 'LENGTH'
2840 GC0712 MOVE 'I' TO WS-RW-Type-CD (WS-RW-IDX)
2841 GC0712 END-SEARCH
2842 MOVE WS-Token-Prev-TXT TO WS-Curr-Division-TXT
2843 EXIT SECTION
2844 END-IF
2845 IF WS-TT-Token-Is-Keyword-BOOL
2846 AND WS-Token-Curr-TXT = 'SECTION'
2847 MOVE WS-Token-Prev-TXT TO WS-Curr-Section-TXT
2848 EXIT SECTION
2849 END-IF
2850 IF (WS-Token-Curr-TXT = 'PIC' OR 'PICTURE')
2851 AND (WS-TT-Token-Is-Keyword-BOOL)
2852 MOVE 'Y' TO WS-RS-Processing-PICTURE-CHR
2853 EXIT SECTION
2854 END-IF
2855 GC0710 IF WS-TT-Token-Is-Reserved-Wd-BOOL
2856 GC0710 AND WS-Token-Prev-TXT = 'LEVEL #'
2857 GC0710 MOVE SPACES TO WS-Token-Prev-TXT
2858 GC0710 EXIT SECTION
2859 GC0710 END-IF
2860 IF WS-TT-Token-Is-Identifier-BOOL
2861 EVALUATE WS-Token-Prev-TXT
2862 WHEN 'FD'
2863 PERFORM 360-Release-Def
2864 MOVE SPACES TO WS-Token-Prev-TXT
2865 WHEN 'SD'
2866 PERFORM 360-Release-Def
2867 MOVE SPACES TO WS-Token-Prev-TXT
2868 WHEN 'LEVEL #'
2869 PERFORM 360-Release-Def
2870 MOVE SPACES TO WS-Token-Prev-TXT
2871 WHEN 'INDEXED'
2872 PERFORM 360-Release-Def
2873 MOVE SPACES TO WS-Token-Prev-TXT
2874 WHEN 'USING'
2875 PERFORM 362-Release-Upd
2876 MOVE SPACES TO WS-Token-Prev-TXT
2877 WHEN 'INTO'
2878 PERFORM 362-Release-Upd
2879 MOVE SPACES TO WS-Token-Prev-TXT
2880 WHEN OTHER
2881 PERFORM 361-Release-Ref
2882 END-EVALUATE
2883 EXIT SECTION
2884 END-IF
=====
```


GNU COBOL V2.0 11FEB2012 Source Listing - GCic for Windows/MinGW Copyright (C) 2009 - 2013, Gary L. Cutler, GPL 2013/11/21
E:/GNU-COBOL/samples/GCic.cbl Page: 67

Line Statement

=====
2885 .

```

Line  Statement
=====
2886      /
2887      350-PROCEDURE-DIVISION SECTION.
2888          IF WS-Curr-Section-TXT NOT = 'PROCEDURE'
2889              MOVE 'PROCEDURE' TO WS-Curr-Section-TXT
2890          END-IF
2891 GC0710      IF WS-Token-Curr-Uc-TXT = 'PROGRAM'
2892 GC0710      AND WS-Token-Prev-TXT = 'END'
2893 GC0710      MOVE '?' TO WS-Curr-Division-TXT
2894 GC0710      EXIT SECTION
2895 GC0710      END-IF
2896          IF WS-TT-Token-Is-Keyword-BOOL
2897          AND WS-Token-Curr-TXT = 'DIVISION'
2898              MOVE WS-Token-Prev-TXT TO WS-Curr-Division-TXT
2899 GC0712      SEARCH ALL WS-Reserved-Word-TXT
2900 GC0712          WHEN WS-RW-Word-TXT (WS-RW-IDX) = 'LENGTH'
2901 GC0712              MOVE 'I' TO WS-RW-Type-CD (WS-RW-IDX)
2902 GC0712      END-SEARCH
2903          EXIT SECTION
2904      END-IF
2905 GC0313      IF WS-TT-Token-Is-Identifier-BOOL
2906 GC0313      AND WS-Token-Prev-TXT = SPACES
2907 GC0313      AND WS-Curr-Verb-TXT = SPACES
2908 GC0313*> ----- Definition of a Paragraph or Section
2909 GC0313      PERFORM 360-Release-Def
2910 GC0313      MOVE SPACES TO WS-Token-Prev-TXT
2911 GC0313      EXIT SECTION
2912 GC0313      END-IF
2913 GC0712      IF WS-Token-Curr-TXT = 'CALL'
2914 GC0712          SET WS-Argument-Is-Updatable-BOOL TO TRUE
2915 GC0712      END-IF
2916 GC0712      IF WS-Curr-Verb-TXT = 'CALL'
2917 GC0712          IF WS-TT-Token-Is-Argtype-BOOL
2918 GC0712              IF WS-Token-Curr-TXT = 'REFERENCE'
2919 GC0712                  SET WS-Argument-Is-Updatable-BOOL TO TRUE
2920 GC0712              ELSE
2921 GC0712                  SET WS-Argument-Is-Updatable-BOOL TO FALSE
2922 GC0712              END-IF
2923 GC0712          EXIT SECTION
2924 GC0712      END-IF
2925 GC0712      ELSE
2926 GC0712          SET WS-Argument-Is-Updatable-BOOL TO FALSE
2927 GC0712      END-IF
2928          IF NOT WS-TT-Token-Is-Identifier-BOOL
2929              EXIT SECTION
2930      END-IF
2931      EVALUATE WS-Curr-Verb-TXT
2932      WHEN 'ACCEPT'
2933          PERFORM 351-ACCEPT
2934      WHEN 'ADD'
2935          PERFORM 351-ADD
2936      WHEN 'ALLOCATE'
2937          PERFORM 351-ALLOCATE
2938      WHEN 'CALL'
2939          PERFORM 351-CALL

```

Line Statement

Page: 69

```
=====
2940      WHEN 'COMPUTE'
2941          PERFORM 351-COMPUTE
2942      WHEN 'DIVIDE'
2943          PERFORM 351-DIVIDE
2944      WHEN 'FREE'
2945          PERFORM 351-FREE
2946      WHEN 'INITIALIZE'
2947          PERFORM 351-INITIALIZE
2948      WHEN 'INSPECT'
2949          PERFORM 351-INSPECT
2950      WHEN 'MOVE'
2951          PERFORM 351-MOVE
2952      WHEN 'MULTIPLY'
2953          PERFORM 351-MULTIPLY
2954      WHEN 'PERFORM'
2955          PERFORM 351-PERFORM
2956      WHEN 'SET'
2957          PERFORM 351-SET
2958      WHEN 'STRING'
2959          PERFORM 351-STRING
2960      WHEN 'SUBTRACT'
2961          PERFORM 351-SUBTRACT
2962      WHEN 'TRANSFORM'
2963          PERFORM 351-TRANSFORM
2964      WHEN 'UNSTRING'
2965          PERFORM 351-UNSTRING
2966      WHEN OTHER
2967          PERFORM 361-Release-Ref
2968      END-EVALUATE
2969      .
```

Line Statement

Page: 70

```

=====
2970      /
2971      351-ACCEPT SECTION.
2972          EVALUATE WS-Token-Prev-TXT
2973          WHEN 'ACCEPT'
2974              PERFORM 362-Release-Upd
2975              MOVE SPACES TO WS-Token-Prev-TXT
2976          WHEN OTHER
2977              PERFORM 361-Release-Ref
2978          END-EVALUATE
2979          .
2980
2981      351-ADD SECTION.
2982          EVALUATE WS-Token-Prev-TXT
2983          WHEN 'GIVING'
2984              PERFORM 362-Release-Upd
2985          WHEN 'TO'
2986              PERFORM 362-Release-Upd
2987          WHEN OTHER
2988              PERFORM 361-Release-Ref
2989          END-EVALUATE
2990          .
2991
2992      351-ALLOCATE SECTION.
2993          EVALUATE WS-Token-Prev-TXT
2994          WHEN 'ALLOCATE'
2995              PERFORM 362-Release-Upd
2996              MOVE SPACES TO WS-Token-Prev-TXT
2997          WHEN 'RETURNING'
2998              PERFORM 362-Release-Upd
2999          WHEN OTHER
3000              PERFORM 361-Release-Ref
3001          END-EVALUATE
3002          .
3003
3004      351-CALL SECTION.
3005          EVALUATE WS-Token-Prev-TXT
3006          WHEN 'RETURNING'
3007              PERFORM 362-Release-Upd
3008          WHEN 'GIVING'
3009              PERFORM 362-Release-Upd
3010          WHEN OTHER
3011      GC0712      IF WS-Argument-Is-Updatable-BOOL
3012      GC0712          PERFORM 362-Release-Upd
3013      GC0712      ELSE
3014      GC0712          PERFORM 361-Release-Ref
3015      GC0712      END-IF
3016          END-EVALUATE
3017          .
3018
3019      351-COMPUTE SECTION.
3020          EVALUATE WS-Token-Prev-TXT
3021          WHEN 'COMPUTE'
3022              PERFORM 362-Release-Upd
3023          WHEN OTHER

```

Line Statement

Page: 71

```
=====
3024         PERFORM 361-Release-Ref
3025         END-EVALUATE
3026         .
3027
3028     351-DIVIDE SECTION.
3029         EVALUATE WS-Token-Prev-TXT
3030         WHEN 'INTO'
3031             PERFORM 363-Set-Upd
3032             MOVE F-Sort-Work-REC TO WS-Held-Reference-TXT
3033         WHEN 'GIVING'
3034             IF WS-Held-Reference-TXT NOT = SPACES
3035                 MOVE WS-Held-Reference-TXT To F-Sort-Work-REC
3036                 MOVE SPACES          To WS-Held-Reference-TXT
3037                 F-SW-Ref-Flag-CHR
3038             RELEASE F-Sort-Work-REC
3039         END-IF
3040         PERFORM 362-Release-Upd
3041         WHEN 'REMAINDER'
3042             PERFORM 362-Release-Upd
3043         WHEN OTHER
3044             PERFORM 361-Release-Ref
3045         END-EVALUATE
3046         .
3047
3048     351-FREE SECTION.
3049         PERFORM 362-Release-Upd
3050         .
3051
3052     351-INITIALIZE SECTION.
3053         EVALUATE WS-Token-Prev-TXT
3054         WHEN 'INITIALIZE'
3055             PERFORM 362-Release-Upd
3056         WHEN 'REPLACING'
3057             PERFORM 361-Release-Ref
3058         END-EVALUATE
3059         .
3060
3061     351-INSPECT SECTION.
3062         EVALUATE WS-Token-Prev-TXT
3063         WHEN 'INSPECT'
3064             PERFORM 364-Set-Ref
3065             MOVE SPACES TO WS-Held-Reference-TXT
3066             MOVE SPACES TO WS-Token-Prev-TXT
3067         WHEN 'TALLYING'
3068             PERFORM 362-Release-Upd
3069             MOVE SPACES TO WS-Token-Prev-TXT
3070         WHEN 'REPLACING'
3071             IF WS-Held-Reference-TXT NOT = SPACES
3072                 MOVE WS-Held-Reference-TXT TO F-Sort-Work-REC
3073                 MOVE SPACES          TO WS-Held-Reference-TXT
3074                 MOVE '*'          TO F-SW-Ref-Flag-CHR
3075             RELEASE F-Sort-Work-REC
3076         END-IF
3077         MOVE SPACES TO WS-Token-Prev-TXT
```

Line Statement

```

=====
3078      WHEN 'CONVERTING'
3079      IF WS-Held-Reference-TXT NOT = SPACES
3080          MOVE WS-Held-Reference-TXT TO F-Sort-Work-REC
3081          MOVE SPACES                TO WS-Held-Reference-TXT
3082          MOVE '*'                   TO F-SW-Ref-Flag-CHR
3083          RELEASE F-Sort-Work-REC
3084      END-IF
3085      MOVE SPACES TO WS-Token-Prev-TXT
3086      WHEN OTHER
3087          PERFORM 361-Release-Ref
3088      END-EVALUATE
3089      .
3090
3091 351-MOVE SECTION.
3092     EVALUATE WS-Token-Prev-TXT
3093     WHEN 'TO'
3094         PERFORM 362-Release-Upd
3095     WHEN OTHER
3096         PERFORM 361-Release-Ref
3097     END-EVALUATE
3098     .
3099
3100 351-MULTIPLY SECTION.
3101     EVALUATE WS-Token-Prev-TXT
3102     WHEN 'BY'
3103         PERFORM 363-Set-Upd
3104         MOVE F-Sort-Work-REC TO WS-Held-Reference-TXT
3105     WHEN 'GIVING'
3106         MOVE WS-Held-Reference-TXT TO F-Sort-Work-REC
3107         MOVE SPACES                TO WS-Held-Reference-TXT
3108                                     F-SW-Ref-Flag-CHR
3109         RELEASE F-Sort-Work-REC
3110         PERFORM 362-Release-Upd
3111     WHEN OTHER
3112         PERFORM 361-Release-Ref
3113     END-EVALUATE
3114     .
3115
3116 351-PERFORM SECTION.
3117     EVALUATE WS-Token-Prev-TXT
3118     WHEN 'VARYING'
3119         PERFORM 362-Release-Upd
3120         MOVE SPACES TO WS-Token-Prev-TXT
3121     WHEN 'AFTER'
3122         PERFORM 362-Release-Upd
3123         MOVE SPACES TO WS-Token-Prev-TXT
3124     WHEN OTHER
3125         PERFORM 361-Release-Ref
3126     END-EVALUATE
3127     .
3128
3129 351-SET SECTION.
3130     EVALUATE WS-Token-Prev-TXT
3131     WHEN 'SET'

```

Line Statement

Page: 73

```
=====
3132          PERFORM 362-Release-Upd
3133          WHEN OTHER
3134          PERFORM 361-Release-Ref
3135          END-EVALUATE
3136          .
3137
3138          351-STRING SECTION.
3139          EVALUATE WS-Token-Prev-TXT
3140          WHEN 'INTO'
3141          PERFORM 362-Release-Upd
3142          WHEN 'POINTER'
3143          PERFORM 362-Release-Upd
3144          WHEN OTHER
3145          PERFORM 361-Release-Ref
3146          END-EVALUATE
3147          .
3148
3149          351-SUBTRACT SECTION.
3150          EVALUATE WS-Token-Prev-TXT
3151          WHEN 'GIVING'
3152          PERFORM 362-Release-Upd
3153          WHEN 'FROM'
3154          PERFORM 362-Release-Upd
3155          WHEN OTHER
3156          PERFORM 361-Release-Ref
3157          END-EVALUATE
3158          .
3159
3160          351-TRANSFORM SECTION.
3161          EVALUATE WS-Token-Prev-TXT
3162          WHEN 'TRANSFORM'
3163          PERFORM 362-Release-Upd
3164          MOVE SPACES TO WS-Token-Prev-TXT
3165          WHEN OTHER
3166          PERFORM 361-Release-Ref
3167          END-EVALUATE
3168          .
3169
3170          351-UNSTRING SECTION.
3171          EVALUATE WS-Token-Prev-TXT
3172          WHEN 'INTO'
3173          PERFORM 362-Release-Upd
3174          WHEN 'DELIMITER'
3175          PERFORM 362-Release-Upd
3176          WHEN 'COUNT'
3177          PERFORM 362-Release-Upd
3178          WHEN 'POINTER'
3179          PERFORM 362-Release-Upd
3180          WHEN 'TALLYING'
3181          PERFORM 362-Release-Upd
3182          WHEN OTHER
3183          PERFORM 361-Release-Ref
3184          END-EVALUATE
3185          .
=====
```

Line Statement

Page: 74

```
=====
3186
3187      360-Release-Def SECTION.
3188          MOVE SPACES TO F-Sort-Work-REC
3189          MOVE WS-Curr-Prog-ID-TXT TO F-SW-Prog-ID-TXT
3190          MOVE WS-Token-Curr-Uc-TXT TO F-SW-Token-Uc-TXT
3191          MOVE WS-Token-Curr-TXT TO F-SW-Token-TXT
3192          MOVE WS-Curr-Section-TXT TO F-SW-Section-TXT
3193          MOVE WS-Curr-Line-NUM TO F-SW-Def-Line-NUM
3194          MOVE 0 TO F-SW-Ref-Line-NUM
3195          RELEASE F-Sort-Work-REC
3196          .
3197
3198      361-Release-Ref SECTION.
3199          PERFORM 364-Set-Ref
3200          RELEASE F-Sort-Work-REC
3201          .
3202
3203      362-Release-Upd SECTION.
3204          PERFORM 363-Set-Upd
3205          RELEASE F-Sort-Work-REC
3206          .
3207
3208      363-Set-Upd SECTION.
3209          MOVE SPACES TO F-Sort-Work-REC
3210          MOVE WS-Curr-Prog-ID-TXT TO F-SW-Prog-ID-TXT
3211          MOVE WS-Token-Curr-Uc-TXT TO F-SW-Token-Uc-TXT
3212          MOVE WS-Token-Curr-TXT TO F-SW-Token-TXT
3213          MOVE WS-Curr-Section-TXT TO F-SW-Section-TXT
3214          MOVE WS-Curr-Line-NUM TO F-SW-Ref-Line-NUM
3215          MOVE '*' TO F-SW-Ref-Flag-CHR
3216          .
3217
3218      364-Set-Ref SECTION.
3219          MOVE SPACES TO F-Sort-Work-REC
3220          MOVE WS-Curr-Prog-ID-TXT TO F-SW-Prog-ID-TXT
3221          MOVE WS-Token-Curr-Uc-TXT TO F-SW-Token-Uc-TXT
3222          MOVE WS-Token-Curr-TXT TO F-SW-Token-TXT
3223          MOVE WS-Curr-Section-TXT TO F-SW-Section-TXT
3224          MOVE WS-Curr-Line-NUM TO F-SW-Ref-Line-NUM
3225          .
=====
```


Line Statement

Page: 75

```

=====
3226      /
3227      400-Produce-Xref-Listing SECTION.
3228          MOVE SPACES          TO WS-Xref-Detail-Line-TXT
3229          WS-Group-Indicators-TXT
3230      MOVE 0                    TO WS-I-SUB
3231          WS-Lines-Left-NUM
3232 GC0710  MOVE 'N'              TO WS-RS-Duplicate-CHR
3233          PERFORM FOREVER
3234              RETURN F-Sort-Work-FILE AT END
3235              EXIT PERFORM
3236          END-RETURN
3237          IF F-SW-Prog-ID-TXT NOT = WS-GI-Prog-ID-TXT
3238          OR F-SW-Token-Uc-TXT NOT = WS-GI-Token-TXT
3239 GC0710  MOVE 'N' TO WS-RS-Duplicate-CHR
3240          IF WS-Xref-Detail-Line-TXT NOT = SPACES
3241              PERFORM 410-Generate-Report-Line
3242          END-IF
3243          IF F-SW-Prog-ID-TXT NOT = WS-GI-Prog-ID-TXT
3244              MOVE 0 TO WS-Lines-Left-NUM
3245          END-IF
3246          MOVE F-SW-Prog-ID-TXT TO WS-GI-Prog-ID-TXT
3247          MOVE F-SW-Token-Uc-TXT TO WS-GI-Token-TXT
3248          END-IF
3249 GC0710  IF F-SW-Token-Uc-TXT = WS-GI-Token-TXT
3250 GC0710  AND F-SW-Def-Line-NUM NOT = SPACES
3251 GC0710  AND WS-Xref-Detail-Line-TXT NOT = SPACES
3252 GC0710  MOVE 'Y' TO WS-RS-Duplicate-CHR
3253 GC0710  PERFORM 410-Generate-Report-Line
3254 GC0710  MOVE 0 TO WS-I-SUB
3255 GC0710  MOVE F-SW-Prog-ID-TXT TO WS-XDL-Prog-ID-TXT
3256 GC0710  MOVE ' (Duplicate Definition)' TO WS-XDL-Token-TXT
3257 GC0710  MOVE F-SW-Section-TXT TO WS-XDL-Section-TXT
3258 GC0710  MOVE F-SW-Def-Line-NUM TO WS-XDL-Def-Line-NUM
3259 GC0710  EXIT PERFORM CYCLE
3260 GC0710  END-IF
3261 GC0710  IF F-SW-Token-Uc-TXT = WS-GI-Token-TXT
3262 GC0710  AND F-SW-Def-Line-NUM = SPACES
3263 GC0710  AND WS-RS-Duplicate-CHR = 'Y'
3264 GC0710  MOVE 'N' TO WS-RS-Duplicate-CHR
3265 GC0710  PERFORM 410-Generate-Report-Line
3266 GC0710  MOVE 0 TO WS-I-SUB
3267 GC0710  MOVE F-SW-Prog-ID-TXT TO WS-XDL-Prog-ID-TXT
3268 GC0710  MOVE ' (Duplicate References)' TO WS-XDL-Token-TXT
3269 GC0710  END-IF
3270      IF WS-Xref-Detail-Line-TXT = SPACES
3271          MOVE F-SW-Prog-ID-TXT TO WS-XDL-Prog-ID-TXT
3272          MOVE F-SW-Token-TXT TO WS-XDL-Token-TXT
3273          MOVE F-SW-Section-TXT TO WS-XDL-Section-TXT
3274          IF F-SW-Def-Line-NUM NOT = SPACES
3275              MOVE F-SW-Def-Line-NUM TO WS-XDL-Def-Line-NUM
3276          END-IF
3277      END-IF
3278      IF F-SW-Reference-TXT > '000000'
3279          ADD 1 TO WS-I-SUB

```

Line Statement

Page: 76

```
=====
3280             IF WS-I-SUB > WS-Lines-Per-Rec-CONST
3281                 PERFORM 410-Generate-Report-Line
3282                 MOVE 1 TO WS-I-SUB
3283             END-IF
3284             MOVE F-SW-Ref-Line-NUM
3285                 TO WS-XDL-Ref-Line-NUM (WS-I-SUB)
3286             MOVE F-SW-Ref-Flag-CHR
3287                 TO WS-XDL-Ref-Flag-CHR (WS-I-SUB)
3288             END-IF
3289             END-PERFORM
3290             IF WS-Xref-Detail-Line-TXT NOT = SPACES
3291                 PERFORM 410-Generate-Report-Line
3292             END-IF
3293             .
=====
```

Line Statement

Page: 77

```
=====
3294      /
3295      410-Generate-Report-Line SECTION.
3296      IF WS-Lines-Left-NUM < 1
3297 GC0712      MOVE SPACES TO F-Listing-REC
3298 GC0712      WRITE F-Listing-REC BEFORE PAGE
3299 GC0712      MOVE SPACES TO F-Listing-REC
3300 GC0712      WRITE F-Listing-REC BEFORE 1
3301 GC0712      WRITE F-Listing-REC FROM WS-Xref-Header-1-TXT BEFORE 1
3302 GC0712      ADD 1 TO WS-Page-NUM
3303 GC0712      MOVE 'Page:' TO WS-PN-Literal-TXT
3304 GC0712      MOVE WS-Page-NUM TO WS-PN-Page-NUM
3305 GC0712      CALL '$JUSTIFY' USING WS-PN-Page-NUM, 'Left'
3306 GC0712      CALL '$JUSTIFY' USING WS-Page-No-TXT, 'Right'
3307 GC0712      MOVE WS-Page-No-TXT TO WS-XH3-Page-No-TXT
3308 GC0712      WRITE F-Listing-REC FROM WS-Xref-Header-2-TXT BEFORE 1
3309 GC0712      WRITE F-Listing-REC FROM WS-Xref-Header-3-TXT BEFORE 1
3310 GC0712      WRITE F-Listing-REC FROM WS-Xref-Header-4-TXT BEFORE 1
3311      COMPUTE WS-Lines-Left-NUM = WS-Lines-Per-Page-NUM - 4
3312      END-IF
3313 GC0712      WRITE F-Listing-REC FROM WS-Xref-Detail-Line-TXT BEFORE 1
3314      MOVE SPACES TO WS-Xref-Detail-Line-TXT
3315      MOVE 0 TO WS-I-SUB
3316      SUBTRACT 1 FROM WS-Lines-Left-NUM
3317      .
=====
```

Line Statement

Page: 78

```
=====
3318      /
3319      500-Produce-Source-Listing SECTION.
3320      OPEN INPUT F-Original-Src-FILE
3321      F-Expanded-Src-FILE
3322      MOVE 0 TO WS-Src-Line-NUM
3323      PERFORM FOREVER
3324      READ F-Expanded-Src-FILE AT END
3325      EXIT PERFORM
3326      END-READ
3327 GC0712 IF F-ES-1-7-TXT NOT = '#DEFLIT'
3328 GC0712     IF F-ES-1-CHR = '#'
3329 GC0712         PERFORM 510-Control-Record
3330 GC0712     ELSE
3331 GC0712         PERFORM 520-Expanded-Src-Record
3332 GC0712     END-IF
3333 GC0712 END-IF
3334      END-PERFORM
3335      CLOSE F-Original-Src-FILE
3336      F-Expanded-Src-FILE
3337      .
```

Line Statement

Page: 79

```
=====
3338      /
3339      510-Control-Record SECTION.
3340          UNSTRING F-ES-2-256-TXT-256
3341              DELIMITED BY '''
3342              INTO WS-Temp-10-Chars-TXT
3343                  WS-Temp-256-Chars-TXT
3344                  WS-Dummy-TXT
3345 GC1010 IF UPPER-CASE(TRIM(WS-Temp-256-Chars-TXT,Trailing)) =
3346 GC1010 TRIM(WS-Main-Module-Name-TXT) *> Main Pgm
3347          SET WS-RS-In-Main-Module-BOOL TO TRUE
3348          IF WS-Src-Line-NUM > 0
3349              READ F-Expanded-Src-FILE END-READ
3350          END-IF
3351          ELSE *> COPY
3352              SET WS-RS-In-Copybook-BOOL TO TRUE
3353          END-IF
3354          .
```

Line Statement

Page: 80

```

=====
3355      /
3356      520-Expanded-Src-Record SECTION.
3357      IF WS-RS-In-Main-Module-BOOL
3358          ADD 1 To WS-Curr-Line-NUM
3359 GC0712    READ F-Original-Src-FILE AT END CONTINUE END-READ
3360          ADD 1 TO WS-Src-Line-NUM
3361          MOVE SPACES TO WS-Src-Detail-Line-TXT
3362          MOVE WS-Src-Line-NUM TO WS-SDL-Line-NUM
3363          MOVE F-OS-1-128-TXT TO WS-SDL-Statement-TXT
3364 GC0712    MOVE LOWER-CASE(TRIM(F-OS-8-72-TXT,LEADING))
3365 GC0712      TO WS-Temp-65-Chars-TXT
3366 GC0712    INSPECT WS-Temp-65-Chars-TXT REPLACING ALL '.' BY SPACE
3367 GC0712    EVALUATE TRUE
3368 GC0712      WHEN F-OS-7-CHR = '/'
3369 GC0712        MOVE 0 TO WS-Lines-Left-NUM
3370 GC0712      WHEN WS-Temp-65-Chars-TXT = "eject"
3371 GC0712        MOVE 0 TO WS-Lines-Left-NUM
3372 GC0712      EXIT SECTION
3373 GC0712      WHEN WS-Temp-65-Chars-TXT = "skip1"
3374 GC0712        MOVE SPACES TO WS-Src-Detail-Line-TXT
3375 GC0712        PERFORM 530-Generate-Source-Line
3376 GC0712      EXIT SECTION
3377 GC0712      WHEN WS-Temp-65-Chars-TXT = "skip2"
3378 GC0712        MOVE SPACES TO WS-Src-Detail-Line-TXT
3379 GC0712        PERFORM 530-Generate-Source-Line 2 TIMES
3380 GC0712      EXIT SECTION
3381 GC0712      WHEN WS-Temp-65-Chars-TXT = "skip3"
3382 GC0712        MOVE SPACES TO WS-Src-Detail-Line-TXT
3383 GC0712        PERFORM 530-Generate-Source-Line 3 TIMES
3384 GC0712      EXIT SECTION
3385 GC0712    END-EVALUATE
3386          PERFORM 530-Generate-Source-Line
3387          IF F-OS-129-256-TXT NOT = SPACES
3388              MOVE SPACES TO WS-Src-Detail-Line-TXT
3389              MOVE F-OS-129-256-TXT TO WS-SDL-Statement-TXT
3390              PERFORM 530-Generate-Source-Line
3391          END-IF
3392      ELSE
3393          IF F-Expanded-Src-REC NOT = SPACES
3394              MOVE SPACES TO WS-Src-Detail-Line-TXT
3395              MOVE F-Expanded-Src-REC(1:128)
3396                  TO WS-SDL-Statement-TXT
3397 GC0712      MOVE LOWER-CASE(TRIM(F-OS-8-72-TXT,LEADING))
3398 GC0712          TO WS-Temp-65-Chars-TXT
3399 GC0712      INSPECT WS-Temp-65-Chars-TXT
3400 GC0712          REPLACING ALL '.' BY SPACE
3401 GC0712      EVALUATE TRUE
3402 GC0712        WHEN WS-Temp-65-Chars-TXT = "eject"
3403 GC0712          MOVE 0 TO WS-Lines-Left-NUM
3404 GC0712          EXIT SECTION
3405 GC0712        WHEN WS-Temp-65-Chars-TXT = "skip1"
3406 GC0712          MOVE SPACES TO WS-Src-Detail-Line-TXT
3407 GC0712          PERFORM 530-Generate-Source-Line
3408 GC0712          EXIT SECTION

```

Line Statement

Page: 81

```
=====
3409 GC0712          WHEN WS-Temp-65-Chars-TXT = "skip2"
3410 GC0712          MOVE SPACES TO WS-Src-Detail-Line-TXT
3411 GC0712          PERFORM 530-Generate-Source-Line 2 TIMES
3412 GC0712          EXIT SECTION
3413 GC0712          WHEN WS-Temp-65-Chars-TXT = "skip3"
3414 GC0712          MOVE SPACES TO WS-Src-Detail-Line-TXT
3415 GC0712          PERFORM 530-Generate-Source-Line 3 TIMES
3416 GC0712          EXIT SECTION
3417 GC0712          END-EVALUATE
3418                PERFORM 530-Generate-Source-Line
3419                IF F-Expanded-Src-REC(129:128) NOT = SPACES
3420                    MOVE SPACES TO WS-Src-Detail-Line-TXT
3421                    MOVE F-Expanded-Src-REC(129:128)
3422                      TO WS-SDL-Statement-TXT
3423                    PERFORM 530-Generate-Source-Line
3424                END-IF
3425            END-IF
3426        END-IF
3427        .
```

Line Statement

Page: 82

```
=====
3428 /
3429 530-Generate-Source-Line SECTION.
3430 IF WS-Lines-Left-NUM < 1
3431 GC0712 WRITE F-Listing-REC FROM SPACES BEFORE PAGE
3432 GC0712 WRITE F-Listing-REC FROM SPACES BEFORE 1
3433 GC0712 WRITE F-Listing-REC FROM WS-Src-Header-1-TXT BEFORE 1
3434 GC0712 ADD 1 TO WS-Page-NUM
3435 GC0712 MOVE 'Page:' TO WS-PN-Literal-TXT
3436 GC0712 MOVE WS-Page-NUM TO WS-PN-Page-NUM
3437 GC0712 CALL 'C$JUSTIFY' USING WS-PN-Page-NUM, 'Left'
3438 GC0712 CALL 'C$JUSTIFY' USING WS-Page-No-TXT, 'Right'
3439 GC0712 MOVE WS-Page-No-TXT TO WS-SH3-Page-No-TXT
3440 WRITE F-Listing-REC FROM WS-Src-Header-2-TXT BEFORE 1
3441 GC0712 WRITE F-Listing-REC FROM WS-Src-Header-3-TXT BEFORE 1
3442 GC0712 WRITE F-Listing-REC FROM WS-Src-Header-4-TXT BEFORE 1
3443 COMPUTE WS-Lines-Left-NUM = WS-Lines-Per-Page-NUM - 4
3444 END-IF
3445 GC0712 WRITE F-Listing-REC FROM WS-Src-Detail-Line-TXT BEFORE 1
3446 MOVE SPACES TO WS-Src-Detail-Line-TXT
3447 SUBTRACT 1 FROM WS-Lines-Left-NUM
3448 .
3449
3450 END PROGRAM LISTING.
```


PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)								
CHECKSRC	000-Main	1294	PROCEDURE									
CHECKSRC	L-A1-CHR	1284	LINKAGE	1296	1308	1311	1318					
CHECKSRC	L-A2-IDENT-DIVISION-BOOL	1289	LINKAGE	1340*								
CHECKSRC	L-A2-LINKAGE-SECTION-BOOL	1288	LINKAGE	1334*								
CHECKSRC	L-A2-Nothing-Special-BOOL	1290	LINKAGE	1295*								
CHECKSRC	L-Argument-1-TXT	1283	LINKAGE	1292								
CHECKSRC	L-Argument-2-CHR	1287	LINKAGE	1293								
CHECKSRC	UPPER-CASE		PROCEDURE	1311								
CHECKSRC	WS-Compressed-Src-TXT	1270	WORKING-STORAGE	1305*	1328	1332	1338					
CHECKSRC	WS-CS-CHR	1271	WORKING-STORAGE	1312*	1318*	1327	1331	1337				
CHECKSRC	WS-I-SUB	1279	WORKING-STORAGE	1306*	1307	1308	1311	1318	1324*	1325	1327	
				1328	1331	1331	1332	1337	1337	1338		
CHECKSRC	WS-J-SUB	1281	WORKING-STORAGE	1304*	1310*	1312	1317*	1318				
CHECKSRC	WS-RS-Found-SPACE-CHR	1275	WORKING-STORAGE									
CHECKSRC	WS-RS-Not-Skipping-SPACE-BOOL	1277	WORKING-STORAGE	1303*	1309	1316*						
CHECKSRC	WS-RS-Skipping-SPACE-BOOL	1276	WORKING-STORAGE	1313*								
CHECKSRC	WS-Runtime-Switches-TXT	1274	WORKING-STORAGE									

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)								
GCic	000-File-Error	642	PROCEDURE									
GCic	000-Main	663	PROCEDURE									
GCic	100-Initialization	688	PROCEDURE	664								
GCic	200-Let-User-Set-Switches	800	PROCEDURE	667								
GCic	210-Run-Compiler	881	PROCEDURE	668								
GCic	220-Make-Listing	1017	PROCEDURE	672								
GCic	230-Run-Program	1045	PROCEDURE	676								
GCic	240-Find-LINKAGE-SECTION	1152	PROCEDURE	734								
GCic	250-Autoload-Listing	1191	PROCEDURE	678	1008	1039						
GCic	900-Terminate	1237	PROCEDURE	680	728	861	1009	1145				
GCic	COB-COLOR-BLACK	170	[screenio]	427	432	522	532	542	547	557	564	
				579	594	610	617	624				
GCic	COB-COLOR-BLUE	170	[screenio]	510								
GCic	COB-COLOR-CYAN	170	[screenio]	523								
GCic	COB-COLOR-GREEN	170	[screenio]	433	548	595	611					
GCic	COB-COLOR-MAGENTA	170	[screenio]									
GCic	COB-COLOR-RED	170	[screenio]	516	580							
GCic	COB-COLOR-WHITE	170	[screenio]	428	511	517	533	543	558	565	618	
				625								
GCic	COB-COLOR-YELLOW	170	[screenio]									
GCic	COB-CRT-STATUS		PROCEDURE	804	805							
GCic	COB-SCR-ESC	170	[screenio]	860								
GCic	COB-SCR-F1	170	[screenio]	806								
GCic	COB-SCR-F10	170	[screenio]									
GCic	COB-SCR-F11	170	[screenio]									
GCic	COB-SCR-F12	170	[screenio]	863								
GCic	COB-SCR-F13	170	[screenio]									
GCic	COB-SCR-F14	170	[screenio]									
GCic	COB-SCR-F15	170	[screenio]									
GCic	COB-SCR-F16	170	[screenio]									
GCic	COB-SCR-F17	170	[screenio]									
GCic	COB-SCR-F18	170	[screenio]									
GCic	COB-SCR-F19	170	[screenio]									
GCic	COB-SCR-F2	170	[screenio]	812								
GCic	COB-SCR-F20	170	[screenio]									
GCic	COB-SCR-F21	170	[screenio]									
GCic	COB-SCR-F22	170	[screenio]									
GCic	COB-SCR-F23	170	[screenio]									
GCic	COB-SCR-F24	170	[screenio]									
GCic	COB-SCR-F25	170	[screenio]									
GCic	COB-SCR-F26	170	[screenio]									
GCic	COB-SCR-F27	170	[screenio]									
GCic	COB-SCR-F28	170	[screenio]									
GCic	COB-SCR-F29	170	[screenio]									
GCic	COB-SCR-F3	170	[screenio]	818								
GCic	COB-SCR-F30	170	[screenio]									
GCic	COB-SCR-F31	170	[screenio]									
GCic	COB-SCR-F32	170	[screenio]									
GCic	COB-SCR-F33	170	[screenio]									
GCic	COB-SCR-F34	170	[screenio]									
GCic	COB-SCR-F35	170	[screenio]									
GCic	COB-SCR-F36	170	[screenio]									
GCic	COB-SCR-F37	170	[screenio]									
GCic	COB-SCR-F38	170	[screenio]									

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)
GCic	COB-SCR-F39	170	[screenio	
GCic	COB-SCR-F4	170	[screenio	824
GCic	COB-SCR-F40	170	[screenio	
GCic	COB-SCR-F41	170	[screenio	
GCic	COB-SCR-F42	170	[screenio	
GCic	COB-SCR-F43	170	[screenio	
GCic	COB-SCR-F44	170	[screenio	
GCic	COB-SCR-F45	170	[screenio	
GCic	COB-SCR-F46	170	[screenio	
GCic	COB-SCR-F47	170	[screenio	
GCic	COB-SCR-F48	170	[screenio	
GCic	COB-SCR-F49	170	[screenio	
GCic	COB-SCR-F5	170	[screenio	830
GCic	COB-SCR-F50	170	[screenio	
GCic	COB-SCR-F51	170	[screenio	
GCic	COB-SCR-F52	170	[screenio	
GCic	COB-SCR-F53	170	[screenio	
GCic	COB-SCR-F54	170	[screenio	
GCic	COB-SCR-F55	170	[screenio	
GCic	COB-SCR-F56	170	[screenio	
GCic	COB-SCR-F57	170	[screenio	
GCic	COB-SCR-F58	170	[screenio	
GCic	COB-SCR-F59	170	[screenio	
GCic	COB-SCR-F6	170	[screenio	836
GCic	COB-SCR-F60	170	[screenio	
GCic	COB-SCR-F61	170	[screenio	
GCic	COB-SCR-F62	170	[screenio	
GCic	COB-SCR-F63	170	[screenio	
GCic	COB-SCR-F64	170	[screenio	
GCic	COB-SCR-F7	170	[screenio	842
GCic	COB-SCR-F8	170	[screenio	848
GCic	COB-SCR-F9	170	[screenio	854
GCic	COB-SCR-FATAL	170	[screenio	
GCic	COB-SCR-KEY-DOWN	170	[screenio	
GCic	COB-SCR-KEY-UP	170	[screenio	
GCic	COB-SCR-MAX-FIELD	170	[screenio	
GCic	COB-SCR-NO-FIELD	170	[screenio	
GCic	COB-SCR-OK	170	[screenio	
GCic	COB-SCR-PAGE_DOWN	170	[screenio	
GCic	COB-SCR-PAGE_UP	170	[screenio	
GCic	COB-SCR-PRINT	170	[screenio	
GCic	COB-SCR-TIME-OUT	170	[screenio	
GCic	CONCATENATE		PROCEDURE	789 968 990 996 1215 1218 1222* 1224*
				1226* 1228* 1230*
GCic	F-Cobc-Output-FILE	163	FILE	154 985 1003
GCic	F-Cobc-Output-REC	164	FILE	986 991* 992 1000* 1001
GCic	F-Source-Code-FILE	166	FILE	157 643 1153 1157 1158 1171 1172 1179
GCic	F-Source-Code-REC	167	FILE	1162 1176
GCic	F1		WORKING-STORAGE	198
GCic	F12		WORKING-STORAGE	192
GCic	F2		WORKING-STORAGE	205
GCic	F3		WORKING-STORAGE	201
GCic	F4		WORKING-STORAGE	199
GCic	F5		WORKING-STORAGE	202

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)								
GCic	F6		WORKING-STORAGE	203								
GCic	F7		WORKING-STORAGE	206								
GCic	F8		WORKING-STORAGE	200								
GCic	F9		WORKING-STORAGE	204								
GCic	LD-Horiz-Line	412	SCREEN	788								
GCic	LD-LL-Corner	407	SCREEN	444	474	490	504					
GCic	LD-Lower-T	411	SCREEN	476								
GCic	LD-LR-Corner	409	SCREEN	446	478	492	506					
GCic	LD-UL-Corner	406	SCREEN	434	448	480	494					
GCic	LD-Upper-T	410	SCREEN	450								
GCic	LD-UR-Corner	408	SCREEN	436	452	482	496					
GCic	LD-Vert-Line	413	SCREEN	438	439	441	442	454	455	456	458	
				459	460	462	463	464	466	467	468	
				470	471	472	484	485	487	488	498	
				499	501	502						
GCic	LENGTH		PROCEDURE	742	770	1070						
GCic	LOWER-CASE		PROCEDURE	887	1065							
GCic	RETURN-CODE		PROCEDURE	987	1020*	1031*	1033					
GCic	S-Blank-SCR	425	SCREEN	671	1142	1213	1242					
GCic	S-Switches-SCR	427	SCREEN	803*	885	975	1004	1019	1037	1239		
GCic	SELCHAR		PROCEDURE	808	814	820	826	832	838	844	850	
				856	1181							
GCic	TRIM		PROCEDURE	651	656	761*	790	894	957	962	968	
				978	984*	997	1070	1082	1100	1125	1143*	
				1197	1199*	1204	1206*	1215	1218	1222*	1224*	
				1226*	1228*	1230*						
GCic	WHEN-COMPILED		PROCEDURE	697								
GCic	WS-Cmd-Args-TXT	221	WORKING-STORAGE	710*	712	713	716	721				
GCic	WS-Cmd-End-Quote-CHR	223	WORKING-STORAGE	714*	717							
GCic	WS-Cmd-SUB	225	WORKING-STORAGE	711*	712	713	715*	719*	724*			
GCic	WS-Cmd-TXT	219	WORKING-STORAGE	882*	977*	982*	984*	1046*	1053*	1063*	1066*	
				1073*	1077*	1083*	1088*	1093*	1101*	1109*	1115*	
				1121*	1126*	1131*	1135*	1143*	1194*	1198*	1199*	
				1201*	1205*	1206*	1216*	1219*	1222*	1224*	1226*	
				1228*	1230*							
GCic	WS-Cobc-Cmd-TXT	227	WORKING-STORAGE	883*	892*	896*	900*	904*	909*	914*	919*	
				924*	929*	934*	938*	951*	959*	963*	978	
GCic	WS-Compilation-Switches-TXT	172	WORKING-STORAGE									
GCic	WS-Config-Fn-TXT	229	WORKING-STORAGE	888*	894							
GCic	WS-CS-All-Switches-TXT	207	WORKING-STORAGE									
GCic	WS-CS-Arg-H1-TXT	174	WORKING-STORAGE	628*								
GCic	WS-CS-Arg-H2-TXT	175	WORKING-STORAGE	629*								
GCic	WS-CS-Args-TXT	173	WORKING-STORAGE	1124	1125							
GCic	WS-CS-Config-NUM	192	WORKING-STORAGE	620	864*	865	866*	887				
GCic	WS-CS-DEBUG-CHR	209	WORKING-STORAGE	581	807	808*	810*	907				
GCic	WS-CS-EXECUTE-CHR	210	WORKING-STORAGE	584	674	825	826*	828*				
GCic	WS-CS-Extra-H1-TXT	195	WORKING-STORAGE	626*								
GCic	WS-CS-Extra-H2-TXT	196	WORKING-STORAGE	627*								
GCic	WS-CS-Extra-TXT	194	WORKING-STORAGE	955	957							
GCic	WS-CS-Filename-TXT	185	WORKING-STORAGE	620	887							
GCic	WS-CS-Filenames-Table-TXT	184	WORKING-STORAGE									
GCic	WS-CS-Filenames-TXT	176	WORKING-STORAGE	184								
GCic	WS-CS-FREE-CHR	211	WORKING-STORAGE	589	849	850*	852*	932				
GCic	WS-CS-LIBRARY-CHR	212	WORKING-STORAGE	583	819	820*	822*	898	1051	1106	1154*	

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)							
				1181*							
GCic	WS-CS-LISTING-CHR	213	WORKING-STORAGE	585	670	831	832*	834*	945	947*	949
GCic	WS-CS-NOFUNC-CHR	214	WORKING-STORAGE	587	837	838*	840*	922			
GCic	WS-CS-NOTRUNC-CHR	215	WORKING-STORAGE	590	855	856*	858*	912			
GCic	WS-CS-Switch-Defaults-TXT	197	WORKING-STORAGE	208							
GCic	WS-CS-TRACEALL-CHR	216	WORKING-STORAGE	582	813	814*	816*	917			
GCic	WS-CS-WARNALL-CHR	217	WORKING-STORAGE	588	843	844*	846*	927			
GCic	WS-Delete-Fn-TXT	231	WORKING-STORAGE								
GCic	WS-File-Name-TXT	233	WORKING-STORAGE	157	651	656	718*	723*	726	739	742
				749	752	763	1026*				
GCic	WS-File-Status-Message-TXT	237	WORKING-STORAGE								
GCic	WS-FN-CHR	234	WORKING-STORAGE	744	751*	755*					
GCic	WS-FSM-Msg-TXT	241	WORKING-STORAGE	646*	646*	646*	646*	646*	646*	646*	646*
				646*	646*	646*	646*	646*	646*	646*	646*
				646*	646*	646*	646*	646*	646*	646*	646*
				646*	646*	646*	646*	646*	646*	646*	646*
GCic	WS-FSM-Status-CD	239	WORKING-STORAGE	159	646	648					
GCic	WS-Horizontal-Line-TXT	243	WORKING-STORAGE	435	445	449	451	475	477	481	491
				495	505	788*					
GCic	WS-I-SUB	245	WORKING-STORAGE	742*	743	744	745	747	751	755	770*
				771	772	773	775	779	783	886*	897*
				901*	905*	910*	915*	920*	925*	930*	935*
				939*	952*	960*	964*	1047*	1054*	1064*	1067*
				1074*	1078*	1084*	1089*	1094*	1102*	1110*	1116*
				1122*	1127*	1132*	1136*				
GCic	WS-J-SUB	247	WORKING-STORAGE	1069*	1070	1071	1076				
GCic	WS-Listing-Filename-TXT	249	WORKING-STORAGE	154	969*	970*	979	997	1025*	1197	1204
GCic	WS-OC-Compile-DT	251	WORKING-STORAGE	513	697*						
GCic	WS-OS-Cygwin-BOOL	285	WORKING-STORAGE	739	763	1061	1091	1193			
GCic	WS-OS-Dir-CHR	260	WORKING-STORAGE	740*	744	755	764*	1087			
GCic	WS-OS-Exe-Ext-CONST	261	WORKING-STORAGE	1107	1108						
GCic	WS-OS-Lib-Ext-CONST	262	WORKING-STORAGE	1113	1114						
GCic	WS-OS-Lib-Type-CONST	263	WORKING-STORAGE	599							
GCic	WS-OS-OSX-BOOL	287	WORKING-STORAGE	1200							
GCic	WS-OS-Type-CD	264	WORKING-STORAGE	790	1027*						
GCic	WS-OS-Type-FILLER-TXT	289	WORKING-STORAGE	295							
GCic	WS-OS-Type-TXT	296	WORKING-STORAGE	790							
GCic	WS-OS-Types-TXT	295	WORKING-STORAGE								
GCic	WS-OS-UNIX-BOOL	286	WORKING-STORAGE	1091							
GCic	WS-OS-Windows-BOOL	284	WORKING-STORAGE	1129	1193	1214					
GCic	WS-Output-Msg-TXT	299	WORKING-STORAGE	518	647*	793*	870*	884*	974*	989*	990
				998*	1006*	1018*	1024*	1030*	1035*	1144*	1238
GCic	WS-Path-Delimiter-CHR	301	WORKING-STORAGE								
GCic	WS-PFN-CHR	308	WORKING-STORAGE	772	779*	783*					
GCic	WS-Pgm-Nm-TXT	311	WORKING-STORAGE	776*	781*	968	1100	1215	1218		
GCic	WS-Prog-Extension-TXT	303	WORKING-STORAGE	777*	782*						
GCic	WS-Prog-File-Name-TXT	307	WORKING-STORAGE	559	749*	754*	770	776	780	962	
GCic	WS-Prog-Folder-TXT	305	WORKING-STORAGE	560	748*	753*	757	758*	761*	1060	1061
				1065	1070	1071	1076	1082			
GCic	WS-RS-1st-Prog-Complete-BOOL	324	WORKING-STORAGE	1156	1165*						
GCic	WS-RS-Compile-Failed-BOOL	317	WORKING-STORAGE	995*	1007						
GCic	WS-RS-Compile-OK-BOOL	315	WORKING-STORAGE	669	988*						
GCic	WS-RS-Compile-OK-CHR	314	WORKING-STORAGE								
GCic	WS-RS-Compile-OK-Warn-BOOL	316	WORKING-STORAGE	669							

GNU COBOL V2.0 11FEB2012 Cross-Reference Listing - GCic for Windows/MinGW Copyright (C) 2009 - 2013, Gary L. Cutler, GPL 2013/11/21

E:/GNU-COBOL/samples/GCic.cbl

Page: 88

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)		
GCic	WS-RS-Complete-BOOL	319	WORKING-STORAGE	666	993*	
GCic	WS-RS-Complete-CHR	318	WORKING-STORAGE			
GCic	WS-RS-Double-Quote-Used-BOOL	322	WORKING-STORAGE	1059*	1085*	1119
GCic	WS-RS-IDENT-DIV-CHR	323	WORKING-STORAGE			
GCic	WS-RS-More-To-1st-Prog-BOOL	325	WORKING-STORAGE	1155*		
GCic	WS-RS-No-Switch-Changes-BOOL	327	WORKING-STORAGE	802	873*	
GCic	WS-RS-No-Switch-Chgs-CHR	326	WORKING-STORAGE			
GCic	WS-RS-Not-Complete-BOOL	320	WORKING-STORAGE	665*		
GCic	WS-RS-Output-File-Avail-BOOL	331	WORKING-STORAGE	675	976*	
GCic	WS-RS-Output-File-Busy-BOOL	330	WORKING-STORAGE			
GCic	WS-RS-Output-File-Busy-CHR	329	WORKING-STORAGE			
GCic	WS-RS-Quote-CHR	321	WORKING-STORAGE			
GCic	WS-RS-Source-Rec-Ident-BOOL	334	WORKING-STORAGE	1164	1170	
GCic	WS-RS-Source-Rec-Ignored-BOOL	335	WORKING-STORAGE	1168*		
GCic	WS-RS-Source-Rec-Linkage-BOOL	333	WORKING-STORAGE	1169	1180	
GCic	WS-RS-Source-Record-Type-CHR	332	WORKING-STORAGE	1163*	1177*	
GCic	WS-RS-Switch-Changes-BOOL	328	WORKING-STORAGE	801*		
GCic	WS-RS-Switch-Error-CHR	336	WORKING-STORAGE			
GCic	WS-RS-Switch-Is-Bad-BOOL	337	WORKING-STORAGE			
GCic	WS-RS-Switch-Is-Good-BOOL	338	WORKING-STORAGE			
GCic	WS-Runtime-Switches-TXT	313	WORKING-STORAGE			
GCic	WS-Tally-QTY	340	WORKING-STORAGE	942*	944*	946

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)							
LISTING	000-Main	2393	PROCEDURE								
LISTING	100-Initialization	2410	PROCEDURE	2394							
LISTING	300-Tokenize-Source	2479	PROCEDURE	2401							
LISTING	310-Get-Token	2530	PROCEDURE	2488							
LISTING	311-Control-Record	2690	PROCEDURE	2549							
LISTING	312-Expanded-Src-Record	2736	PROCEDURE	2551							
LISTING	313-Check-For-Numeric-Token	2743	PROCEDURE	2677							
LISTING	320-IDENTIFICATION-DIVISION	2785	PROCEDURE	2509							
LISTING	330-ENVIRONMENT-DIVISION	2809	PROCEDURE	2511							
LISTING	340-DATA-DIVISION	2832	PROCEDURE	2513							
LISTING	350-PROCEDURE-DIVISION	2887	PROCEDURE	2515							
LISTING	351-ACCEPT	2971	PROCEDURE	2933							
LISTING	351-ADD	2981	PROCEDURE	2935							
LISTING	351-ALLOCATE	2992	PROCEDURE	2937							
LISTING	351-CALL	3004	PROCEDURE	2939							
LISTING	351-COMPUTE	3019	PROCEDURE	2941							
LISTING	351-DIVIDE	3028	PROCEDURE	2943							
LISTING	351-FREE	3048	PROCEDURE	2945							
LISTING	351-INITIALIZE	3052	PROCEDURE	2947							
LISTING	351-INSPECT	3061	PROCEDURE	2949							
LISTING	351-MOVE	3091	PROCEDURE	2951							
LISTING	351-MULTIPLY	3100	PROCEDURE	2953							
LISTING	351-PERFORM	3116	PROCEDURE	2955							
LISTING	351-SET	3129	PROCEDURE	2957							
LISTING	351-STRING	3138	PROCEDURE	2959							
LISTING	351-SUBTRACT	3149	PROCEDURE	2961							
LISTING	351-TRANSFORM	3160	PROCEDURE	2963							
LISTING	351-UNSTRING	3170	PROCEDURE	2965							
LISTING	360-Release-Def	3187	PROCEDURE	2825	2863	2866	2869	2872	2909		
LISTING	361-Release-Ref	3198	PROCEDURE	2827	2881	2967	2977	2988	3000	3014	3024
				3044	3057	3087	3096	3112	3125	3134	3145
				3156	3166	3183					
LISTING	362-Release-Upd	3203	PROCEDURE	2875	2878	2974	2984	2986	2995	2998	3007
				3009	3012	3022	3040	3042	3049	3055	3068
				3094	3110	3119	3122	3132	3141	3143	3152
				3154	3163	3173	3175	3177	3179	3181	
LISTING	363-Set-Upd	3208	PROCEDURE	3031	3103	3204					
LISTING	364-Set-Ref	3218	PROCEDURE	3064	3199						
LISTING	400-Produce-Xref-Listing	3227	PROCEDURE	2402							
LISTING	410-Generate-Report-Line	3295	PROCEDURE	3241	3253	3265	3281	3291			
LISTING	500-Produce-Source-Listing	3319	PROCEDURE	2396							
LISTING	510-Control-Record	3339	PROCEDURE	3329							
LISTING	520-Expanded-Src-Record	3356	PROCEDURE	3331							
LISTING	530-Generate-Source-Line	3429	PROCEDURE	3375	3379	3383	3386	3390	3407	3411	3415
				3418	3423						
LISTING	F-ES-1-7-TXT	1397	FILE	2547	3327						
LISTING	F-ES-1-CHR	1394	FILE	2548	3328						
LISTING	F-ES-2-256-TXT-256	1395	FILE	2691	3340						
LISTING	F-ES-8-256-TXT	1398	FILE								
LISTING	F-Expanded-Src-FILE	1392	FILE	1383	2480	2527	2537	3321	3324	3336	3349
LISTING	F-Expanded-Src-REC	1393	FILE	2481*	2535	2559	2561	2591	2595	2604	2608
				2617	2621	2629	2647	2732*	3393	3395	3419
				3421							
LISTING	F-Expanded-Src2-REC	1396	FILE								

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)								
LISTING	F-Listing-FILE	1400	FILE	1385	2395	2403						
LISTING	F-Listing-REC	1401	FILE	3297*	3298	3299*	3300	3301	3308	3309	3310	3310
				3313	3431	3432	3433	3440	3441	3442	3445	
LISTING	F-Original-Src-FILE	1403	FILE	1387	3320	3335	3359					
LISTING	F-Original-Src-REC	1404	FILE									
LISTING	F-OS-1-128-TXT	1405	FILE	3363								
LISTING	F-OS-129-256-TXT	1410	FILE	3387	3389							
LISTING	F-OS-7-CHR	1407	FILE	3368								
LISTING	F-OS-8-72-TXT	1408	FILE	3364	3397							
LISTING	F-Sort-Work-FILE	1412	FILE	1389	2397	3234						
LISTING	F-Sort-Work-REC	1413	FILE	2502*	2504	2539*	2541	3032	3035*	3038	3072*	
				3075	3080*	3083	3104	3106*	3109	3188*	3195	
				3200	3205	3209*	3219*					
LISTING	F-SW-Def-Line-NUM	1418	FILE	3193*	3250	3258	3262	3274	3275			
LISTING	F-SW-Prog-ID-TXT	1414	FILE	2398	3189*	3210*	3220*	3237	3243	3246	3255	
				3267	3271							
LISTING	F-SW-Ref-Flag-CHR	1421	FILE	3037*	3074*	3082*	3108*	3215*	3286			
LISTING	F-SW-Ref-Line-NUM	1420	FILE	2400	3194*	3214*	3224*	3284				
LISTING	F-SW-Reference-TXT	1419	FILE	3278								
LISTING	F-SW-Section-TXT	1417	FILE	3192*	3213*	3223*	3257	3273				
LISTING	F-SW-Token-TXT	1416	FILE	3191*	3212*	3222*	3272					
LISTING	F-SW-Token-Uc-TXT	1415	FILE	2399	3190*	3211*	3221*	3238	3247	3249	3261	
LISTING	L-Listing-Fn-TXT	2384	LINKAGE	1385	2390							
LISTING	L-OS-Type-CD	2388	LINKAGE	2392	2414	2420						
LISTING	L-Src-Fn-TXT	2386	LINKAGE	1387	2391	2424	2424*	2425	2426	2430	2433	
				2439	2443	2445	2450					
LISTING	LENGTH		PROCEDURE	2425	2443	2713						
LISTING	LOWER-CASE		PROCEDURE	3364	3397							
LISTING	NUMVAL		PROCEDURE	2465	2700							
LISTING	RETURN-CODE		PROCEDURE	2461								
LISTING	TRIM		PROCEDURE	2424	2443	2455	2701	2702	2713	3345	3346	
				3364	3397							
LISTING	UPPER-CASE		PROCEDURE	2430	2433	2492	2638	2661	2701	3345		
LISTING	WS-Argument-Is-Updatable-BOOL	2305	WORKING-STORAGE	2914*	2919*	2921*	2926*	3011				
LISTING	WS-Argument-Type-CD	2304	WORKING-STORAGE									
LISTING	WS-CD-In-DATA-DIV-BOOL	1438	WORKING-STORAGE	2512								
LISTING	WS-CD-In-ENV-DIV-BOOL	1437	WORKING-STORAGE	2510								
LISTING	WS-CD-In-IDENT-DIV-BOOL	1436	WORKING-STORAGE	2508								
LISTING	WS-CD-In-PROC-DIV-BOOL	1439	WORKING-STORAGE	2514								
LISTING	WS-CPI-13-15-TXT	1445	WORKING-STORAGE	2799*								
LISTING	WS-CPI-16-CHR	1446	WORKING-STORAGE	2798								
LISTING	WS-CS-1-CHR	1449	WORKING-STORAGE	2725*								
LISTING	WS-CS-11-14-TXT	1452	WORKING-STORAGE	2727	2728*							
LISTING	WS-CS-15-CHR	1453	WORKING-STORAGE	2730*								
LISTING	WS-CS-2-14-TXT	1450	WORKING-STORAGE	2726*								
LISTING	WS-Curr-Char-Is-Punct-BOOL	1427	WORKING-STORAGE	2570	2578	2583						
LISTING	WS-Curr-Char-Is-Quote-BOOL	1431	WORKING-STORAGE	2589								
LISTING	WS-Curr-Char-Is-X-BOOL	1432	WORKING-STORAGE	2602								
LISTING	WS-Curr-Char-Is-Z-BOOL	1433	WORKING-STORAGE	2615								
LISTING	WS-Curr-CHR	1426	WORKING-STORAGE	2560*	2563	2565	2571	2579	2585	2592		
LISTING	WS-Curr-Division-TXT	1435	WORKING-STORAGE	2485*	2521	2572	2580	2680	2792*	2815*	2842*	
				2893*	2898*							
LISTING	WS-Curr-Line-NUM	1441	WORKING-STORAGE	2484*	2544*	2703*	2739*	3193	3214	3224	3358*	
LISTING	WS-Curr-Prog-ID-TXT	1443	WORKING-STORAGE	2474*	2797*	3189	3210	3220				

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)							
LISTING	WS-Curr-Section-TXT	1448	WORKING-STORAGE	2706*	2711	2820*	2847*	2888	2889*	3192	3213
				3223							
LISTING	WS-Curr-Verb-TXT	1455	WORKING-STORAGE	2475*	2499*	2523*	2907	2916	2931		
LISTING	WS-Delim-TXT	1457	WORKING-STORAGE	2632*	2634	2652*	2654	2657			
LISTING	WS-Dummy-TXT	1459	WORKING-STORAGE	2453*	2695*	2724*	2753*	2758*	2759	3344*	
LISTING	WS-Expanded-Src-Fn-TXT	1461	WORKING-STORAGE	1383	2458*	2459*					
LISTING	WS-Filename-TXT	1463	WORKING-STORAGE	2452*	2455	2723*	2726				
LISTING	WS-GI-Prog-ID-TXT	1466	WORKING-STORAGE	3237	3243	3246*					
LISTING	WS-GI-Token-TXT	1467	WORKING-STORAGE	3238	3247*	3249	3261				
LISTING	WS-Group-Indicators-TXT	1465	WORKING-STORAGE	3229*							
LISTING	WS-Held-Reference-TXT	1469	WORKING-STORAGE	2476*	2501	2502	2503*	2538	2539	2540*	3032*
				3034	3035	3036*	3065*	3071	3072	3073*	3079
				3080	3081*	3104*	3106	3107*			
LISTING	WS-I-SUB	1471	WORKING-STORAGE	2425*	2426	2427	2429	2432*	2433	2443*	2445
				2446	2447	2450	2700*	2703	2714*	2716	2717
				2718	2721	3230*	3254*	3266*	3279*	3280	3282*
				3285	3287	3315*					
LISTING	WS-J-SUB	1473	WORKING-STORAGE	2444*	2448*	2450	2715*	2719*	2721		
LISTING	WS-Lines-Left-NUM	1475	WORKING-STORAGE	3231*	3244*	3296	3311*	3316	3369*	3371*	3403*
				3430	3443*	3447					
LISTING	WS-Lines-Per-Page-Env-TXT	1479	WORKING-STORAGE	2436*	2464	2465					
LISTING	WS-Lines-Per-Page-NUM	1477	WORKING-STORAGE	2466*	2469*	3311	3443				
LISTING	WS-Lines-Per-Rec-CONST	1424	WORKING-STORAGE	2352	3280						
LISTING	WS-Main-Module-Name-TXT	1481	WORKING-STORAGE	2430*	2434*	2702	3346				
LISTING	WS-Next-Char-Is-Quote-BOOL	1484	WORKING-STORAGE	2602	2615						
LISTING	WS-Next-CHR	1483	WORKING-STORAGE	2562*	2605	2618					
LISTING	WS-OS-Type-FILLER-TXT	1486	WORKING-STORAGE	1492							
LISTING	WS-OS-Type-TXT	1493	WORKING-STORAGE	2414	2420						
LISTING	WS-OS-Types-TXT	1492	WORKING-STORAGE								
LISTING	WS-Page-No-TXT	1498	WORKING-STORAGE	3306*	3307	3438*	3439				
LISTING	WS-Page-NUM	1496	WORKING-STORAGE	2411*	3302*	3304	3434*	3436			
LISTING	WS-PN-Literal-TXT	1499	WORKING-STORAGE	3303*	3435*						
LISTING	WS-PN-Page-NUM	1500	WORKING-STORAGE	3304*	3305*	3436*	3437*				
LISTING	WS-Program-Path-TXT	1502	WORKING-STORAGE	2439*	2440						
LISTING	WS-Reserved-Word-Table-TXT	2261	WORKING-STORAGE								
LISTING	WS-Reserved-Word-TXT	2262	WORKING-STORAGE	2667	2801	2838	2899				
LISTING	WS-Reserved-Words-TXT	1504	WORKING-STORAGE	2261							
LISTING	WS-RS-Duplicate-CHR	2270	WORKING-STORAGE	3232*	3239*	3252*	3263	3264*			
LISTING	WS-RS-In-Copybook-BOOL	2273	WORKING-STORAGE	2709*	3352*						
LISTING	WS-RS-In-Main-Module-BOOL	2272	WORKING-STORAGE	2704*	2738	3347*	3357				
LISTING	WS-RS-In-Which-Pgm-CHR	2271	WORKING-STORAGE								
LISTING	WS-RS-Last-Token-Ended-Sent-CHR	2274	WORKING-STORAGE	2533*	2679						
LISTING	WS-RS-Processing-PICTURE-CHR	2275	WORKING-STORAGE	2628	2642*	2852*					
LISTING	WS-RS-Token-Ended-Sentence-CHR	2276	WORKING-STORAGE	2520	2532	2534*	2567*	2596*	2609*	2622*	2635*
				2655*							
LISTING	WS-RS-Verb-Has-Been-Found-CHR	2277	WORKING-STORAGE	2486*	2671*	2789*					
LISTING	WS-Runtime-Switches	2269	WORKING-STORAGE								
LISTING	WS-RW-IDX	2265	WORKING-STORAGE	2668	2669	2802	2803	2839	2840	2900	2901
LISTING	WS-RW-Type-CD	2266	WORKING-STORAGE	2669	2803*	2840*	2901*				
LISTING	WS-RW-Word-TXT	2267	WORKING-STORAGE	2264	2668	2802	2839	2900			
LISTING	WS-Saved-Section-TXT	2279	WORKING-STORAGE	2705	2706	2710	2711*				
LISTING	WS-SDL-Line-NUM	2282	WORKING-STORAGE	3362*							
LISTING	WS-SDL-Statement-TXT	2284	WORKING-STORAGE	3363*	3389*	3396*	3422*				
LISTING	WS-SH1-DT	2288	WORKING-STORAGE	2473*							

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)								
LISTING	WS-SH1-Title-TXT	2287	WORKING-STORAGE	2417*								
LISTING	WS-SH3-Page-No-TXT	2294	WORKING-STORAGE	3439*								
LISTING	WS-Src-Detail-Line-TXT	2281	WORKING-STORAGE	3361*	3374*	3378*	3382*	3388*	3394*	3406*	3410*	
				3414*	3420*	3445	3446*					
LISTING	WS-Src-Header-1-TXT	2286	WORKING-STORAGE	3433								
LISTING	WS-Src-Header-2-TXT	2290	WORKING-STORAGE	2440*	2441*	2442	3440					
LISTING	WS-Src-Header-3-TXT	2292	WORKING-STORAGE	3441								
LISTING	WS-Src-Header-4-TXT	2296	WORKING-STORAGE	3442								
LISTING	WS-Src-Line-NUM	2300	WORKING-STORAGE	3322*	3348	3360*	3362					
LISTING	WS-Src-SUB	2302	WORKING-STORAGE	2482*	2535	2536	2555*	2559	2561	2564*	2573*	
				2584*	2590*	2594*	2595	2597*	2603*	2607*	2608	
				2610*	2616*	2620*	2621	2623*	2633*	2636*	2653*	
				2658	2733*	2737*						
LISTING	WS-Tally-QTY	2307	WORKING-STORAGE	2768*	2770*	2771	2775*	2776				
LISTING	WS-Temp-10-Chars-TXT	2309	WORKING-STORAGE	2693*	2697	2698*	2700	3342*				
LISTING	WS-Temp-256-Chars-TXT	2319	WORKING-STORAGE	2460*	2694*	2701	2713	2716	2721	3343*	3345	
LISTING	WS-Temp-32-Chars-1-TXT	2311	WORKING-STORAGE	2744*	2747	2754						
LISTING	WS-Temp-32-Chars-2-TXT	2313	WORKING-STORAGE	2751*	2756*	2762	2763*	2779				
LISTING	WS-Temp-32-Chars-3-TXT	2315	WORKING-STORAGE	2752*	2757*	2765	2766*	2779				
LISTING	WS-Temp-65-Chars-TXT	2317	WORKING-STORAGE	3365*	3370	3373	3377	3381	3398*	3402	3405	
				3409	3413							
LISTING	WS-Today-DT	2321	WORKING-STORAGE	2471*	2472							
LISTING	WS-Token-Curr-TXT	2323	WORKING-STORAGE	2492	2496*	2565*	2574*	2585*	2593*	2606*	2619*	
				2631*	2638	2651*	2661	2663*	2681*	2744	2791	
				2797	2814	2819	2837	2846	2850	2897	2913	
				2918	3191	3212	3222					
LISTING	WS-Token-Curr-Uc-TXT	2325	WORKING-STORAGE	2493*	2496	2499	2518	2891	3190	3211	3221	
LISTING	WS-Token-Prev-TXT	2327	WORKING-STORAGE	2500*	2518*	2522*	2581*	2792	2795	2796*	2815	
				2820	2824	2842	2847	2856	2857*	2861	2864*	
				2867*	2870*	2873*	2876*	2879*	2892	2898	2906	
				2910*	2972	2975*	2982	2993	2996*	3005	3020	
				3029	3053	3062	3066*	3069*	3077*	3085*	3092	
				3101	3117	3120*	3123*	3130	3139	3150	3161	
				3164*	3171							
LISTING	WS-Token-Search-TXT	2329	WORKING-STORAGE	2661*	2662	2668						
LISTING	WS-Token-Type-CD	2331	WORKING-STORAGE	2566*	2575*	2586*	2639*	2643*	2664*	2669*	2682*	
LISTING	WS-TT-Token-Is-Angtype-BOOL	2332	WORKING-STORAGE	2786	2810	2833	2917					
LISTING	WS-TT-Token-Is-EOF-BOOL	2333	WORKING-STORAGE	2489	2543*							
LISTING	WS-TT-Token-Is-Identifiser-BOOL	2334	WORKING-STORAGE	2676*	2823	2860	2905	2928				
LISTING	WS-TT-Token-Is-Keyword-BOOL	2335	WORKING-STORAGE	2494	2517	2790	2813	2818	2836	2845	2851	
				2896								
LISTING	WS-TT-Token-Is-Lit-Alpha-BOOL	2336	WORKING-STORAGE	2599*	2625*							
LISTING	WS-TT-Token-Is-Lit-Number-BOOL	2337	WORKING-STORAGE	2612*	2678	2748*	2780*					
LISTING	WS-TT-Token-Is-Reserved-Wd-BOOL	2339	WORKING-STORAGE	2495	2787*	2811*	2834*	2855				
LISTING	WS-TT-Token-Is-Verb-BOOL	2338	WORKING-STORAGE	2498	2670							
LISTING	WS-Usernames-QTY	2341	WORKING-STORAGE	2483*								
LISTING	WS-XDL-Def-Line-NUM	2348	WORKING-STORAGE	3258*	3275*							
LISTING	WS-XDL-Prog-ID-TXT	2344	WORKING-STORAGE	3255*	3267*	3271*						
LISTING	WS-XDL-Ref-Flag-CHR	2354	WORKING-STORAGE	3287*								
LISTING	WS-XDL-Ref-Line-NUM	2353	WORKING-STORAGE	3285*								
LISTING	WS-XDL-Reference-TXT	2352	WORKING-STORAGE									
LISTING	WS-XDL-Section-TXT	2350	WORKING-STORAGE	3257*	3273*							
LISTING	WS-XDL-Token-TXT	2346	WORKING-STORAGE	3256*	3268*	3272*						
LISTING	WS-XH1-DT	2359	WORKING-STORAGE	2472*								

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)						
LISTING	WS-XH1-Title-TXT	2358	WORKING-STORAGE	2423*						
LISTING	WS-XH3-Page-No-TXT	2370	WORKING-STORAGE	3307*						
LISTING	WS-Xref-Detail-Line-TXT	2343	WORKING-STORAGE	3228*	3240	3251	3270	3290	3313	3314*
LISTING	WS-Xref-Header-1-TXT	2357	WORKING-STORAGE	3301						
LISTING	WS-Xref-Header-2-TXT	2361	WORKING-STORAGE	2442*	3308					
LISTING	WS-Xref-Header-3-TXT	2363	WORKING-STORAGE	3309						
LISTING	WS-Xref-Header-4-TXT	2372	WORKING-STORAGE	3310						

10.5. STREAMIO – A Utility Subroutine to Simplify Stream I/O

STREAMIO is a utility I created to assist with handling stream I/O functions. I've used it to construct a number of useful little command-line utilities.

Usage of this subroutine is completely documented in the program comments. The program **COPYs** a copybook named **STREAMIOcb**, the format of which is described in the program comments.

Both **STREAMIO.cbl** and **STREAMIOcb.cpy** are included in the "samples" directory of any pre-built distributions of GNU COBOL that I have created.

Line Statement

Page: 1

```

=====
1      >>SOURCE FORMAT IS FIXED
2      IDENTIFICATION DIVISION.
3      PROGRAM-ID. STREAMIO.
4      *>*****
5      *> Author: Gary L. Cutler          **
6      *>      CutlerGL@gmail.com      **
7      *>                               **
8      *> This routine centralizes all bytestream file I/O functions **
9      *> into one routine. The manner in which this routine is **
10     *> CALLED is as follows:          **
11     *>                               **
12     *>      CALL "STREAMIO" USING control-block [ , arg2 ] **
13     *>                               **
14     *> where 'control-block' is defined by the "STREAMIOcb.cpy" **
15     *> copybook and 'arg2' will vary, depending upon the function **
16     *> specified in the control block. **
17     *>                               **
18     *> The STREAMIO routine has an advantage over the various **
19     *> "CBL_XXXXXX_FILE" routines in that: **
20     *>                               **
21     *> 1. It automates the establishment and on-going adjustment of **
22     *> the file-offset value in such a way as to simplify the **
23     *> sequential processing of a bytestream file (you may still **
24     *> specify a file-offset manually on each read or write, if **
25     *> you wish)                       **
26     *>                               **
27     *> 2. It auto-detects the size of the I/O buffer you supply to **
28     *> STREAMIO, using that as the byte-count of all read and **
29     *> write operations.                **
30     *>                               **
31     *> 3. Not only does it support the raw input and output of data **
32     *> that the CBL_READ_FILE and CBL_WRITE_FILE routines do, **
33     *> but on input it is also capable of delivering just a **
34     *> single newline-delimited or carriage-return/newline de- **
35     *> limited record to the caller.    **
36     *>                               **
37     *> 4. On output, STREAMIO can optionally append either a new- **
38     *> line or carriage-return/newline sequence (your choice) to **
39     *> the end of every record it writes. **
40     *>                               **
41     *> 5. STREAMIO can automatically generate filenames for output **
42     *> files if you wish, simplifying the process of creating **
43     *> scratch or work files.            **
44     *>                               **
45     *> 6. The STREAMIO routine also allows you to (optionally) re- **
46     *> gister a general error-handling routine to be given con- **
47     *> trol should a fatal error be detected with STREAMIO. **
48     *>                               **
49     *> This routine can be "turned on" and "turned off" at will. **
50     *>                               **
51     *> The control block format is as follows. This structure must **
52     *> be defined under an 01-level data item of your creation and **
53     *> should be INITIALIZED before any items within it are used. **
54     *>                               **
=====

```

```

Line  Statement
=====
55     *> 05 SCB-Handle-NUM                PIC X(4) COMP-X.      **
56     *> 05 SCB-Mode-CD                  PIC X(1).             **
57     *>      88 SCB-Mode-Input-BOOL      VALUE 'I', 'i'.      **
58     *>      88 SCB-Mode-Output-BOOL    VALUE 'O', 'o'.      **
59     *>      88 SCB-Mode-Both-BOOL      VALUE 'B', 'b'.      **
60     *> 05 SCB-Function-CD              PIC X(2).             **
61     *>      88 SCB-Func-CLOSE-BOOL     VALUE 'C ', 'c '.    **
62     *>      88 SCB-Func-DELETE-BOOL    VALUE 'D ', 'd '.    **
63     *>      88 SCB-Func-OPEN-BOOL      VALUE 'O ', 'o '.    **
64     *>      88 SCB-Func-READ-BOOL      VALUE 'R ', 'r '.    **
65     *>      88 SCB-Func-READ-Delim-BOOL VALUE 'RD', 'rd',    **
66     *>                                'rD', 'Rd'.          **
67     *>      88 SCB-Func-WRITE-BOOL     VALUE 'W ', 'w '.    **
68     *>      88 SCB-Func-WRITE-Delim-BOOL VALUE 'WD', 'wd',    **
69     *>                                'wD', 'Wd'.          **
70     *> 05 SCB-Delimiter-Mode-CD        PIC X(1).             **
71     *>      88 SCB-Delim-Unix-BOOL     VALUE 'U', 'u'.      **
72     *>      88 SCB-Delim-Windows-BOOL  VALUE 'W', 'w'.      **
73     *> 05 SCB-Offset-NUM                PIC X(8) COMP-X.     **
74     *> 05 SCB-Error-Routine-PTR        USAGE PROGRAM-POINTER. **
75     *> 05 SCB-Error-Routine-NUM REDEFINES SCB-Error-Routine-PTR **
76     *>                                USAGE BINARY-LONG.    **
77     *> 05 SCB-Return-CD                 USAGE BINARY-LONG.    **
78     *> 05 SCB-Filename-TXT             PIC X(256).           **
79     *>                                **
80     *> Such a structure is defined for your use using the copybook **
81     *> "STREAMIOcb.cpy" (you may also define your own, provided it **
82     *> conforms to the above layout). **
83     *>----- **
84     *> SCB-Handle-NUM **
85     *>----- **
86     *> **
87     *> Serves as a file handle to the file once it has been opened **
88     *> (via the "SCB-Func-OPEN-BOOL" function). **
89     *> **
90     *>----- **
91     *> SCB-Mode-CD **
92     *>----- **
93     *> **
94     *> Prior to calling "STREAMIO" for the first time for a file, **
95     *> the appropriate subordinate level-88 must be set to TRUE to **
96     *> select an I/O mode. You may also simply move one of the **
97     *> string values listed on the level-88 items to "SCB-Mode-CD". **
98     *> **
99     *>----- **
100    *> SCB-Function-CD **
101    *>----- **
102    *> **
103    *> The appropriate subordinate level-88 must be set to TRUE to **
104    *> select a function you'd like to execute against a file. You **
105    *> may also simply move one of the string values listed on the **
106    *> level-88 items to "SCB-Function-CD". **
107    *> **
108    *> Available functions are as follows: **

```

```

Line  Statement
=====
109      *>
110      *> SCB-Func-OPEN-BOOL
111      *>
112      *>   This must be the function specified the first time you
113      *>   call STREAMIO for any given file. It opens the file &
114      *>   makes it available for use according to the
115      *>   "SCB-Mode-CD" specification.
116      *>
117      *>   The filename being opened must be specified in the
118      *>   "SCB-Filename-TXT" field.
119      *>
120      *>   The SCB-Offset-NUM field will be initialized to ZERO.
121      *>
122      *>   If "arg2" is specified in conjunction with this funct-
123      *>   ion, it will be ignored.
124      *>
125      *> SCB-Func-CLOSE-BOOL
126      *>
127      *>   This function should be the one specified the LAST time
128      *>   you call STREAMIO against a specific file. After this
129      *>   function has been executed, you'll have to re-open the
130      *>   file if you wish to use it with STREAMIO again.
131      *>
132      *>   The SCB-Handle-NUM item will be reset to ZERO.
133      *>
134      *>   If "arg2" is specified in conjunction with this funct-
135      *>   ion, it will be ignored.
136      *>
137      *> SCB-Func-DELETE-BOOL
138      *>
139      *>   This function will delete the file specified in the
140      *>   control block (see SCB-Filename-TXT).
141      *>
142      *>   This function should not be performed against a file
143      *>   that is open.
144      *>
145      *>   If "arg2" is specified in conjunction with this funct-
146      *>   ion, it will be ignored.
147      *>
148      *> SCB-Func-READ-BOOL
149      *>
150      *>   This function invokes a standard CBL_READ_FILE against
151      *>   the file specified in the control block (see
152      *>   SCB-Filename-TXT).
153      *>
154      *>   The buffer into which you wish to read data must be
155      *>   supplied as "arg2". The size of that buffer, in bytes,
156      *>   will define the "byte-count" value supplied to the
157      *>   CBL_READ_FILE subroutine. The buffer data item will be
158      *>   set to SPACES before the read takes place.
159      *>
160      *>   If the file-offset value (SCB-Offset-NUM) is greater
161      *>   than the size of the file, a "no more data" return code
162      *>   (01) will be passed back in SCB-Return-CD and the

```

```

Line  Statement
=====
163      *>      buffer will have been set to SPACES.          **
164      *>      **
165      *>      At the conclusion of a successful SCB-Func-READ-BOOL, **
166      *>      the value of SCB-Offset-NUM will have been automati- **
167      *>      cally incremented by the byte-count size of "arg2". **
168      *>      **
169      *> SCB-Func-WRITE-BOOL **
170      *>      **
171      *>      This function invokes a standard CBL_WRITE_FILE against **
172      *>      the file specified in the control block (see **
173      *>      SCB-Filename-TXT). **
174      *>      **
175      *>      The buffer from which data will be written to the file **
176      *>      must be supplied as "arg2". The size of that buffer, **
177      *>      in bytes, will define the "byte-count" value supplied **
178      *>      CBL_WRITE_FILE subroutine. The buffer data will be **
179      *>      written to the file-offset position defined by the **
180      *>      SCB-Offset-NUM value. You may specify "arg2" either **
181      *>      as an actual alphanumeric data item or as an alpha- **
182      *>      numeric literal. **
183      *>      **
184      *>      If the file-offset value (SCB-Offset-NUM) is greater **
185      *>      than the size of the file, a "no more data" return code **
186      *>      will be passed back in SCB-Return-CD and the buffer **
187      *>      will have been set to SPACES. **
188      *>      **
189      *>      At the conclusion of a successful SCB-Func-WRITE-BOOL **
190      *>      operation, the value of SCB-Offset-NUM will have been **
191      *>      automatically incremented by the byte-count size of **
192      *>      "arg2". **
193      *>      **
194      *> SCB-Func-READ-Delim-BOOL **
195      *>      **
196      *>      SCB-Func-READ-Delim-BOOL bahaves like the SCB-FUNC- **
197      *>      READ function, with the following behavioral dif- **
198      *>      ferences: **
199      *>      **
200      *>      1. When data is read from the file, only that data read **
201      *>      up to BUT NOT INCLUDING an end-of-line delimiter **
202      *>      sequence (either a LF or CRLF) will be retained in **
203      *>      the buffer - the remainder of the buffer from the **
204      *>      end-of-line sequence onward will be reset to SPACES. **
205      *>      The file-offset value (SCB-Offset-NUM) will be in- **
206      *>      cremented ONLY by the amount of data transferred up **
207      *>      to AND INCLUDING the end-of-line sequence. **
208      *>      **
209      *>      2. When data is read from the file and an end-of-line **
210      *>      delimiter sequence (either a LF or a CRLF) cannot be **
211      *>      found within the buffer, the assumption is made that **
212      *>      the record is too long to fit within the buffer. In **
213      *>      these instances, an SCB-Return-CD value of 02 will **
214      *>      be returned and the SCB-Offset-NUM value will be **
215      *>      incremented past the next end-of-line sequence in **
216      *>      the file (this will involve at least one additional **

```



```

Line  Statement
=====
217      *>      call to CBL_READ_FILE to locate that eol sequence, **
218      *>      but any additional such reads will be done internal- **
219      *>      ly to STREAMIO and will be entirely transparent to **
220      *>      the caller of STREAMIO. **
221      *> **
222      *>      DO NOT USE the Streamio-READ-Delim function if the **
223      *>      possibility exists that linefeed (X"0A") or carriage- **
224      *>      return (X"0D") characters could exist as actual data **
225      *>      characters in the file. **
226      *> **
227      *> SCB-Func-WRITE-Delim-BOOL **
228      *> **
229      *>      SCB-Func-WRITE-Delim-BOOL acts like the Streamio- **
230      *>      FUNC-WRITE function, with the following difference: **
231      *> **
232      *>      After the specified data is written to the file, an **
233      *>      end-of-line sequence will also be written to the file. **
234      *>      The file-offset value (SCB-Value) will be incremented **
235      *>      by the byte-count size of the data PLUS the size of the **
236      *>      end-of-line sequence. One of two possible end-of-line **
237      *>      sequences must be specified using the value of SCB- **
238      *>      Delimiter-Mode. **
239      *> **
240      *>----- **
241      *> SCB-Delimiter-Mode-CD **
242      *>----- **
243      *> **
244      *> This data item is needed only when issuing the Streamio- **
245      *>      FUNC-WRITE-Delim function. In those circumstances, this **
246      *>      item defines what end-of-line delimiter sequence is to be **
247      *>      written: **
248      *> **
249      *> If SCB-Delim-Unix-BOOL is true, a linefeed character will **
250      *>      be written. **
251      *> **
252      *> If SCB-Delim-Windows-BOOL is true, a carriage-return and **
253      *>      linefeed sequence will be written. **
254      *> **
255      *>----- **
256      *> SCB-Offset-NUM **
257      *>----- **
258      *> **
259      *> This data item specifies the next relative byte number with- **
260      *>      in the file where the next read or write will start. **
261      *> **
262      *> SCB-Offset-NUM is automatically set to 0 (the first byte) **
263      *>      when the file is opened, and is automatically incremented as **
264      *>      the file is read or written via STREAMIO. **
265      *> **
266      *> You may also manually set this value as desired before any **
267      *>      call to STREAMIO. **
268      *> **
269      *>----- **
270      *> SCB-Error-Routine-PTR **

```

Line Statement

Page: 6

```

=====
271      *>-----**
272      *> **
273      *> To specify a general error-handling routine for handling **
274      *> STREAMIO failures, Create the routine and define an entry- **
275      *> name for it via the ENTRY statement. Then use the following **
276      *> to set that routine up as the error handler: **
277      *> **
278      *>     SET SCB-Error-Routine-PTR TO ENTRY "entry-name" **
279      *> **
280      *> To "turn off" the error-routine: **
281      *> **
282      *>     SET SCB-Error-Routine-PTR TO NULL **
283      *> **
284      *> If a fatal error occurs (any error not marked with a ">" in **
285      *> the SCB-Return-CD discussion), the error routine you spe- **
286      *> cified (if any) will be set up as an exit routine via the **
287      *> CBL_EXIT_PROC subroutine; the STREAMIO routine will then is- **
288      *> sue a STOP RUN to intentionally trigger your error routine. **
289      *> You will not be able to recover your program once your error **
290      *> routine triggers. If you wish to be able to recover from **
291      *> fatal STREAMIO errors, you should NOT use the SCB-Error- **
292      *> Routine feature but instead you should explicitly test the **
293      *> SCB-Return-CD value after every call to STREAMIO. **
294      *> **
295      *> A default error routine is defined by the "STREAMIOError.cpy" **
296      *> copybook. **
297      *> **
298      *>-----**
299      *> SCB-Return-CD **
300      *>-----**
301      *> **
302      *> The following are the possible SCB-Return-CD values. The **
303      *> ones marked with a ">" will NOT trigger an error-routine, if **
304      *> one is currently registered via SCB-Error-Routine-PTR. **
305      *> **
306      *>     12 I/O error writing to file **
307      *>     11 File does not exist **
308      *>     10 File already OPEN or already CLOSEd **
309      *> > 02 READ-Delim was truncated **
310      *> > 01 No more data is available from the current **
311      *>     SCB-Offset-NUM **
312      *> > 00 OK - the operation was successful **
313      *>     -1 Invalid SCB-Function-CD **
314      *>     -2 Invalid SCB-Mode-CD **
315      *>     -3 CBL_XXXXX_FILE routine rejected operation **
316      *>     -4 Invalid delimiter mode specified (Not U/W) **
317      *> **
318      *>-----**
319      *> SCB-Filename-TXT **
320      *>-----**
321      *> **
322      *> This is the name of the file you wish to access. **
323      *> **
324      *> If you are planning on reading the file, the file MUST exist **

```

Line Statement

Page: 7

```

=====
325     *> at the time the SCB-Func-OPEN-BOOL is executed.      **
326     *>                                                       **
327     *> If you are planning on writing to the file, the file need **
328     *> exist when the SCB-Func-OPEN-BOOL is issued.         **
329     *>                                                       **
330     *> In general, the contents of SCB-Filename-TXT should re- **
331     *> flect the complete path to the file as well as the name of **
332     *> the file itself, unless the file is contained in whatever **
333     *> directory is current at the time the SCB-Func-OPEN-BOOL is **
334     *> executed.                                             **
335     *>                                                       **
336     *> The following special values may be used for         **
337     *> SCB-Filename-TXT:                                     **
338     *>                                                       **
339     *> SPACES If the filename is SPACES, a filename will be created **
340     *> automatically for you in whatever directory is de- **
341     *> fined by the TEMP environment variable. If there IS **
342     *> no TEMP variable defined, the "/tmp" folder will be **
343     *> assumed. The filename will be STREAMIO-nnnnnnnn.dat **
344     *> where "nnnnnnnn" is a random number.                 **
345     *>                                                       **
346     *> . If you specify only a dot (period) as the filename, **
347     *> the behavior will be the same as with a value of **
348     *> SPACES except there will be no ".dat" at the end of **
349     *> the generated filename.                               **
350     *>                                                       **
351     *> .ext If you specify a filename extension prefixed with a **
352     *> dot (period), the behavior will be the same as if a **
353     *> value of SPACES were specified, except that the given **
354     *> extension will be used instead of ".dat". Note that **
355     *> if you are using a Unix/Cygwin implementation of **
356     *> OpenCOBOL and you'd like to specify a hidden file in **
357     *> the current directory as the SCB-Filename-TXT, you **
358     *> MUST code the filename as "./.xxxxx" to avoid having **
359     *> it treated as this special name.                       **
360     *>                                                       **
361     *>*****
362     ENVIRONMENT DIVISION.
363     CONFIGURATION SECTION.
364     REPOSITORY.
365         FUNCTION ALL INTRINSIC.
366     DATA DIVISION.
367     WORKING-STORAGE SECTION.
368     01 WS-Access-Mode-CD          PIC X(1) COMP-X.
369     01 WS-Arg-Length-NUM         PIC X(4) COMP-X.
370     01 WS-Buffer-TXT            PIC X(256).
371     01 WS-Delim-Buffer-TXT     PIC X(2).
372     01 WS-Env-Temp-TXT         PIC X(256).
373     01 WS-Slash-CHR            PIC X(1).
374     01 WS-Tally-NUM            USAGE BINARY-LONG.
375     01 WS-8-Digit-NUM          PIC 9(8).
376     01 WS-256-Byte-TXT        PIC X(256).
377     LINKAGE SECTION.
378     01 L-StreamIO-Control-Block-TXT.
=====

```

Line Statement

Page: 8

```

=====
379          COPY STREAMIOcb
380          REPLACING LEADING ==SCB== BY ==L-SCB==.
          05 L-SCB-Handle-NUM PIC X(4) COMP-X.
          05 L-SCB-Mode-CD PIC X(1).
          88 L-SCB-MODE-Input-BOOL VALUE 'I' 'i'.
          88 L-SCB-MODE-Output-BOOL VALUE 'O' 'o'.
          88 L-SCB-MODE-Both-BOOL VALUE 'B' 'b'.
          05 L-SCB-Function-CD PIC X(2).
          88 L-SCB-Func-CLOSE-BOOL VALUE 'C' 'c'.
          88 L-SCB-Func-DELETE-BOOL VALUE 'D' 'd'.
          88 L-SCB-Func-OPEN-BOOL VALUE 'O' 'o'.
          88 L-SCB-Func-READ-BOOL VALUE 'R' 'r'.
          88 L-SCB-Func-READ-Delim-BOOL VALUE 'RD' 'rd'
          'rD' 'Rd'.
          88 L-SCB-Func-WRITE-BOOL VALUE 'W' 'w'.
          88 L-SCB-Func-WRITE-Delim-BOOL VALUE 'WD' 'wd'
          'wD' 'Wd'.
          05 L-SCB-Delimiter-Mode-CD PIC X(1).
          88 L-SCB-DELIM-Unix-BOOL VALUE 'U' 'u'.
          88 L-SCB-DELIM-Windows-BOOL VALUE 'W' 'w'.
          05 L-SCB-Offset-NUM PIC X(8) COMP-X.
          05 L-SCB-Error-Routine-PTR USAGE PROGRAM-POINTER.
          05 L-SCB-Error-Routine-NUM REDEFINES L-SCB-Error-Routine-PTR
          USAGE BINARY-LONG.
          05 L-SCB-Return-CD USAGE BINARY-LONG.
          05 L-SCB-Filename-TXT PIC X(256).
381          01 L-Arg2-TXT                                PIC X ANY LENGTH.
382          PROCEDURE DIVISION USING L-StreamIO-Control-Block-TXT,
383          L-Arg2-TXT.
384          000-Main SECTION.
385          MOVE 00 TO L-SCB-Return-CD
386          EVALUATE TRUE
387              WHEN L-SCB-Func-CLOSE-BOOL
388                  PERFORM 030-Validate-Handle-NonZero
389                  PERFORM 200-CLOSE
390              WHEN L-SCB-Func-DELETE-BOOL
391                  CALL "CBL_DELETE_FILE" USING L-SCB-Filename-TXT
392              WHEN L-SCB-Func-OPEN-BOOL
393                  PERFORM 020-Validate-Handle-Zero
394                  PERFORM 100-OPEN
395              WHEN L-SCB-Func-READ-BOOL
396                  PERFORM 030-Validate-Handle-NonZero
397                  PERFORM 400-READ
398              WHEN L-SCB-Func-READ-Delim-BOOL
399                  PERFORM 030-Validate-Handle-NonZero
400                  PERFORM 500-READ-Delim
401              WHEN L-SCB-Func-WRITE-BOOL
402                  PERFORM 030-Validate-Handle-NonZero
403                  PERFORM 300-WRITE
404              WHEN L-SCB-Func-WRITE-Delim-BOOL
405                  EVALUATE TRUE
406                      WHEN L-SCB-Delim-Unix-BOOL
407                          PERFORM 030-Validate-Handle-NonZero
408                          PERFORM 300-WRITE

```

Line Statement

Page: 9

```

=====
409             MOVE 1 TO WS-Arg-Length-NUM
410             MOVE X"0A" TO WS-Delim-Buffer-TXT
411             WHEN L-SCB-Delim-Windows-BOOL
412                 PERFORM 030-Validate-Handle-NonZero
413                 PERFORM 300-WRITE
414                 MOVE 2 TO WS-Arg-Length-NUM
415                 MOVE X"0D0A" TO WS-Delim-Buffer-TXT
416             WHEN OTHER
417                 MOVE -4 TO L-SCB-Return-CD
418                 PERFORM 099-ERROR-Return
419             END-EVALUATE
420             CALL "CBL_WRITE_FILE" USING L-SCB-Handle-NUM
421                                     L-SCB-Offset-NUM
422                                     WS-Arg-Length-NUM
423                                     0
424                                     WS-Delim-Buffer-TXT
425             PERFORM 040-Check-WRITE-SCB-Return-CD
426             ADD WS-Arg-Length-NUM TO L-SCB-Offset-NUM
427             WHEN OTHER
428                 MOVE -1 TO L-SCB-Return-CD
429                 PERFORM 099-ERROR-Return
430             END-EVALUATE
431             GOBACK
432             .
433             020-Validate-Handle-Zero SECTION.
434             IF L-SCB-Handle-NUM NOT = ZERO
435                 MOVE 10 TO L-SCB-Return-CD
436                 PERFORM 099-ERROR-Return
437             END-IF
438             .
439             030-Validate-Handle-NonZero SECTION.
440             IF L-SCB-Handle-NUM = ZERO
441                 MOVE 10 TO L-SCB-Return-CD
442                 PERFORM 099-ERROR-Return
443             END-IF
444             .
445             040-Check-WRITE-SCB-Return-CD SECTION.
446             IF RETURN-CODE < 0
447                 MOVE -3 TO L-SCB-Return-CD
448                 PERFORM 099-ERROR-Return
449             END-IF
450             IF RETURN-CODE = 30
451                 MOVE 12 TO L-SCB-Return-CD
452                 PERFORM 099-ERROR-Return
453             END-IF
454             MOVE 00 TO L-SCB-Return-CD
455             .
456             050-Check-READ-SCB-Return-CD SECTION.
457             IF RETURN-CODE < 0
458                 MOVE -3 TO L-SCB-Return-CD
459                 PERFORM 099-ERROR-Return
460             END-IF
461             IF RETURN-CODE = 10
462                 MOVE 01 TO L-SCB-Return-CD

```

```

Line  Statement
=====
463         GOBACK
464         END-IF
465         MOVE 00 TO L-SCB-Return-CD
466         .
467     060-Identify-TEMP SECTION.
468         ACCEPT WS-Env-Temp-TXT FROM ENVIRONMENT "TEMP"
469         EVALUATE TRUE
470             WHEN WS-Env-Temp-TXT (1:1) = "/"
471                 MOVE "/" TO WS-Slash-CHR
472             WHEN WS-Env-Temp-TXT (2:1) = ":"
473                 MOVE "\" TO WS-Slash-CHR
474             WHEN OTHER
475                 MOVE "/tmp" TO WS-Env-Temp-TXT
476                 MOVE "/" TO WS-Slash-CHR
477         END-EVALUATE
478         .
479     099-ERROR-Return SECTION.
480         IF L-SCB-Error-Routine-NUM NOT = 0
481             CALL "CBL_EXIT_PROC" USING 0, L-SCB-Error-Routine-PTR
482             STOP RUN
483         END-IF
484         GOBACK
485         .
486     100-OPEN SECTION.
487         IF (L-SCB-Mode-Input-BOOL OR L-SCB-Mode-Both-BOOL)
488             AND (L-SCB-Filename-TXT = SPACES OR LOW-VALUES)
489             MOVE 11 TO L-SCB-Return-CD
490             PERFORM 099-ERROR-Return
491         END-IF
492         EVALUATE TRUE
493             WHEN L-SCB-Filename-TXT = SPACES OR LOW-VALUES
494                 PERFORM 060-Identify-TEMP
495                 MOVE SPACES TO L-SCB-Filename-TXT
496                 COMPUTE
497                     WS-8-Digit-NUM =
498                     RANDOM(SECONDS-PAST-MIDNIGHT) * 100000000
499                 END-COMPUTE
500                 STRING
501                     TRIM(WS-Env-Temp-TXT,TRAILING)
502                     WS-Slash-CHR
503                     "STREAMIO-"
504                     WS-8-Digit-NUM
505                     ".dat"
506                 DELIMITED BY SIZE
507                 INTO L-SCB-Filename-TXT
508             WHEN L-SCB-Filename-TXT(1:1) = "."
509                 PERFORM 060-Identify-TEMP
510                 IF L-SCB-Filename-TXT(2:1) = SPACE
511                     MOVE SPACES TO WS-256-Byte-TXT
512                 ELSE
513                     MOVE L-SCB-Filename-TXT TO WS-256-Byte-TXT
514             END-IF
515         MOVE SPACES TO L-SCB-Filename-TXT
516         COMPUTE WS-8-Digit-NUM =

```

Line Statement

Page: 11

```

=====
517          RANDOM(SECONDS-PAST-MIDNIGHT) * 100000000
518          STRING
519          TRIM(WS-Env-Temp-TXT,TRAILING)
520          WS-Slash-CHR
521          "STREAMIO-"
522          WS-8-Digit-NUM
523          TRIM(WS-256-Byte-TXT,TRAILING)
524          DELIMITED BY SIZE
525          INTO L-SCB-Filename-TXT
526      END-EVALUATE
527      EVALUATE TRUE
528          WHEN L-SCB-Mode-Input-BOOL
529              MOVE 1 TO WS-Access-Mode-CD
530          WHEN L-SCB-Mode-Output-BOOL
531              MOVE 2 TO WS-Access-Mode-CD
532          WHEN L-SCB-Mode-Both-BOOL
533              MOVE 3 TO WS-Access-Mode-CD
534          WHEN OTHER
535              MOVE -2 TO L-SCB-Return-CD
536              PERFORM 099-ERROR-Return
537      END-EVALUATE
538      CALL "CBL_OPEN_FILE" USING TRIM(L-SCB-Filename-TXT,TRAILING)
539                               WS-Access-Mode-CD
540                               0
541                               0
542                               L-SCB-Handle-NUM
543      IF RETURN-CODE = 35
544          MOVE 11 TO L-SCB-Return-CD
545          PERFORM 099-ERROR-Return
546      END-IF
547      IF RETURN-CODE < 0
548          MOVE -2 TO L-SCB-Return-CD
549          PERFORM 099-ERROR-Return
550      END-IF
551      MOVE 00 TO L-SCB-Return-CD
552      MOVE 0 TO L-SCB-Offset-NUM
553      .
554 200-CLOSE SECTION.
555      CALL "CBL_CLOSE_FILE" USING L-SCB-Handle-NUM
556      IF RETURN-CODE < 0
557          MOVE -2 TO L-SCB-Return-CD
558          PERFORM 099-ERROR-Return
559      END-IF
560      MOVE 00 TO L-SCB-Return-CD
561      MOVE 0 TO L-SCB-Handle-NUM
562      .
563 300-WRITE SECTION.
564      CALL "C$PARAMSIZE" USING 2
565      MOVE RETURN-CODE TO WS-Arg-Length-NUM
566      CALL "CBL_WRITE_FILE" USING L-SCB-Handle-NUM
567                               L-SCB-Offset-NUM
568                               WS-Arg-Length-NUM
569                               0
570                               L-Arg2-TXT
=====

```

Line Statement

Page: 12

```

=====
571      PERFORM 040-Check-WRITE-SCB-Return-CD
572      ADD WS-Arg-Length-NUM TO L-SCB-Offset-NUM
573      .
574      400-READ SECTION.
575      CALL "C$PARAMSIZE" USING 2
576      MOVE RETURN-CODE TO WS-Arg-Length-NUM
577      MOVE SPACES TO L-Arg2-TXT(1:WS-Arg-Length-NUM)
578      CALL "CBL_READ_FILE" USING L-SCB-Handle-NUM
579      L-SCB-Offset-NUM
580      WS-Arg-Length-NUM
581      0
582      L-Arg2-TXT
583      PERFORM 050-Check-READ-SCB-Return-CD
584      ADD WS-Arg-Length-NUM TO L-SCB-Offset-NUM
585      .
586      500-READ-Delim SECTION.
587      CALL "C$PARAMSIZE" USING 2
588      MOVE RETURN-CODE TO WS-Arg-Length-NUM
589      MOVE SPACES TO L-Arg2-TXT(1:WS-Arg-Length-NUM)
590      CALL "CBL_READ_FILE" USING L-SCB-Handle-NUM
591      L-SCB-Offset-NUM
592      WS-Arg-Length-NUM
593      0
594      L-Arg2-TXT
595      PERFORM 050-Check-READ-SCB-Return-CD
596      MOVE 0 TO WS-Tally-NUM
597      INSPECT L-Arg2-TXT(1:WS-Arg-Length-NUM)
598      TALLYING WS-Tally-NUM FOR ALL X"0A"
599      IF WS-Tally-NUM = 0 *> No LF found - return truncated data and position past next LF (if any)
600      IF L-Arg2-TXT(WS-Arg-Length-NUM:1) = X"0D"
601      MOVE SPACE TO L-Arg2-TXT(WS-Arg-Length-NUM:1)
602      END-IF
603      ADD WS-Arg-Length-NUM TO L-SCB-Offset-NUM
604      MOVE 02 TO L-SCB-Return-CD
605      MOVE 256 TO WS-Arg-Length-NUM
606      PERFORM UNTIL 0 = 1
607      MOVE SPACES TO WS-Buffer-TXT
608      CALL "CBL_READ_FILE" USING L-SCB-Handle-NUM
609      L-SCB-Offset-NUM
610      WS-Arg-Length-NUM
611      0
612      WS-Buffer-TXT
613      IF RETURN-CODE < 0
614      MOVE -3 TO L-SCB-Return-CD
615      PERFORM 099-ERROR-Return
616      END-IF
617      IF RETURN-CODE = 10
618      GOBACK
619      END-IF
620      MOVE 0 TO WS-Tally-NUM
621      INSPECT WS-Buffer-TXT
622      TALLYING WS-Tally-NUM FOR ALL X"0A"
623      IF WS-Tally-NUM = 0
624      ADD 256 TO L-SCB-Offset-NUM
=====

```


Line Statement

Page: 13

```
=====
625             ELSE
626                 MOVE 0 TO WS-Tally-NUM
627                 INSPECT WS-Buffer-TXT
628                     TALLYING WS-Tally-NUM
629                     FOR CHARACTERS BEFORE INITIAL X"0A"
630                 ADD WS-Tally-NUM, 1 TO L-SCB-Offset-NUM
631                 GOBACK
632             END-IF
633         END-PERFORM
634     ELSE *> There is (at least) one LF in the buffer
635         MOVE 0 TO WS-Tally-NUM
636         INSPECT L-Arg2-TXT(1:WS-Arg-Length-NUM)
637             TALLYING WS-Tally-NUM
638             FOR CHARACTERS BEFORE INITIAL X"0A"
639         ADD WS-Tally-NUM, 1 TO L-SCB-Offset-NUM
640         IF WS-Tally-NUM > 1
641             IF L-Arg2-TXT(WS-Tally-NUM:1) = X"0D"
642                 COMPUTE WS-Arg-Length-NUM =
643                     WS-Arg-Length-NUM
644                     - WS-Tally-NUM
645                     + 1
646             ELSE
647                 COMPUTE WS-Arg-Length-NUM =
648                     WS-Arg-Length-NUM
649                     - WS-Tally-NUM
650                 ADD 1 TO WS-Tally-NUM
651             END-IF
652         MOVE SPACES
653             TO L-Arg2-TXT(WS-Tally-NUM:WS-Arg-Length-NUM)
654     ELSE
655         MOVE SPACES
656             TO L-Arg2-TXT(1:WS-Arg-Length-NUM)
657     END-IF
658 END-IF
659 .
```

GNU COBOL V2.0 11FEB2012 Cross-Reference Listing - GCic for Windows/MinGW Copyright (C) 2009 - 2013, Gary L. Cutler, GPL 2013/11/21
 E:/GNU-COBOL/samples/STREAMIO.cbl Page: 14

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)								
STREAMIO	000-Main	384	PROCEDURE									
STREAMIO	020-Validate-Handle-Zero	433	PROCEDURE	393								
STREAMIO	030-Validate-Handle-NonZero	439	PROCEDURE	388	396	399	402	407	412			
STREAMIO	040-Check-WRITE-SCB-Return-CD	445	PROCEDURE	425	571							
STREAMIO	050-Check-READ-SCB-Return-CD	456	PROCEDURE	583	595							
STREAMIO	060-Identify-TEMP	467	PROCEDURE	494	509							
STREAMIO	099-ERROR-Return	479	PROCEDURE	418	429	436	442	448	452	459	490	
				536	545	549	558	615				
STREAMIO	100-OPEN	486	PROCEDURE	394								
STREAMIO	200-CLOSE	554	PROCEDURE	389								
STREAMIO	300-WRITE	563	PROCEDURE	403	408	413						
STREAMIO	400-READ	574	PROCEDURE	397								
STREAMIO	500-READ-Delim	586	PROCEDURE	400								
STREAMIO	L-Arg2-TXT	381	LINKAGE	383	570*	577*	582*	589*	594*	600	601*	
				641	653*	656*						
STREAMIO	L-SCB-DELIM-Unix-BOOL	380	[STREAMIOcb]	406								
STREAMIO	L-SCB-DELIM-Windows-BOOL	380	[STREAMIOcb]	411								
STREAMIO	L-SCB-Delimiter-Mode-CD	380	[STREAMIOcb]									
STREAMIO	L-SCB-Error-Routine-NUM	380	[STREAMIOcb]	480								
STREAMIO	L-SCB-Error-Routine-PTR	380	[STREAMIOcb]	380	481*							
STREAMIO	L-SCB-Filename-TXT	380	[STREAMIOcb]	391*	488	493	495*	507*	508	510	513	
				515*	525*	538*						
STREAMIO	L-SCB-Func-CLOSE-BOOL	380	[STREAMIOcb]	387								
STREAMIO	L-SCB-Func-DELETE-BOOL	380	[STREAMIOcb]	390								
STREAMIO	L-SCB-Func-OPEN-BOOL	380	[STREAMIOcb]	392								
STREAMIO	L-SCB-Func-READ-BOOL	380	[STREAMIOcb]	395								
STREAMIO	L-SCB-Func-READ-Delim-BOOL	380	[STREAMIOcb]	398								
STREAMIO	L-SCB-Func-WRITE-BOOL	380	[STREAMIOcb]	401								
STREAMIO	L-SCB-Func-WRITE-Delim-BOOL	380	[STREAMIOcb]	404								
STREAMIO	L-SCB-Function-CD	380	[STREAMIOcb]									
STREAMIO	L-SCB-Handle-NUM	380	[STREAMIOcb]	420*	434	440	542*	555*	561*	566*	578*	
				590*	608*							
STREAMIO	L-SCB-MODE-Both-BOOL	380	[STREAMIOcb]	487	532							
STREAMIO	L-SCB-Mode-CD	380	[STREAMIOcb]									
STREAMIO	L-SCB-MODE-Input-BOOL	380	[STREAMIOcb]	487	528							
STREAMIO	L-SCB-MODE-Output-BOOL	380	[STREAMIOcb]	530								
STREAMIO	L-SCB-Offset-NUM	380	[STREAMIOcb]	421*	426*	552*	567*	572*	579*	584*	591*	
				603*	609*	624*	630*	639*				
STREAMIO	L-SCB-Return-CD	380	[STREAMIOcb]	385*	417*	428*	435*	441*	447*	451*	454*	
				458*	462*	465*	489*	535*	544*	548*	551*	
				557*	560*	604*	614*					
STREAMIO	L-StreamIO-Control-Block-TXT	378	LINKAGE	382								
STREAMIO	RANDOM		PROCEDURE	498	517							
STREAMIO	RETURN-CODE		PROCEDURE	446	450	457	461	543	547	556	565	
				576	588	613	617					
STREAMIO	SECONDS-PAST-MIDNIGHT		PROCEDURE	498	517							
STREAMIO	TRIM		PROCEDURE	501	519	523	538*					
STREAMIO	WS-256-Byte-TXT	376	WORKING-STORAGE	511*	513*	523						
STREAMIO	WS-8-Digit-NUM	375	WORKING-STORAGE	497*	504	516*	522					
STREAMIO	WS-Access-Mode-CD	368	WORKING-STORAGE	529*	531*	533*	539*					
STREAMIO	WS-Arg-Length-NUM	369	WORKING-STORAGE	409*	414*	422*	426	565*	568*	572	576*	
				577	580*	584	588*	589	592*	597	600	
				601	603	605*	610*	636	642*	643	647*	
				648	653	656						

GNU COBOL V2.0 11FEB2012 Cross-Reference Listing - GCic for Windows/MinGW Copyright (C) 2009 - 2013, Gary L. Cutler, GPL 2013/11/21
 E:/GNU-COBOL/samples/STREAMIO.cbl Page: 15

PROGRAM-ID	Identifier/Register/Function	Defn	Where Defined	References (* = Updated)							
=====				=====							
STREAMIO	WS-Buffer-TXT	370	WORKING-STORAGE	607*	612*						
STREAMIO	WS-Delim-Buffer-TXT	371	WORKING-STORAGE	410*	415*	424*					
STREAMIO	WS-Env-Temp-TXT	372	WORKING-STORAGE	468*	470	472	475*	501	519		
STREAMIO	WS-Slash-CHR	373	WORKING-STORAGE	471*	473*	476*	502	520			
STREAMIO	WS-Tally-NUM	374	WORKING-STORAGE	596*	598*	599	620*	622*	623	626*	628*
				630	635*	637*	639	640	641	644	649
				650*	653						

11. Glossary of Terms

There are many terms that are used throughout this document (as well as throughout ANY document dealing with the COBOL language) that are used to make discussions of syntax and semantics more concise. The following is a list of such terms and their definitions.

Alphanumeric Literal	A string of characters enclosed within a pair of quotation marks (“”) or apostrophes (‘’). See section 1.8 .
Collating Sequence	The sequence in which the characters that are acceptable to a computer are ordered for purposes of all types of sorting, merging, comparing, and processing. GNU COBOL programs may utilize standard character-set collating sequences (such as that defined by the ASCII or EBCDIC character sets) or programmer-defined custom sequences as specified in the OBJECT-COMPUTER paragraph (section 4.1.2) and defined in the SPECIAL-NAMES paragraph (section 4.1.4).
Compilation Group	The collection of all compilation units being compiled by a single execution of the GNU COBOL compiler.
Compilation Unit	A single source file being compiled by the GNU COBOL compiler. A compilation unit may contain one or more programs .
Division	<p>COBOL programs are broken into four major areas, called DIVISIONS. Divisions are used to collect program components oriented toward specific similar goals together in a single place. The COBOL divisions are:</p> <ul style="list-style-type: none"> ▶ IDENTIFICATION DIVISION – names the program and, optionally, if it is a subprogram, defines its high-level data initialization policy and/or global availability to other programs compiled in the same compilation group. ▶ ENVIRONMENT DIVISION – defines characteristics of the environment in which the program will be executed, such as files the program will be reading and/or writing, run-time switches that may be used to pass information into the program from the operating system environment and any special options that may be needed in order for the program to properly compile; typically, those special options are used to enable COBOL programs created using some other version of COBOL to be compiled and executed under a different version. ▶ DATA DIVISION – provides detailed descriptions of the files, data and data structures the program will be working with. ▶ PROCEDURE DIVISION – contains the actual executable program code.
Dynamically-loadable library	The GNU COBOL compiler can create dynamically-loadable library files when compiling subprograms as their own separate compilation groups . On UNIX systems, these will be “.so” files while on Windows systems these will be DLLs. Main programs can be created in this manner also. The “-m” compiler switch is used to create dynamically-loadable libraries.
Dynamically-loadable module	A synonym for Dynamically-loadable library .
Elementary Item	A data item described as not being further logically subdivided.
Entry-point	A spot in the PROCEDURE DIVISION where a program may begin execution when it is executed from the operating system, invoked as a user-defined function or CALL ed by another program. Every program has at least one entry-point – known as the <i>primary entry-point</i> – which corresponds to the first executable statement in the PROCEDURE DIVISION following the DECLARATIVES area, if any. Additional entry-points may be defined via the ENTRY statement (see section 6.4.14).
Entry-point name	Every <i>entry-point</i> has a name. That name must be unique for all programs that comprise an executable program. Entry-point names are defined using a subroutine’s PROGRAM-ID clause (see section 3) or via ENTRY statements coded in the subroutine’s PROCEDURE DIVISION (see section 6.4.14).

Executable file	The GNU COBOL compiler can create operating-system appropriate files that may be executed directly from the operating system environment. On Windows systems, these will be “.exe” files whereas on UNIX systems they will have no specific extensions. The “-x” compiler switch is used to create executable files. Only main programs should be compiled in this manner.
Figurative constants	GNU COBOL, like other COBOL implementations, supports a number of reserved words that may be used to represent a specific literal value. These are known as figurative constants. See section 1.9 .
Group item	A group item is an identifier that is broken down into sub-items. For example, a MAILING-ADDRESS might be broken down into STREET-ADDRESS, APARTMENT-NUMBER, CITY, STATE and ZIP-CODE components.
Identifiers	These are data items a COBOL program will be working with. The vast majority of identifiers are defined by the user (programmer) while a few are pre-defined by the GNU COBOL compiler. Identifiers pre-defined by the compiler are referred to as registers . Other programming languages generally refer to identifiers as “variables”.
Imperative statement	<p>There are two types of GNU COBOL statements that meet this definition:</p> <ol style="list-style-type: none"> 1. A non-conditional GNU COBOL statement; i.e. one that performs an unconditional action and lacks any decision-making capabilities (including EXCEPTION, ON SIZE ERROR and AT END clauses), or... 2. A conditional GNU COBOL statement properly terminated with the correct “END-xxxx” trailer. <p>Any PROCEDURE DIVISION statement can be made to be imperative– and therefore may be used in circumstances that only allow imperative statements - under one or the other definition.</p>
Intrinsic Function	<p>A built-in routine that accepts arguments and returns a value; syntactically, these may be used most places where GNU COBOL identifiers are valid.</p> <p>See section 6.1.7 for documentation on all supported intrinsic functions.</p>
Level number	<p>A user-defined word expressed as a 1- or 2-digit number that indicates the hierarchical position of a data item or the special properties of a data description entry.</p> <p>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 can be written either as a single digit or as a zero followed by the significant digit.</p> <p>Level numbers 66, 77, 78 and 88 identify special properties of a data description entry.</p> <p>See sections 5.3, 5.4, 5.5 and 0.</p>
Literal	A numeric literal or an alphanumeric literal .
Main program	A GNU COBOL program that is to be executed directly from an operating system or shell event. Main programs are not executed from other programs unless such execution is accomplished via the CALL “SYSTEM” facility.
Numeric literal	A numeric constant. See section 1.8 .
Primary Entry-Point	See entry-point .
Procedure	All executable code statements within a single PROCEDURE DIVISION paragraph or SECTION.
Procedure name	A programmer-defined SECTION or paragraph name in the PROCEDURE DIVISION assigned to a procedure . Procedure names serve as a means by which a statement may refer to the statements that follow the procedure name.
Program	A GNU COBOL main program or subprogram. Subprogram programs may be nested inside of other programs and a main program may be followed by any number of subprogram programs in the same compilation group.

Qualification	The process of establishing a unique reference to a data item whose name is duplicated in a program. This takes the form of using the duplicated data name and the name of any of its parent data items, connected by "OF" or "IN" such that the combination of those two data names is unique within the program.
Record	The most-inclusive, highest level, data item. The level number for a record is 01. A record can be either an elementary item or a group item .
Registers	Special data items that are automatically defined for your use by the GNU COBOL compiler. See section 6.1.8 .
Reserved word	A COBOL word specified in the list of words that can be used in a COBOL source program, but that must not appear in the program as user-defined words or system names.
Sentence	Any number of COBOL statements , followed by a period.
Statement	A single COBOL instruction. Every statement starts with a verb which defines the overall action the statement will take. Any additional syntax following the verb refines the actions that will be taken.
Subprogram	A user-defined function or a subroutine .
Subroutine	A program executed from another via a GNU COBOL "CALL" statement (or the equivalent in whatever programming language that other program was written in).
User-defined Function	A user-written GNU COBOL subprogram that may be executed in a syntactically-similar manner to that by which the various built-in intrinsic functions are executed.
User-defined names	Either the name of an identifier or a procedure in the program. GNU COBOL limits user-defined names to a maximum of 31 characters taken from the set of numeric digits, upper- and lower-case letters, hyphens and underscores. A user-defined name may neither begin nor end with a hyphen or underscore. User-defined names used as file names may additionally not begin with a digit although - unlike many other programming languages - user-defined names used as identifiers or procedure names may.
Verb	A single COBOL reserved-word which defines an action a COBOL program will take at execution time. Every COBOL statement begins with a verb. Some verbs perform relatively simple actions (MOVE , STOP , SET , etc.) while others can perform extremely complex actions (SEARCH , SORT , MERGE , STRING , UNSTRING , etc.).

Index

"

"*" In Column 7, 1-17
 ">", 1-17
 "/" In Column 7, 1-17

>

>>D, 1-17

A

ACCEPT, 5-19, 6-30
 Command-Line Arguments, 6-32
 CONSOLE, 6-32
 Date/Time, 6-35
 Environment, 6-33
 Screen Data, 6-33
 Screen Size, 6-35
 ACCESS MODE, 4-12, 4-13
 DYNAMIC, 6-49, 6-78, 6-79, 6-85, 6-96
 RANDOM, 6-49, 6-78, 6-79, 6-85
 SEQUENTIAL, 4-11, 4-12, 6-49, 6-85, 6-96
 ADD
 CORRESPONDING, 6-39
 GIVING, 6-39
 TO, 6-38
 ADDRESS OF
 FREE, 6-61
 SET, 6-91
 AFTER, 6-77
 INSPECT, 6-68, 6-69
 PERFORM VARYING, 6-76
 PERFORM WITH TEST, 6-77
 WRITE ADVANCING, 6-109
 ALL
 INSPECT, 6-68, 6-69
 VALUE, 5-17
 ALL PROCEDURES, 6-30
 ALLOCATE, 5-6, 6-40
 ALPHABET, 4-6
 ALPHABETIC, 6-6
 ALPHABETIC-LOWER, 6-6
 ALPHABETIC-UPPER, 6-6
 Alphanumeric Literal, 1-18
 ALTER, 6-41
 ALTERNATE RECORD KEY, 4-13, 6-96
 ALTERNATE RECORD KEY fields, 6-79
 ANY, 6-57
 ANY LENGTH, 5-6
 ARGUMENT-NUMBER, 6-32
 ARGUMENT-VALUE, 6-32, 6-33
 Arithmetic Expressions, 6-2
 ASCENDING KEY
 SORT, 6-94, 6-95
 Table, 6-88
 ASSIGN, 4-9
 AT
 ACCEPT, 6-34
 DISPLAY, 6-51
 END (READ), 6-78
 END-OF-PAGE, 6-109
 AUTO, 5-19

B

BACKGROUND-COLOR, 5-19, 5-23
 Back-Tab Key, 5-19
 BASED, 5-5, 5-6, 6-91
 BEEP, 5-20
 BEFORE
 INSPECT, 6-68, 6-69
 PERFORM WITH TEST, 6-77
 WRITE ADVANCING, 6-109
 BELL, 5-20
 Big-Endian, 5-16
 BLANK LINE, 5-20
 BLANK SCREEN, 5-20
 BLANK WHEN ZERO, 5-6, 5-20
 BLINK, 5-20
 BLOCK CONTAINS, 5-3
 BY
 CONTENT, 7-11
 PERFORM VARYING, 6-76, 6-77
 REFERENCE, 6-30, 6-43, 7-10, 7-12
 VALUE, 6-30, 7-12
 BY CONTENT, 7-4
 BY REFERENCE, 7-4
 BY VALUE, 7-4
 BYTE-LENGTH, 5-25

C

C\$CALLED BY, 8-11
 C\$CHDIR, 8-11
 C\$COPY, 8-11
 C\$DELETE, 8-11
 C\$FILEINFO, 8-12
 C\$GETPID, 8-12
 C\$JUSTIFY, 8-12
 C\$MAKEDIR, 8-12
 C\$NARG, 8-12, 8-23
 C\$PARAMSIZE, 8-13
 C\$PRINTABLE, 8-13
 C\$SLEEP, 8-13
 C\$TOLOWER, 8-13
 C\$TOUPPER, 8-13
CALL, 6-6, 6-42, 10-1
CALL-CONVENTION, 6-42
 Called Program, 7-1
 Calling Program, 7-1
 CANCEL, 6-44, 7-2
 CBL_AND, 8-13
 CBL_CHANGE_DIR, 8-14
 CBL_CHECK_FILE_EXIST, 8-14
 CBL_CLOSE_FILE, 8-14
 CBL_COPY_FILE, 8-15
 CBL_CREATE_DIR, 8-15
 CBL_CREATE_FILE, 8-15
 CBL_DELETE_DIR, 8-15
 CBL_DELETE_FILE, 8-15
 CBL_EQ, 8-18
 CBL_ERROR_PROC, 8-16
 CBL_EXIT_PROC, 8-17
 CBL_FLUSH_FILE, 8-18
 CBL_GET_CSR_LOCN, 8-18

CBL_GET_SCR_SIZE, 8-19
 CBL_IMP, 8-19
 CBL_NIMP, 8-19
 CBL_NOR, 8-20
 CBL_NOT, 8-20
 CBL_OC_NANOSLEEP, 8-20
 CBL_OPEN_FILE, 8-15, 8-20
 CBL_OR, 8-21
 CBL_READ_FILE, 8-20, 8-21
 CBL_RENAME_FILE, 8-21
 CBL_TOUPPER, 8-22
 CBL_WRITE_FILE, 8-15, 8-20, 8-22
 CBL_XOR, 8-22
 CDF Statements
 >>DEFINE, 2-2
 >>IF, 2-3
 >>SET, 2-4
 >>SOURCE, 2-4
 >>TURN, 2-5
 COPY, 2-1
 REPLACE, 2-2
 CHAIN, 6-29
 CHAINING, 6-29
 CHARACTERS, 6-69
 CLASS, 4-6
 CLASSIFICATION, 4-2
 CLOSE, 6-26, 6-45, 6-75
 COB-CRT-STATUS, 4-4, 6-36
 cobcrun, 8-7
 CODE-SET, 5-3
 COL, 5-20
 Collating Sequence, 10-1
 COLLATING SEQUENCE, 4-2, 4-8
 COLUMN, 5-19, 5-20
 Column 7
 "*", 1-17
 "/", 1-17
 "D", 1-17
 COLUMNS, 6-35
 Combined Conditions, 6-8
 COMMAND-LINE, 6-32
 COMMIT, 6-26, 6-46, 6-85
 COMMON, 3-1
 Compilation Group, 10-1
 Compiler Switches
 All Switches, 8-1
 -b, 8-3
 -conf, 8-5
 -fdebugging-line, 1-17, 4-2
 -ffold-copy, 2-4
 -ffunctions-all, 4-3
 -fixed, 2-4
 -fnotrunc, 8-26
 -foptional-file, 4-8
 -free, 1-14, 1-15, 2-4
 -fsyntax-extension, 4-6
 -g, 6-14
 -m, 8-3, 8-7, 10-1
 -o, 8-3
 -S, 8-3
 -Wobsolete, 3-1
 -x, 8-3, 8-7, 10-2
 COMPUTE, 6-47
 Condition Names, 6-5
 Conditional Expressions, 6-2, 6-5
 Conditions

Combined, 6-8
 Level-88 Condition Names, 6-5
 Negated, 6-8
 Relation, 6-7
 Switch Status, 6-7
 Configuration Files, 8-5
 CONFIGURATION SECTION, 4-1
 CONSOLE, 6-50
 CONSOLE IS CRT, 4-4
 CONSTANT, 2-3, 5-25
 Constant Descriptions, 5-25
 CONTINUE, 6-48
 CONVERSION, 6-34, 6-51
 CONVERTING, 6-68, 6-69, 6-104
 CORRESPONDING, 6-39
 COUNT, 6-107
 CRT, 6-50
 CRT STATUS, 4-4, 6-36
 CURRENCY SIGN, 4-4
 CURSOR IS, 4-5

D

D In Column 7, 1-17
 DATA DIVISION, 1-9, 1-12
 DATA RECORD, 5-3
 DATE, 6-35
 DATE YYYYMMDD, 6-35
 DAY, 6-35
 DAY YYYYDDD, 6-35
 DAY-OF-WEEK, 6-35
 DEBUGGING MODE, 4-2
 DECIMAL POINT IS COMMA, 4-4
 DECLARATIVES, 6-30, 6-49, 6-75, 10-1
 DEFAULT, 6-66
 DEFINED, 2-3
 DELETE, 6-49, 6-75
 DELIMITED BY
 STRING, 6-99
 UNSTRING, 6-106
 DELIMITED BY SIZE, 6-99
 DELIMITER, 6-107
 DESCENDING KEY
 SORT, 6-94, 6-95
 Table, 6-88
 DISC, 4-9
 DISK, 4-9, 5-3
 DISPLAY, 5-19
 Command-Line Arguments, 6-50
 CONSOLE, 6-50
 Environment, 6-50
 Screen Data, 6-51
 DISPLAY (ASSIGN), 4-9
 DIVIDE
 BY/GIVING, 6-54
 BY/REMAINDER, 6-55
 INTO, 6-53
 INTO/GIVING, 6-53
 INTO/REMAINDER, 6-54
 DIVISION, 10-1
 DYNAMIC, 4-12, 4-13
 Dynamically-Loadable Library, 10-1

E

Elementary Item, 10-1

ELSE, 6-65
 EMPTY-CHECK, 5-23
 END-IF, 6-65
 ENTRY, 6-56, 6-90, 10-1
 Entry Point, 10-1
 ENVIRONMENT, 6-33
 ENVIRONMENT DIVISION, 1-9, 4-1
 Environment Variables
 COB_CC, 8-4
 COB_CFLAGS, 8-4
 COB_CONFIG_DIR, 8-4
 COB_CONFIG_PATH, 8-5
 COB_COPY_DIR, 8-4, 8-5
 COB_DISPLAY_WARNINGS, 8-8
 COB_LDADD, 8-4
 COB_LDFLAGS, 8-4
 COB_LIBRARY_PATH, 8-8
 COB_LIBS, 8-4
 COB_PRE_LOAD, 8-8
 COB_SCREEN_ESC, 6-34, 8-9
 COB_SCREEN_EXCEPTIONS, 6-34, 8-9
 COB_SET_DEBUG, 8-8
 COB_SET_TRACE, 8-9
 COB_SORT_MEMORY, 8-9
 COB_SWITCH_n, 8-9
 COB_SYNC, 8-9
 COBCPY, 8-4
 dd_literal-1, 4-9
 DD_literal-1, 4-9
 LD_LIBRARY_PATH, 8-4
 literal-1, 4-9
 PATH, 8-10
 TEMP, 8-10
 TMP, 8-5, 8-10
 TMPDIR, 8-5, 8-10
 ENVIRONMENT-NAME, 6-33
 ENVIRONMENT-VALUE, 6-33
 ERASE EOL, 5-21
 ERASE EOS, 5-21
 Error Procedure (user-defined), 6-75, 8-16
 ESCAPE KEY, 6-36
 EVALUATE, 6-57
 EVENT STATUS, 4-5
 EXCEPTION
 ACCEPT, 6-37
 CALL, 6-42
 DISPLAY, 6-52
 Executable File, 10-2
 EXIT, 6-59
 PARAGRAPH, 6-59, 6-71
 PERFORM, 6-59
 PERFORM CYCLE, 6-59
 PROGRAM, 6-71, 6-93, 6-95
 SECTION, 6-59, 6-71
 Simple, 6-59
 Exit Procedure (user-defined), 8-17
 Expressions
 Arithmetic, 6-2
 Conditional, 6-2, 6-5
 EXTEND, 6-75, 6-108
 EXTERNAL, 4-8, 5-5
 FD, 5-2, 5-5

F

FD, 6-108, 6-109

11FEB2012 Version

-fdebugging-line, 1-17, 4-2
 -ffold-copy, 2-4
 -ffunctions-all, 4-3
 Figurative Constant, 1-19, 10-2
 File Description, 6-108, 6-109
 FILE SECTION, 1-9
 FILE STATUS, 4-9
 FILE-CONTROL, 1-9, 4-8
 FILLER, 5-5
 FIRST
 INSPECT, 6-68, 6-69
 -fixed, 2-4
 Fixed Format Mode, 1-14
 FOLDCOPYNAME, 2-4
 -foptional-file, 4-8
 FOREGROUND-COLOR, 5-21, 5-23
 FOREVER, 6-76, 6-77
 -free, 1-14, 1-15, 2-4
 FREE, 6-61
 Free Format Mode, 1-15
 FROM
 PERFORM VARYING, 6-76
 REWRITE, 6-85
 Screen Item Description, 5-21
 WRITE, 6-108
 -fsyntax-extention, 4-6
 FULL, 5-21
 Function
 User-Defined, 7-1
 FUNCTION-ID, 3-1

G

GENERATE, 6-62
 GIVING
 CALL, 6-42
 MERGE, 6-71
 SORT, 6-94
 STOP, 6-98
 GLOBAL, 6-30
 FD, 5-2, 5-5
 GO TO, 6-71, 6-76, 6-93, 6-95
 DEPENDING ON, 6-64
 Simple, 6-64
GOBACK, 6-60, 6-63, 6-71, 6-93, 6-95, 8-17
 Group Item, 10-2

H

HIGHLIGHT, 5-22

I

IDENTIFICATION DIVISION, 3-1
 Identifier, 10-2
 IF, 6-65
 IGNORING LOCK, 6-26
 Imperative Statement, 10-2
 INDEXED BY, 6-88, 6-91
 INITIAL, 3-1, 6-42
 INITIALIZE, 6-40
 Verb, 6-66
 INITIATE, 6-67
 INPUT, 6-75
 INPUT PROCEDURE, 6-82, 6-93

INPUT-OUTPUT SECTION, 1-9, 4-7
 INSPECT, 6-68, 6-104
 Intrinsic Function, 10-2
 Intrinsic Functions (Supported)
 ABS, 6-11
 ACOS, 6-11
 ANNUITY, 6-11
 ASIN, 6-11
 ATAN, 6-11
 BYTE-LENGTH, 6-11
 CHAR, 6-11
 COMBINED-DATETIME, 6-12
 CONCATENATE, 6-12
 COS, 6-12
 CURRENCY-SYMBOL, 6-12
 CURRENT-DATE, 6-12
 DATE-OF-INTEGGER, 6-13
 DATE-TO-YYYYMMDD, 6-13
 DAY-OF-INTEGGER, 6-13
 DAY-TO-YYYYDDD, 6-13
 E, 6-13
 EXCEPTION-FILE, 6-13
 EXCEPTION-LOCATION, 6-14
 EXCEPTION-STATEMENT, 6-14
 EXCEPTION-STATUS, 6-14
 EXP, 6-14
 EXP10, 6-14
 FACTORIAL, 6-14
 FRACTIONAL-PART, 6-14
 HIGHEST-ALGEBRAIC, 6-15
 INTEGER, 6-15
 INTEGER-OF-DATE, 6-15
 INTEGER-OF-DAY, 6-15
 INTEGER-PART, 6-15
 LENGTH, 6-15
 LENGTH-AN, 6-15
 LOCALE-COMPARE, 6-15
 LOCALE-DATE, 6-16
 LOCALE-TIME, 6-16
 LOCALE-TIME-FROM-SECS, 6-16
 LOG, 6-16
 LOG10, 6-16
 LOWER-CASE, 6-16
 LOWEST-ALGEBRAIC, 6-17
 MAX, 6-17, 6-22
 MIDRANGE, 6-17
 MOD, 6-17
 MODULE-CALLER-ID, 6-17
 MODULE-DATE, 6-17
 MODULE-FORMATTED-DATE, 6-17
 MODULE-ID, 6-18
 MODULE-PATH, 6-18
 MODULE-SOURCE, 6-18
 MODULE-TIME, 6-18
 MONETARY-DECIMAL-POINT, 6-19
 MONETARY-THOUSANDS-SEPARATOR, 6-19
 NUMERIC-DECIMAL-POINT, 6-19
 NUMERIC-THOUSANDS-SEPARATOR, 6-19
 NUMVAL, 1-18, 6-19
 NUMVAL-C, 1-18, 6-20
 NUMVAL-F, 6-20
 ORD, 6-20
 ORD-MAX, 6-20
 ORD-MIN, 6-21
 PI, 6-21
 PRESENT-VALUE, 6-21

RANDOM, 6-21
 RANGE, 6-21
 REM, 6-21
 REVERSE, 6-21
 SECONDS-FROM-FORMATTED-TIME, 6-22
 SECONDS-PAST-MIDNIGHT, 6-22
 SIGN, 6-22
 SIN, 6-22
 SQRT, 6-22
 STORED-CHAR-LENGTH, 6-22
 SUBSTITUTE, 6-22, 6-23
 SUM, 6-23
 TAN, 6-23
 TEST-DATE-YYYYMMDD, 6-23
 TEST-DAY-YYYYDDD, 6-23
 TEST-NUMVAL, 6-23
 TEST-NUMVAL-C, 6-23
 TEST-NUMVAL-F, 6-23
 TRIM, 6-23
 UPPER-CASE, 6-24
 VARIANCE, 6-24
 WHEN-COMPILED, 6-24
 YEAR-TO-YYYY, 6-24
 Intrinsic Functions (Unsupported)
 BOOLEAN-OF-INTEGGER, 6-10
 CHAR-NATIONAL, 6-10
 DISPLAY-OF, 6-11
 EXCEPTION-FILE-N, 6-10
 EXCEPTION-LOCATION-N, 6-10
 INTEGER-OF-BOOLEAN, 6-11
 NATIONAL-OF, 6-10
 STANDARD-COMPARE, 6-10
 INVALID KEY
 DELETE, 6-49
 REWRITE, 6-85
 START, 6-96
 WRITE, 6-108
 I-O, 6-75, 6-108
 I-O-CONTROL, 4-14

J

JUSTIFIED RIGHT, 5-6, 5-22

K

KEY (START), 6-96
 KEYBOARD (ASSIGN), 4-9

L

LABEL RECORD, 5-3
 LEADING
 INSPECT, 6-68, 6-69
 SIGN, 5-13, 5-24
 LEFTLINE, 5-22
 LENGTH, 5-25
 LENGTH OF, 6-38
 Use With Alphanumeric Literals, 1-20
 Level
 66, 5-19
 78, 5-19, 5-25
 88, 5-19
 Level Number, 10-2
 LINAGE, 5-1, 5-3, 6-24, 6-109

LINE, 5-19, 5-22
 LINE ADVANCING, 1-7, 6-108
 LINES
 ACCEPT, 6-35
 AT BOTTOM, 6-109
 AT TOP, 6-109
 WRITE ADVANCING, 6-109
 LINKAGE SECTION, 6-29, 6-91
 Literal, 10-2
 Little-Endian, 5-16
 LOCALE, 4-3, 4-4
 LOCAL-STORAGE SECTION, 6-43
 LOCK, 4-10
 LOWLIGHT, 5-22

M

Main Program, 10-2
 MEMORY SIZE, 4-2
 MERGE, 6-70
MOVE
 CORRESPONDING, 6-72
 Simple, 6-72
 MULTIPLE FILE TAPE, 4-14
 MULTIPLY
 BY, 6-73
 GIVING, 6-73

N

NEAREST-AWAY-FROM-ZERO, 6-27
 Negated Conditions, 6-8
 NEGATIVE, 6-6
 Nested Subprograms, 3-1
 NEXT, 6-78
 NEXT SENTENCE, 6-74
 NO ADVANCING, 6-50
 NO REWIND, 6-45
 NO-ECHO, 5-23
 NOFOLDCOPYNAME, 2-4
 NOT AT END, 6-78
 NOT AT END-OF-PAGE, 6-109
 NOT EXCEPTION
 ACCEPT, 6-37
 DISPLAY, 6-52
 NOT INVALID KEY
 DELETE, 6-49
 READ, 6-80
 READ, 6-79
 REWRITE, 6-85
 START, 6-97
 WRITE, 6-108
 NOT ON OVERFLOW
 STRING, 6-99
 UNSTRING, 6-107
 NOT ON SIZE ERROR
 ADD, 6-38
 COMPUTE, 6-47
 DIVIDE, 6-53, 6-54, 6-55
 MULTIPLY, 6-73
 SUBTRACT, 6-100
 NUMBER-OF-CALL-PARAMETERS, 8-13, 8-23
 NUMERIC, 6-6
 Numeric Literal, 1-18, 10-2

O

OBJECT-COMPUTER, 4-2
 OCCURS, 5-6, 6-88
 OMITTED, 6-6
 ON OVERFLOW
 STRING, 6-99
 UNSTRING, 6-107
 ON SIZE ERROR
 ADD, 6-38
 COMPUTE, 6-47
 DIVIDE, 6-53, 6-54, 6-55
 MULTIPLY, 6-73
 SUBTRACT, 6-100
 OPEN, 6-75, 6-78, 6-79, 6-96, 6-108
 OPTIONAL, 4-8
 ORGANIZATION
 INDEXED, 1-8, 4-13, 6-49, 6-78, 6-85, 6-96, 6-108, 8-9
 LINE SEQUENTIAL, 1-6, 4-11, 5-3, 6-45, 6-70, 6-85, 6-93, 6-108
 RECORD BINARY SEQUENTIAL, 1-7, 4-11, 5-3, 6-45, 6-70, 6-85, 6-93, 6-108
 RELATIVE, 1-8, 4-12, 6-49, 6-85, 6-96, 6-108
 OUTPUT, 6-75, 6-108
 OUTPUT PROCEDURE, 6-84
 MERGE, 6-71
 SORT, 6-94
 OVERFLOW
 CALL, 6-42
 OVERLINE, 5-22
 OVERRIDE, 2-3

P

PAGE
 WRITE ADVANCING, 6-109
 PARAMETER
 CDF, 2-3
 PERFORM, 6-31, 6-59
 Inline, 6-77
 Procedural, 6-76
 POSITIVE, 6-6
 PREVIOUS, 6-78
 Primary Entry-Point Name, 3-1
 PRIMARY RECORD KEY, 6-79
 PRINTER, 4-9, 4-12, 6-50
 Procedure, 10-2
 PROCEDURE DIVISION, 6-29
 Procedure Name, 10-2
 Program, 10-2
 Called, 7-1
 Calling, 7-1
 PROGRAM COLLATING SEQUENCE, 6-94, 6-95
 PROGRAM-ID, 3-1
 PROGRAM-POINTER, 6-90
 PROMPT, 5-23

Q

Qualification, 6-1, 10-2

R

RANDOM, 4-9, 4-12, 4-13
 READ, 6-49, 6-75, 6-78, 6-79, 6-85

READY TRACE, 6-81
 Record, 10-3
 RECORD CONTAINS, 5-3, 6-85
 RECORD DELIMITER, 4-8
 RECORD IS VARYING, 5-3, 6-85
 RECORD KEY, 4-13, 6-49, 6-85, 6-96
 RECORDING MODE, 5-3
 REDEFINES, 5-5, 5-12
 REEL, 6-45
 Reference Modifier, 6-2
 Registers, 10-3
 Relation Conditions, 6-7
 RELATIVE KEY, 4-12, 6-49, 6-85, 6-96
 RELEASE, 6-82, 6-93
 REPLACING
 INITIALIZE, 6-66
 REPLACING (COPY), 2-1
 REPLACING (INSPECT), 6-68, 6-69
 REPORT IS, 5-3
 REPORT SECTION, 5-1
 REPOSITORY, 4-3, 6-10
 REQUIRED, 5-23
 RESERVE, 4-8
 Reserved Word, 10-3
 RESET TRACE, 6-83
 RETURN, 6-71, 6-84, 6-94
RETURN-CODE, 6-98, 8-11, 8-12, 8-14, 8-15, 8-16, 8-17, 8-18, 8-19, 8-20, 8-21, 8-23
 RETURNING, 6-29, 6-30, 6-40
 CALL, 6-42
 STOP, 6-98
 REVERSED, 6-75
 REVERSE-VIDEO, 5-23
 REWRITE, 6-85
 ROLLBACK, 6-26, 6-86
 ROUNDED, 6-27
 DIVIDE, 6-55

S

SAME RECORD AREA, 4-14, 6-70
 SAME SORT AREA, 4-14, 6-70
 SAME SORT-MERGE AREA, 4-14, 6-70
 SCREEN CONTROL, 4-5
 SCREEN SECTION, 1-12
 SCROLL, 6-34
 SEARCH
 (Sequential), 6-87
 ALL (Binary Search), 6-88
 SECURE, 5-23
 SEGMENT-LIMIT, 4-2
 SELECT, 1-9
 Sentence, 10-3
 SEPARATE CHARACTER, 5-13, 5-24
 SEQUENTIAL, 4-12, 4-13
 SET
 >>IF Option, 2-3
 Address, 6-90
 Condition Name, 6-91
 ENVIRONMENT, 6-90
 Index, 6-91
 Program-Pointer, 6-90
 Switch, 6-92
 UP/DOWN, 6-91
 SET ADDRESS, 5-6
 SET ATTRIBUTE, 6-92

11FEB2012 Version

SHARING, 4-10, 6-75
 SHARING WITH ALL OTHER, 4-8
 Shift-Tab Key, 5-19
 SIGN, 5-13, 5-24
 SIZE, 6-30
 SIZE IS AUTO, 6-30
 SORT
 File, 6-93
 Table, 6-95
 SORT STATUS, 4-9
 SOURCE-COMPUTER, 4-2
 Special Registers, 10-3
 SPECIAL-NAMES, 4-3, 4-4, 6-50, 8-24
 Split Keys, 4-13
 START, 6-75, 6-96
 Statement, 10-3
STOP RUN, 6-63, 6-71, 6-76, 6-93, 6-95, 6-98, 8-17
 STRING, 6-99
 Subprogram, 10-3
 Subroutine, 10-3
 Subscripts, 6-1
 SUBTRACT
 CORRESPONDING, 6-101
 FROM, 6-100
 GIVING, 6-100
 SUPPRESS, 6-102
 Switch Status Conditions, 6-7
 SWITCH-n, 4-6, 8-23, 8-24
 SYMBOLIC CHARACTERS, 4-7
 SYNCHRONIZED, 5-13
 SYSTEM, 8-23
 SYSTEM-DEFAULT, 4-3

T

Tab Key, 5-19
 TALLYING, 6-68
 UNSTRING, 6-107
 TAPE, 4-9
 TERMINATE, 6-103
 THROUGH, 6-57, 6-76
 THRU, 6-57, 6-76
 TIME, 6-35
 TIMEOUT, 6-34
 TIME-OUT, 6-34
 TIMES, 6-59, 6-76, 6-77
 TO
 Screen Item Description, 5-21
 TO VALUE, 6-66
 TRACE
 COB_SET_TRACE Environment Variable, 8-9
 READY, 6-81
 RESET, 6-83
 TRAILING
 INSPECT, 6-68, 6-69
 SIGN, 5-13, 5-24
 TRANSFORM, 6-104
 TRUNCATION, 6-27

U

UNDERLINE, 5-22
 UNIT, 6-45
 UNLOCK, 6-26, 6-46, 6-75, 6-105
 UNSTRING, 6-106
 UNTIL, 6-59, 6-76, 6-77

UNTIL EXIT, 6-76
UPDATE, 6-34
UPON, 6-50
USAGE, 5-14
 BINARY-CHAR, 7-9, 8-23
 BINARY-CHAR SIGNED, 7-9
 BINARY-CHAR UNSIGNED, 7-9
 BINARY-C-LONG SIGNED, 7-9
 BINARY-DOUBLE, 7-9
 BINARY-DOUBLE SIGNED, 7-9
 BINARY-DOUBLE UNSIGNED, 7-9
 BINARY-INT, 7-9
 BINARY-LONG, 7-9
 BINARY-LONG SIGNED, 7-9
 BINARY-LONG UNSIGNED, 7-9
 BINARY-LONG-LONG, 7-9
 BINARY-SHORT, 7-9
 BINARY-SHORT SIGNED, 7-9
 BINARY-SHORT UNSIGNED, 7-9
 COMPUTATIONAL-1, 7-9
 COMPUTATIONAL-2, 7-10
 DISPLAY, 6-5
 INDEX, 6-91
 POINTER, 6-5, 6-40, 6-91
 PROGRAM POINTER, 6-5
 PROGRAM-POINTER, 6-91, 8-16, 8-17
USE AFTER STANDARD ERROR PROCEDURE, 6-30
USE BEFORE REPORTING, 6-30
USE FOR DEBUGGING, 6-30
 COB_SET_DEBUG Environment Variable, 8-8
USER-DEFAULT, 4-3
User-Defined Function, 7-1, 10-3
user-defined name, 10-3
User-Defined Name, 10-3
USING, 6-29
 Screen Item Description, 5-21
USING (CALL), 6-43
USING (SORT), 6-93

V

VALUE, 5-17, 5-24, 5-26
VALUE OF, 5-3
VARYING, 6-59, 6-76, 6-77
Verb, 10-3

W

-Wall, 6-75
WHEN, 6-57
WITH
 DUPLICATES IN ORDER, 6-70, 6-93, 6-95
 IGNORE LOCK, 6-26
 LOCK, 6-26
 CLOSE, 6-45
 OPEN, 6-75
 NO LOCK, 6-26
 NO REWIND
 OPEN, 6-75
 POINTER
 STRING, 6-99
 UNSTRING, 6-106
 TEST, 6-77
WITH FILLER, 6-66
WITH WAIT, 6-27
-Wobsolete, 6-75
WRITE, 6-108

X

X"E4", 8-24
X"E5", 8-24
X"F4", 8-24
X"F5", 8-25
X"91", 8-23

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially.

Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or

whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below,

refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical

connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero

Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with

generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input

to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent.

An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats

include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition.

Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material.

If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice.

These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number.

Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit.

When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form.

Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4.

Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant

Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number.

If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.