



COBOL-IT® Compiler Suite Enterprise Edition
Compiler & Runtime Reference Manual
Version 4.1



Acknowledgment

This documentation is derived from COBOL-IT Source code, parts of which are derived from OpenCOBOL.

Copyright (C) 2002-2007 Keisuke Nishida

Copyright (C) 2007 Roger While

Copyright (C) 2008-2020 COBOL-IT

In 2008, COBOL-IT forked its own compiler branch, with the intention of developing a fully featured product and offering professional support to the COBOL user industry.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Free Software Foundation.

Conventions used in the General Format diagrams:

Brackets [] identify syntax elements that are supported but not required.

Curly Braces { } identify alternative syntax elements. Among syntax elements described within stacked curly braces, only one of the entries may be selected.

Ellipses (...) indicate (optional) repetition. If the syntax element is a required element, then it will be surrounded by curly braces.

Copyright 2008-2020 COBOL-IT S.A.R.L. All rights reserved. Reproduction of this document in whole or in part, for any purpose, without COBOL-IT's express written consent is forbidden.

COBOL-IT® Developer Studio, COBOL-IT® Sort (CitSORT®), COBOL-IT® MF Command Line Emulator (CitEMUL®), COBOL-IT® Lib Optimizer are registered trademarks of COBOL-IT, S.A.R.L All rights reserved.

The CitSQL® family: COBOL-IT® Precompiler for MySQL, COBOL-IT® Precompiler for PostgreSQL. COBOL-IT® Precompiler for Microsoft SQL Server, are registered trademarks of COBOL-IT. All rights reserved.

COBOL-IT® Precompiler for MySQL, COBOL-IT® Precompiler for PostgreSQL. COBOL-IT® Precompiler for Microsoft SQL Server are licensed by COBOL-IT under exclusive license with the Raincode Company.

Third-Party software components embedded in the SOFTWARE and Services and submitted to specific licenses:

VBISAM

- * Copyright (C) 2003 Trevor van Bremen
- * Copyright (C) 2008-2020 COBOL-IT
- * License: LGPL

GMP (GNU Multiprecision Library)

- * Copyright 1991, 1996, 1999, 2000, 2007 Free Software Foundation, Inc.
- * License: LGPL

GNU LIBICONV

The libiconv libraries and their header files are under LGPL.

Microsoft and Windows are registered trademarks of the Microsoft Corporation. UNIX is a registered trademark of the Open Group in the United States and other countries. Other brand and product names are trademarks or registered trademarks of the holders of those trademarks.

Contact Information:

The Lawn
22-30 Old Bath Road
Newbury, Berkshire, RG14 1QN
United Kingdom
Tel: +44-0-1635-565-200

ACKNOWLEDGMENT2
COMPILER & RUNTIME REFERENCE19
The COBOL Compiler: cobc19

Informational Flags	19
-check-codepage <codepage-id>	19
-help	20
-list-codepage	20
-list-intrinsics	20
-list-mnemonics	20
-list-reserved	20
-version, -V	20
Standard Flags.....	20
-b	20
-c	21
-codepage <codepage-id>	21
-conf=[+<file>]	21
-constant "key=value"	21
-debug.....	21
-debugdb [=<DebugDB-name>].....	22
-dump-config.....	22
-err <file>	22
-ext<extension>.....	22
-fixed	23
-free	23
-g	23
-initcall=<program-name>	23
-l <lib>	23
-linkage-desc=[program.xsd]	23
-m	23
-makesyn "oldvalue=newvalue"	23
-o <file> <dir>.....	24
-preprocess=<CMD> [input file]	24
-save-temps (= <dir>).....	24
-source-codepage <codepage-id>.....	24
-std=<dialect>	25
-sysin=<input file>.....	25
-sysout=<output file> [,S/L [,Min [,Max]]].....	25
-t <file> <dir>.....	26
-use-extfh <NAME>	27
-use-extsm <NAME>	27
-v	27
-x	27
-xdd-prefix=<dir>	27
-C.....	28
-D <define>	28
-E.....	28
-G	28
-I <path>[,ext1,ext2,..,extn][@<LibName>] <command-file>	29
-L <directory>	29
-MF <file>.....	29
-MT <target>.....	29
-O, -Os, -O2	29
-R <directory>.....	30
-S.....	30
-Wc CC_opt	30

-W1 LD_opt.....	30
Guidelines for Searching and Locating COPY files	31
Guidelines for enforcing bounds-checking	34
Guidelines for optimizing performance	35
-O compiler flags.....	35
Optimizations enabled with the -O compiler flag.....	35
COB_OPTSIZE_FLAG	36
The CALL statement	37
The PERFORM statement.....	37
Resolving File Names	37
Removing debug-oriented compiler flags	38
Optimizing compiler flags set by default	39
Guidelines for use of -preprocess=cmd.....	39
Compiler -f Flags	41
-f77-opt.....	41
-faccept-with-auto	41
-faccept-with-update	42
-falign-8.....	42
-fall-external-call.....	42
-fall-external-link	42
-faloc-unused-linkage.....	42
-fas400-like	42
-fauto-load-symb	43
-fautolock	43
-fauto-sprwr.....	43
-fbdb	43
-fbinary-byteorder-big-endian.....	43
-fbinary-byte-order-native.....	43
-fbin-opt	43
-fbin-opt-strict	43
-fcall-comp5-as-comp	44
-fcall-lowercase	44
-fcall-opt.....	44
-fcall-uppercase	44
-fcarealia-sign.....	44
-fC-cmd-line.....	45
-fC-data-init.....	45
-fcheckpoint	45
-fcics.....	45
-fcmp-inline.....	45
-fcmp-opt.....	45
-fcobol-lines	45
-fcompat-display-to-int	46
-fcompute-ibm.....	46
-fcompute-ibm-trunc	46
-fcontinuation-line.....	46
-fcopy-default-leading	46
-fcopy-exec-replace.....	46
-fcopy-mark.....	47
-fcopy-partial-replace	47
-fctree	47
-fctree-field-numbering	47
-fctree-no-full-qualification.....	47
-fcudir-include	47
-fdebugdb	48
-fdebug-exec.....	48
-fdebugging-line.....	48
-fdebug-parser	48

-fdecimal-optimize	48
-fdisam	48
-fdisplay-dos.....	48
-fdisplay-ibm.....	49
-fdiv-check	49
-febcdic-charset.....	49
-femulate-vms	49
-fexclusivelock.....	49
-fexec-check	49
-fexpand-exec-copy.....	49
-fexpand-sql-include	50
-ffast-figurative-move	50
-ffast-op.....	50
-ffcdreg.....	50
-ffdclear.....	51
-ffile-auto-external	51
-ffold-copy-lower	52
-ffold-copy-upper	52
-ffp-opt	52
-ffree-thread-safe-data.....	52
-ffunctions-all.....	52
-ffunctions-all-intrinsic	52
-fgcc	52
-fgcc-bug	52
-fgcc-goto.....	52
-fgcc-O-bug.....	53
-fgcos-mode	53
-fggen-xdd.....	53
-fglobal-typedef.....	53
-fibm-listing-macro	53
-fibm-mainframe	53
-fibm-sync	53
-fimplicit-init	53
-finclude-main.....	53
-fincomplete-subscript.....	54
-findex-optimize.....	54
-finitialize-fd	55
-finitialize-opt.....	55
-fkeep-copy-statement.....	55
-fkeep-org-src-line.....	55
-fkeep-unused.....	55
-fline-seq-dos	55
-flink-only	55
-flisting-sources.....	55
-floosy-comment	55
-fls-expand-tab	56
-fmain	56
-fmain-as-object	56
-fmainframe-vb	56
-fmakesyn-patch-preprocess.....	56
-fmanuallock	56
-fmem-info	56
-fmfcomment.....	56
-mf-compat-parser	56
-mf-ctrl-escaped-parser.....	56
-mf-file-optional.....	57
-mf-gnt	57
-mf-hostnumcompare.....	57

-fmf-int.....	57
-fmf-relativefile.....	57
-fmodule-name-entry.....	58
-fmodule-uppercase.....	58
-fmove-all-edited.....	58
-fno-cbl-error-proc.....	58
-fno-realpath.....	58
-fnostrip.....	58
-fnotrunc.....	58
-fnull-param.....	58
-fnumeric-compare.....	58
-fnumval-validate.....	58
-fobj-cit.....	59
-fodo-slide.....	59
-foptimize-move.....	59
-foptimize-move-call.....	59
-foptional-file.....	59
-fperform-osvs.....	59
-fprepro_cut_line.....	60
-fprinter-crlf.....	61
-fprofiling.....	61
-fprotect-linkage.....	61
-fraw-by-value.....	61
-fraw-pic9-display (Internal use only).....	61
-fread-into-copy.....	61
-fready-trace.....	61
-frecmode-f.....	61
-frecmode-v.....	61
-frecord-depending-iso.....	62
-fregion0.....	62
-frelativefile-bigendian.....	62
-freplace-additive.....	62
-freturn-opt.....	62
-fround-fp.....	62
-frw-after-preprocess.....	62
-frw-mode-nopf.....	62
-frw-mode-nopf-dos.....	63
-fsafe-linkage.....	63
-fsequential-line.....	63
-fshare-all-autolock.....	63
-fshare-all-default.....	63
-fshare-all-manulock.....	64
-fsign-ascii.....	64
-fsign-ebcdic.....	64
-fsign-leading.....	65
-fsign-separate.....	65
-fsimple-trace.....	65
-fsource-location.....	65
-fsplit-debug-mark.....	65
-fstack-check.....	65
-fstatic-call.....	65
-fstrict-compare-low.....	65
-fstrict-record-contains.....	66
-fsyntax-only.....	66
-fthread-safe.....	66
-ftrace.....	66
-ftrace-ts.....	66
-ftrace-upon-sysout.....	66

-ftraceall	67
-ftrap-unhandled-exception	67
-ftruncate-listing	67
-funstring-use-move	68
-futf-8	68
-futf16-le	68
-fvalidate-dep-on	68
-fvalidate-only	69
-fvalue-of-id-priority	69
-fvalue-size-is-auto	69
-fvbisam	69
-fvms-error-handler	69
-fxparse-event	69
Guidelines for handling Linkage Section parameters	69
Guidelines for modifying default handling of the LOCK MODE	70
Guidelines for use of Checkpoints	70
Guidelines for use of Profiler	71
Dumping Profiling Data at the Module Level	71
Using the PRAGMA statement to produce Profiling Reports	72
Info profiling debugging command	72
Attaching a program compiled with -fprofiling to a running process	73
Guidelines for thread-safe programs	73
Compiler -w Flags	74
-w	75
-Wall	75
-Warchaic	75
-Wbdb	75
-Wcall-params	75
-Wconstant	75
-Wimplicit-define	75
-Winformation	75
-Wlinkage	75
-Wobsolete	75
-Wparentheses	75
-Wredefinition	75
-Wstrict-typing	75
-Wsuggestion	75
-Wterminator	75
-Wtruncate	76
-Wunreachable	76
Compiler Configuration File	77
77-opt:[yes/no]	78
accept-but-ignore-comp6-signed:[yes/no]	78
accept-with-auto:[yes/no]	78
accept-with-update:[yes/no]	78
align-8:[yes/no]	78
all-external-call:[yes/no]	79
all-external-link:[yes/no]	79
alloc-unused-linkage:[yes/no]	79
as400-like :[yes/no]	79
assign-clause: [COBOL2002 / mf / ibm / external]	79
auto-initialize:[yes/no]	80
auto-load-symb:[yes/no]	80
autolock:[yes/no]	80
bdb:[yes/no]	80
bin-opt:[yes/no]	80
bin-opt-strict:[yes/no]	80

binary-byteorder: native / big-endian	81
binary-size: 2-4-8 / 1-2-4-8 / 1--8.....	81
binary-truncate:[yes/no]	81
bitfield-first-is-lsb: [yes/no]	82
call-comp5-as-comp:[yes/no].....	82
call-lowercase: [yes/no].....	82
call-opt: [yes/no]	82
call-uppercase: [yes/no].....	82
carealia-sign: [yes/no]	83
C-cmd-line:[yes/no]	83
C-data-init:[yes/no] [Internal use only].....	83
check-linkage-bound: [yes/no]	83
checkpoint: [yes/no]	83
cics: [yes/no]	83
cmp-inline:[yes/no]	83
cmp-opt:[yes/no]	84
cobol-lines: [yes/no].....	84
codepage: <codepage-id>.....	84
comp5-byteorder: [native/big-endian].....	84
compat-display-to-int:[yes/no]	84
complex-odo:[yes/no]	84
compute-ibm:[yes/no]	84
compute-ibm-trunc: [yes/no].....	85
console-is-sysfile: [yes/no].....	85
constant: "key=value"	85
continuation-line	85
copy-default-leading:[yes/no]	85
copy-exec-replace[yes/no]	86
copy-partial-replace:[yes/no]	86
crtstatus-map:[cit-value] [user-value]	86
ctree: [yes/no].....	87
ctree-field-numbering: [yes/no].....	87
ctree-no-full-qualification: [yes/no]	87
curdir-include: [yes/no]	88
datacompress: <integer>	88
debug-exec: [yes/no]	88
debugging-line:[yes/no]	89
debug-parser: [yes/no].....	89
decimal-optimize:[yes/no].....	89
defaultbyte:[any integer]	89
defaultcall:[any integer]	89
disam:[yes/no].....	90
display-dos:[yes/no]	90
display-ibm:[yes/no]	90
displaynumeric-edited-mf50: [yes/no]	90
displaynumeric-mf50:[yes/no]	90
div-check: [yes/no].....	91
ebcdic-charset: [yes/no].....	91
emulate-vms [yes/no]	91
exception checking.....	92
exclusivelock: [yes/no].....	95
exec-check: [yes/no].....	95
exit-program-forced:[yes/no]	95
expand-exec-copy:[yes/no]	95
expand-sql-include:[yes/no]	96
external-link: <function name>	96
external-mapping:[yes/no]	96
fast-figurative-move: [yes/no].....	96

fcldreg: [yes/no]	96
fdclear:[yes/no]	97
file-auto-external:[yes/no]	97
filename-mapping:[yes/no]	97
first-tab-width:[any integer]	97
flat-extfh: <DRIVER NAME>	97
flat-extfh-lib: <library to use for this extfh driver>	97
fold-copy-lower: [yes/no]	97
fold-copy-upper:[yes/no]	98
fp-opt:[yes/no]	98
free-thread-safe-data: [yes/no]	98
fstatus-map:[cit-status] = [custom-status]	98
full-cancel:[yes/no]	98
functions-all:[yes/no]	99
functions-all-intrinsic:[yes/no]	99
gcc:[yes/no]	99
gcc-O-bug:[yes/no]	99
gcc-bug:[yes/no]	99
gcc-goto:[yes/no]	99
gcos-mode:[yes/no]	100
gen-xdd:[yes/no]	100
global-typedef:[yes/no]	100
ibm-listing-macro:[yes/no]	100
ibm-mainframe:[yes/no]	100
ibm-sync:[yes/no]	100
identifier-length:<max-length>	100
ignore-global-in-local-storage: [yes/no]	101
ignore-with-rollback: [yes/no]	101
implicit-init: [yes/no]	101
Include-main: [yes/no]	101
incomplete-subscript:[yes/no]	101
index-optimize:[yes/no]	101
indirect-redefines:[yes/no]	102
initcall:<program-name>	102
Initialize-fd:[yes/no]	102
initialize-filler:[yes/no]	102
initialize-opt:[yes/no]	103
initialize-pointer:[yes/no]	103
initialize-to-value:[yes/no]	103
isam-extfh: <DRIVER NAME>	103
isam-extfh-lib: <library to use for this extfh driver>	103
keep-copy-statement:[yes/no]	104
keep-org-src-line:[yes/no]	104
keep-unused:[yes/no]	104
key-dup-always-22:[yes/no]	104
keycompress: [integer between 0 and 9]	104
larger-redefines-ok:[yes/no]	105
line-seq-dos:[yes/no]	105
line-seq-mf:[yes/no]	105
line-seq-notrunc:[yes/no]	105
line-seq-recording-mode:[yes/no]	106
line-seq-unix:[yes/no]	106
link-only: [yes/no]	106
listing-sources: [yes/no]	106
local-storage-guard: 8 (internal use only)	106
loosy-comment[yes/no]	106
ls-expand-tab:[yes/no]	106
ls-ignore-record-size:[yes/no]	107

ls-utf16:[yes/no]	107
main:[yes/no]	107
main-as-object:[yes/no]	107
mainframe-vb:[yes/no]	107
makesyn: oldvalue=newvalue	108
makesyn-patch-preprocess: [yes/no]	108
manuallock: [yes/no]	108
max-literal-expand: 32 (internal use only)	108
mem-info: [yes/no]	108
mfcomment: [yes/no]	109
mf-compat-parser: [yes/no]	109
mf-ctrl-escaped-parser: [yes/no]	109
mf-file-optional:[yes/no]	109
mf-gnt: [yes/no]	110
mf-hostnumcompare:[yes/no]	110
mf-int:[yes/no]	110
mf-relativefile :[yes/no]	110
module-load-priority:[yes/no]	111
module-name-entry: [yes/no]	111
module-uppercase: [yes/no]	111
move-all-edited: [yes/no]	111
move-high-low-to-displaynumeric [error/zero/value]	111
move-picx-to-pic9:[cit / mf50 / mf40 / mvs / raw / iso / none]	112
move-spaces-to-comp3:[error/space/zero]	112
move-spaces-to-displaynumeric:[yes/no/error]	113
move-to-group-separated:[yes/no]	113
name:[any string]	113
no-realpath: [yes/no]	113
non-ibm-5.2-syntax: [(ok or yes)/(error or no)/warning]	114
nostrip: [yes/no]	114
notrunc: [yes/no]	114
not-reserved:[any reserved word]	114
null-param: [yes/no]	114
numeric-compare: [yes/no]	114
numeric-group: [yes/no/warning]	115
numval-validate: [yes/no]	115
obj-cit: [yes/no]	115
odo-slide: [yes/no]	115
optimize-move: [yes/no]	116
optimize-move-call:[yes/no]	116
optional-file: [yes/no]	116
pack-comp-4:[yes/no]	116
perform-osvs:[yes/no]	116
prepro_cut_line: [yes/no]	118
pretty-display:[yes/no]	118
printer-crlf:[yes/no]	118
profiling:[yes/no]	118
protect-linkage:[yes/no]	118
quote:[any single character]	118
raw-by-value: [yes/no]	118
raw-compare: [yes/no]	119
raw-pic9-display: [yes/no]	119
read-at-end-mf:[yes/no]	119
read-into-copy: [yes/no]	119
ready-trace:[y/n]	119
Enables paragraph tracing between READY TRACE and RESET TRACE procedural COBOL statements.	119
When set to yes,	119
In the interval between the READY TRACE and RESET TRACE statements, paragraph tracing output is written	

to the console in the format:	119
PROGRAM-ID: [program-id]: [paragraph name].....	119
recmode-f:[yes/no]	119
recmode-osvs:[yes/no]	120
recmode-v:[yes/no]	120
record-depending-iso:[yes/no]	120
redefine-identifier: [error / warning / ok]	120
region0: [yes/no]	121
relativefile-bigendian:[yes/no]	121
relax-bounds-check:[yes/no]	121
relax-level-hierarchy:[yes/no]	121
relaxed-syntax-check:[yes/no]	121
replace-additive:[yes/no]	121
return-opt:[yes/no].....	122
round-fp:[yes/no].....	122
rtncode-size: <integer>.....	122
rw-after-preprocess:[yes/no]	122
rw-mode-nopf:[yes/no]	122
rw-mode-nopf-dos:[yes/no].....	123
safe-linkage:[yes/no]	123
screen-exceptions:[yes/no]	123
screen-raw-keys:[yes/no]	124
sequential-line:[yes/no]	124
share-all-autolock:[yes/no].....	124
share-all-default:[yes/no]	125
share-all-manulock:[yes/no].....	125
sign-ascii:[yes/no]	125
sign-ebcdic:[yes/no]	125
sign-leading:[yes/no].....	125
sign-separate:[yes/no]	126
signed-comp6-as-comp3:[yes/no]	126
simple-trace:[yes/no].....	126
source-location:[yes/no].....	126
split-debug-mark:[yes/no]	126
spzero:[yes/no]	126
stack-check:[yes/no].....	127
static-call:[yes/no]	127
static-link:[function-name].....	127
sticky-linkage:[yes / no / fixed / variable].....	127
strict-compare-low: [yes/no]	128
strict-record-contains:[yes/no]	128
synchronized-double-word-bound:[yes/no]	128
synchronized-propagate-to-occurs: [yes/no] (Internal Use Only)	129
synchronized-propagate-to-occurs-with-group-size: [yes/no] (Internal Use Only).....	129
syntax-only:[yes/no].....	129
syntax-support:[ok / archaic / obsolete / skip / ignore / unconformable / error].....	129
tab-width:[integer].....	130
tally-register:[yes/no]	130
text-column:[integer].....	130
thread-safe:[yes/no].....	130
trace:[yes/no].....	130
trace-ts: [yes/no].....	130
trace-upon-systout: [yes/no].....	131
traceall:[yes/no].....	131
trap-unhandled-exception:[yes/no].....	131
truncate-listing:[yes/no]	131
unstring-use-move:[yes/no].....	131
use-defaultbyte:[yes/no].....	131

utf16-le:[yes/no].....	132
utf-8:[yes/no].....	132
validate-dep-on: [yes/no].....	132
validate-odo:[yes/no].....	132
validate-only:[yes/no]	132
value-of-id-priority: [yes/no].....	132
value-size-is-auto: [yes/no]	133
variable-rec-pad-mf:[yes/no].....	133
vbisam: [yes/no].....	133
vms-error-handler: [yes/no].....	133
when-compiled-function-all :[yes/no].....	133
wnone:[yes/no].....	133
xparse-event:[yes/no]	133
zero-length-trim:[yes/no]	134

Compiler Environment Variables 135

COB_AR <program>	135
COB_ARFLAGS <ar flags>	135
COB_CC <program>	135
COB_CFLAGS <cc flags>	135
COB_CONFIG_DIR<directory>.....	135
COB_CONSOLE_CP=<codepage-id>	135
COB_COPY_DIR<directory>	136
COB_EXTRA_FLAGS.....	136
COB_LDADD <ld flags>	136
COB_LDFLAGS <ld flags>	136
COB_LIBS <libs>.....	136
COB_OPTIMIZE_FLAG<cc flags>.....	136
COB_OPTSIZE_FLAG=[optimization flag]	136
COB_STDUNIX <1/0>	137
COB_SUNSTUDIO12=[Y/N]	137
COBCPY <directory list>	137
COBCTMP=<directory>.....	137
COBITOPT=[string of command-line compiler flags]	137
COBOPT=[string of command-line compiler flags]	138
COBOLITDIR=<directory>	138
TMPDIR or TMP=<directory>	138

COBOL-IT Runtime Options 138

COBOL-IT runtime parameters	138
--checkpoint <file>.....	138
--console, -c.....	138
--debug, -d.....	138
--debug, -d --remote -r.....	138
--help, -h.....	139
--reload.....	139
--version, -V	139
COBOL-IT runtime environment variables	139
COB_CALL_CASE=xul [where x=exact match, u=uppercase, l=lowercase].....	139
COB_LOAD_CASE=xul [where x= exact match, u=uppercase, l=lowercase]	139
COB_CONSOLE_CP=<code page>.....	140
COB_CURRENT_DATE	140
COB_DEBUG_ALLUSER=1	140
COB_DEBUG_ID=<debug-id>.....	141
COB_DEBUG_MODULES=<program-id1>:<program-id2>.....	141
COB_DEBUG_STARTUP_FILE=<filename>	142
COB_DEBUG_TMP=<directory>.....	143
COB_DISPLAY_PRINTER=<filename>	143

COB_DUMP=<filename>	143
COB_ERROR_FILE=<filename>	144
COB_EXTFH=<EXTFH Entry>	144
COB_EXTFH_FLAT=<EXTFH Entry>	144
COB_EXTFH_INDEXED=<EXTFH Entry>.....	145
COB_EXTFH_LIB=[list of shared libraries]	145
COB_FILE_CASE=[UPPER LOWER]	145
COB_FILEMAP_CASE=[UPPER/LOWER]	145
COB_FILE_PATH=[PATH]	145
COB_FILE_RELATIVE_MF=Y	146
COB_FILE_TRACE=[Y/N]	146
COB_FULL_CANCEL=[Y/N]	146
COB_KEY_DUP_ALWAYS_22=[Y/N]	146
COB_LIBRARY_PATH =[PATH]	147
COB_LOAD_CASE=[UPPER/LOWER]	147
COB_LOAD_PRIORITY=[Y/N]	147
COB_LS_DOS=[Y/N]	147
COB_LS_FIXED=[Y/N]	148
COB_LS_NULLS=[Y/N]	148
COB_NO_DOT_DAT	148
COB_NO_SIGNAL=[Y/N]	148
COB_PAD_BUG=[0/1]	148
COB_PRE_LOAD=[list of modules]	149
COB_PROFILING_DIR	149
COB_PROFILING_EACH_MODULE	149
COB_RTL_CP=<codepage>	150
COB_RUNTIME_CHECK_TRACE=[Y/N/Module list separated by ; or : (Windows)]	150
COB_SCREEN_DISABLE_REFORMAT=[Y/N]	151
COB_SCREEN_ESC=[Y/N]	151
COB_SCREEN_EXCEPTIONS=[Y/N]	151
COB_SCREEN_INPUT_BOLDDED=[Y/N]	152
COB_SCREEN_INPUT_FILLER=[char]	152
COB_SCREEN_INPUT_INSERT_TOGGLE=[Y/N]	152
COB_SCREEN_INPUT_REVERSED=[Y/N]	152
COB_SCREEN_INPUT_UNDERLINED=[Y/N]	152
COB_SCREEN_RAW_KEYS=[Y/N]	152
COB_SCREEN_UPDATE_FIRST_KEY_ERASE=[Y/N]	153
COB_STDUNIX=[Y/N]	153
COB_SWITCH_0... COB_SWITCH_16	154
COB_SYNC=[Y/N]	154
COB_VAR_REC_PAD=[Y/N]	154
COBLPFORM="n:n:n: : : : : :n"	154
COBOLITDIR=<directory>	155
TMPDIR or TMP=<directory>	155
File Status Codes	156
Runtime Error Codes	159
Data Memory allocation	161
BINARY, COMPUTATIONAL	161
bin-opt-strict:[yes/no]	161
BINARY-CHAR, BINARY-CHAR SIGNED	161
BINARY-CHAR UNSIGNED	161
BINARY-C-LONG, BINARY-C-LONG SIGNED	161
BINARY-C-LONG UNSIGNED	162
BINARY-DOUBLE, BINARY-DOUBLE SIGNED	162

BINARY-DOUBLE UNSIGNED.....	162
BINARY-LONG, BINARY-LONG SIGNED, SIGNED-LONG, SIGNED-INT.....	162
BINARY-SHORT BINARY-SHORT SIGNED SIGNED-SHORT	162
BINARY-SHORT UNSIGNED UNSIGNED-SHORT	164
COMPUTATIONAL-1	164
COMPUTATIONAL-2.....	164
COMPUTATIONAL-3 PACKED-DECIMAL.....	164
COMPUTATIONAL-4.....	164
bin-opt-strict:[yes/no].....	165
COMPUTATIONAL-5	165
COMPUTATIONAL-6.....	165
COMPUTATIONAL-X.....	165
bin-opt-strict:[yes/no].....	166
DISPLAY	166
INDEX	166
POINTER PROGRAM-POINTER	166
Using EXTFH-Compliant Indexed File Systems.....	167
COBOL-IT includes EXTFH Libraries.....	167
VBISAM	167
-fvbisam	167
vbisam: [yes/no].....	168
BerkeleyDB	168
-fbdb	168
bdb: [yes/no]	168
D-ISAM	168
-fdisam	168
disam: [yes/no].....	168
dcheck	168
C-Tree ACE	169
Documentation	169
Installation.....	170
Compiling.....	171
Running.....	171
Start/Stop Engine	171
Data file location	171
Reserved Word List.....	173
Intrinsic Function List.....	179
COBOL-IT® Library Routines.....	185
C\$CALLERNAME.....	187
C\$CHDIR	188
C\$COPY	189
C\$DEBUG	190
C\$DELETE.....	201
C\$FILEINFO	202
C\$JUSTIFY	204
C\$MAKEDIR	204
C\$NARG	206
C\$PARAMSIZE	207
C\$PID	208
C\$SLEEP.....	209
C\$TOLOWER	210
C\$TOUPPER.....	211
CBL_ALLOC_DYN_MEM	211

CBL_ALLOC_MEM.....	213
CBL_AND.....	215
CBL_CHANGE_DIR.....	217
CBL_CHECK_FILE_EXIST.....	218
CBL_CLOSE_FILE.....	220
CBL_COPY_FILE.....	221
CBL_CREATE_DIR.....	222
CBL_CREATE_FILE.....	223
CBL_DEBUGBREAK.....	225
CBL_DELETE_DIR.....	226
CBL_DELETE_FILE.....	227
CBL_ERROR_PROC.....	228
CBL_EQ.....	230
CBL_EXIT_PROC.....	231
CBL_FLUSH_FILE.....	232
CBL_FREE_MEM.....	232
CBL_FREE_DYN_MEM.....	233
CBL_GET_CURRENT_DIR.....	234
CBL_IMP.....	235
CBL_NIMP.....	237
CBL_NOR.....	238
CBL_NOT.....	239
CBL_OC_NANOSLEEP.....	240
CBL_OPEN_FILE.....	241
CBL_OR.....	243
CBL_READ_FILE.....	244
CBL_RENAME_FILE.....	246
CBL_TOLOWER.....	247
CBL_TOUPPER.....	248
CBL_WRITE_FILE.....	249
CBL_XOR.....	251
SYSTEM.....	252
X"91" function 11.....	253
X"91" function 12.....	254
X"91" function 15.....	255
X"91" function 16.....	256
X"F4".....	257
X"F5".....	258
The Runtime Data Structure (rtd)	259
The COBOL-IT Region Interface	260
Overview.....	260
The REGION API.....	261
The Sample Programs.....	263
What cobc does	266
Creating a shared object/dll (Windows).....	266
Creating an executable (.exe) (Windows).....	267
Compiling a "C" program with cobc (Windows).....	269
COMPATIBILITY TOPICS.....	270
cobmf	270
Overview of cobmf.....	270
>cobmf [return].....	270
Cobmf options.....	271

Using the -CIT option with cobmf	271
What cobmf does	271
How cobmf handles the -C “[directive]” compiler flag.....	272
COBOPT.....	273
Table of equivalents to cob compiler flags.....	273
Table of equivalents to compiler directives	276
THE COBOL-IT DEBUGGER ENGINE (COBCDB).....	282
Conventions Used.....	282
The Debugger Prompt.....	282
Source Location	282
Variables names	282
Usage of the COBOL-IT Debugger:.....	284
command-line parameters	284
program name	284
options.....	284
-listdid	284
-n	284
-p <did>.....	284
-r host:port.....	285
-trace	285
-w <did>.....	285
-y tty	285
Debugger Commands	286
break	286
break [-t] label.....	286
break [-t] module!label	286
break [-t] module!line-nr.....	286
break [-t] module!0	287
bt	287
continue.....	287
contreturn	288
delete <x>	288
frame <frame-number>.....	288
info	289
info locals.....	289
info profiling.....	289
info sources	290
info target.....	290
kill	290
list	290
next	291
print <variable-name>.....	291
printh <variable-name>.....	292
quit	292
replace.....	292
>replace <oldprefix> : <newprefix>	293
>replace ?.....	293
>replace <no arguments>	293
set.....	293
set prompt <prompt string>	294
set var <variable-name> <variable-value>	294

set varh <variable-name> <variable-value-hex>	294
step	294
stop	295
up -[n]	295
version	296
Debugger Events	296
-event-breakpoint-hit	296
-event-continue	296
-event-contreturn	296
-event-end-stepping-range	296
-event-next	296
-event-program-exited	296
-event-step	297
Our Sample Programs	297
hello.cbl	297
subpgm.cbl	298

Compiler & Runtime Reference

The COBOL Compiler: `cobc`

`cobc` is the COBOL-IT Open Source COBOL compiler.

`cobc` translates COBOL programs to C code and compiles it using the C platform compiler. However, this C code is just a compilation artifact that provides portability and performance. It is not meant to be maintained, or even read. Except for the cases where the developer will be interfacing COBOL-IT with other libraries or dealing with other similarly low-level task, there is no need for the developer to even be aware of this intermediate C code, or any other C-related topic.

For a more detailed presentation of the steps `cobc` goes through in creating an executable object, see the topic “What `cobc` does”.

This chapter provides an overview of supported compiler flags, and environment variables, and includes a section highlighting some important usage cases.

Usage: `>cobc [options] file...`

The COBOL-IT Compiler supports over 100 compiler flags. For a full list, execute the command:

```
>cobc -help
```

Informational Flags

Informational compiler flags do not product any objects, and are used to display information about the COBOL-IT compiler.

-check-codepage <codepage-id>

The `-check-codepage <name>` compiler flag is an informational compiler flag that you can use to check if a given codepage is recognized by the ICU library.

As examples:

To check for support of codepage 500, (which is supported):

```
C:\COBOL\COBOLIT>cobc -check-codepage 500
```

```
500 is found as ibm-500_P100-1995
```

To check for support of codepage 13488 (which is not supported):

```
C:\COBOL\COBOLIT>cobc -check-codepage 13488
```

Internal Error: can't open converteur 13488 : U_FILE_ACCESS_ERROR

--help

Display this message

--list-codepage

The `--list-codepage` compiler flag is an informational compiler flag that lists all supported codepages in the following format:

Codepage: [list of synonyms for codepage]

As an example:

UTF-8 : UTF-8 ibm-1208 ibm-1209 ibm-5304 ibm-5305 ibm-13496 ibm-13497 ibm-17592 ibm-17593 windows-65001 cp1208 x-UTF_8J unicode-1-1-utf-8 unicode-2-0-utf-8

--list-intrinsics

Display intrinsic functions

--list-mnemonics

Display mnemonic names

--list-reserved

Display all reserved words

--version, -V

Display compiler version

Standard Flags

Standard compiler flags are applied to COBOL-IT's basic build process, which consists of separate preprocess, translate, compile, assemble, and link steps, and are used to direct the compiler in basic functions such as locating copy files, placing object files, creating listing files, adding error-checks, and adding information required for running with the COBOL-IT debugger.

-b

Links all input files into a single dynamically loadable module

-c

Compile and assemble, but do not link

This is equivalent to ``cc -c'` (Linux/Unix) or `'cl -c'` (Windows).

On Linux/Unix systems, the output is saved in a ``*.o'` file.

On Windows systems, the output is saved in a ``*.obj'` file.

-codepage <codepage-id>

Defines the encoding of PIC X in memory.

If `-source-codepage` is specified, the compiler converts from the `codepage-id` used in the `-source-codepage` compiler flag to the `codepage-id` used in the `-codepage` compiler flag.

The debugger makes use of the `-codepage <codepage-id>` compiler flag setting, and converts alphanumeric (PIC X) data to/from UTF-8 when sending data to/from GUI interface.

The conversion of data described with `USAGE NATIONAL` to PIC X also uses the `-codepage <codepage-id>` compiler flag setting. The `-codepage <codepage-id>` compiler flag sets the default code page for National literals (`N'xxx'`) and for conversion `DISPLAY-OF` and `NATIONAL-OF` when used without code page indication.

If `-codepage <codepage-id>` is not specified then if `-source-codepage <codepage-id>` is specified, the setting of `-source-codepage` is used for the conversion of data described with `USAGE NATIONAL` to PIC X.

-conf=[+]<file>

Causes a user-defined compiler configuration file to be referenced either as the default configuration file, or, if the `“+”` operand is used, as an addition to the base compiler configuration file. For rules on how the base compiler configuration file is set, see the `-std=<file>` compiler flag.

-constant "key=value"

Provides a way to define constants that can be tested for purposes of conditional compilation.

After using the `-constant "key=value"` compiler flag, the conditional compilation below will test true.

```
$if key=value  
$else  
$end
```

-debug

Enables all run-time error checking. See Guidelines for Enforcing Bounds-Checking for more details.

-debugdb [=<DebugDB-name>]

The `-debugdb` compiler flag causes the compiler to store debugging meta information in an SQLite3 database.

When compiling with `-g`, the COBOL-IT compiler stores all debugging meta information in the program binaries. This could make programs compiled for debug very huge. In some situations, it could prevent the program from loading into memory.

When used without a filename, and when used with `-g`, stores all debugging information into a file name `<modulename>.dbd`. Copy this file to the same location as the the object file `.so` or `.dll`. This will permit the runtime debugger to load the debugging information dynamically when needed.

If you have multiple programs in your project for which debugging information must be stored, you must use `-debugdb=<DebugDB-name>`, where `DebugDB-name` is the name of an SQLite3 database that stores metadata for all of the programs in the project.

During a debug session, the runtime debugger will check for the existence of the `COB_DEBUGDB` environment variable containing the full path to the DebugDB file. If the environment variable is not set, the runtime will attempt to retrieve the location of the `COB_DEBUGDB` data file from the location of the compiled object.

Currently only 1 database may be used at a time. This means that the Customer must use the same one for all of his programs. Several programs may write metadata to the same database.

-dump-config

Dumps all of the compiler config file settings being used in the compilation session to stdout. The `-dump-config` compiler flag can be used with or without a COBOL source file.

To dump default compiler configuration file setting to a file:
>cobc `-dump-config` > [dumpfile].

To dump compiler configuration file settings for the current compilation:
>cobc [compiler flags] `-dump-config` hello.cbl > [dumpfile].

-err <file>

Causes errors and warnings to be written to `<file>` instead of `stderr`

-ext<extension>

Adds `<extension>` to list of default copy file extensions. For example, to direct the compiler to search for copy files with a `.fd` extension, use the compiler flag `"-ext fd"`. Multiple iterations of `-ext` are used to express multiple non-default copy file extensions. For example, `"-ext fd -ext sl"`.

-fixed

Instructs the compiler that source code is in the fixed source format. The `-fixed` compiler flag is assumed, by default.

-free

Instructs the compiler that source code is in the free, or terminal source format.

-g

Produce debugging information in the output.

-initcall=<program-name>

The `initcall` compiler flag names modules to be called immediately before the first statement of a program is executed.

-l <lib>

Causes the library `<mylib>` to be used by the linker. Note- The linker will look for `mylib` in the path indicated by the `-L` compiler flag.

-linkage-desc=[program.xsd]

Generates an XSD file from the Source file. This process parses the program, and records the names of the entry points, and for each entry point, the linkage section items. This information is stored in an XML format in a file with a `.xsd` extension. The `-linkage-desc` compiler flag is invoked in the Developer Studio Web Services Perspective, when generating an XSD.

Command-line usage:

➤ `cobc -linkage-desc=[programname].xsd [programname].cbl`

-m

Builds a dynamically loadable module. The `-m` compiler flag is implied by default.

-makesyn "oldvalue=newvalue"

Provides a way to make a reserved word a synonym for another reserved word. The first word, represented by "oldvalue" becomes a synonym of the second word, represented by "newvalue". A common usage is to make `COMP` a synonym of `COMP-5`. To do this:

>`cobc -makesyn comp=comp-5 hello.cbl`

The `-makesyn "oldvalue=newvalue"` compiler flag provides compatibility with the `MAKESYN` directive.

A COBOL verb or field-name may be used as “old-value”. The COBOL-IT compiler will replace all instances of this “old-value” with the “new-value” when compiling.

CAUTION- While this provides an equivalent capability to the implementation of the MAKESYN directive in other COBOLs, the order of the parameters is reversed. COBOL-IT requires that the “old-value” be listed first, and followed by the “new-value”.

-o <file> | <dir>

Causes a compiled object to be output into <file> or <directory>. When using a directory name, standard naming conventions are used, and the output is written to the named directory.

-preprocess=<CMD> [input file]

Causes <CMD> to be run after the COBOL pre-processing step. <CMD> is a script of batch file in which an external pre-processor is run.

Consider an example, related to the use of Pro*COBOL. A simple Pro*COBOL script would be:
procob iname=%1 oname=%2

Where INAME would be the input file, which needed to be preprocessed, and ONAME would be the resulting output file. To cause the COBOL-IT precompiler to run its precompilation prior to calling this script, run the script in a .BAT file (in Windows), and invoke as follows:

```
cobc -preprocess=myprocob.bat mysource.cbl
```

For more detail, see Guidelines for use of -preprocess=cmd for more details. The -preprocess compiler flag is only available in the Enterprise Edition of the Compiler Suite.

-save-temps (= <dir>)

Causes all intermediate files to be preserved. Note- “intermediate files” are the “C” source and header files that are created during the compilation process. These files will be located in a subdirectory named “c”, when using the -save-temps compiler flag.

-source-codepage <codepage-id>

Defines the code page to be used when editing the source and the code page used for string literals in the COBOL source code.

When used with -codepage <codepage-id>

If -source-codepage is specified, the compiler converts from the codepage-id used in the -source-codepage compiler flag to the codepage-id used in the -codepage compiler flag.

When used without -codepage <codepage-id>

If -codepage <codepage-id> is not specified then if -source-codepage <codepage-id> is

specified, the setting of `-source-codepage` is used for the conversion of data described with `USAGE NATIONAL` to `PIC X`.

For more details, reference documentation on the `-codepage <codepage-id>` compiler flag.

-std=<dialect>

Causes one of the dialect-oriented compiler configuration files to be used instead of the default compiler configuration file. COBOL-IT provides several dialect-oriented compiler configuration files, which can be located in `$COBOLITDIR/share/COBOL-it/config/*.conf` on Unix/Linux-based systems and in `%COBOLITDIR%\config/*.conf` on Windows-based systems.

For a specific dialect :

COBOL2002	COBOL 2002
COBOL85	COBOL 85
ibm	IBM Compatible
mvs	MVS Compatible
bs2000	BS2000 Compatible
mf	Micro Focus Compatible

Example: `>cobc -std=ibm hello.cbl` causes the compiler to select the file `ibm.conf` as the standard compiler configuration file, instead of `default.conf`.

-sysin=<input file>

Allows redirection of `ACCEPT` statements at compilation time. `<input file>` is the file used by `ACCEPT` instead of the console. It must be a line sequential file.

For example: `cobc -sysin=sysin.txt testit.cbl`

-sysout=<output file> [,S/L [,Min [,Max]]]

Allows redirection of `DISPLAY` statements at compilation time. `<output file>` is the file used by `DISPLAY` instead of the console. By default, `<output file>` is created as a line sequential file.

The `-sysout` compiler flag provides compatibility with the `OUTDD` compiler directive supported by some other COBOL compilers.

For example: `-sysout=sysout.txt testit.cbl`

When `<output file>` exists, output is appended to the existing file. When `<output file>` does not exist, it is created in the current working directory. When dot `'.'` is used as a file name, `sysout` is sent to `stdout`. As an example: `-sysout=.,SEQ,100` produces output on `stdout` as a sequential record of 100 bytes.

Use the `S` flag, in conjunction with the `Min/Max` flags to cause `<output file>` to be created as a binary sequential file, with minimum and maximum record lengths.

Use the L flag optionally. You may use the Min/Max flags with the L flag. For the case where an output string exceeds a maximum record length, a line feed (Linux/Unix) or CR/LF (Windows) is inserted after a maximum length has been reached, and the output string is then continued.

For example: `-sysout=sysout.txt testit.cbl`

The `-sysout` compiler flag supports a file name of dot `'.'`. When dot `'.'` is used as a file name, `sysout` is sent to `stdout`. As an example:

`-sysout=.,SEQ,100` produces output on `stdout` as a sequential record of 100 bytes..

General Format

`-sysout=filename [,S/L [,Min [,Max]]]`

Syntax

, may be replace by `“.”`

No spaces are allowed in the compiler command

S = Sequential (Binary Sequential)

L = Line Sequential

If a Max value is not given for Sequential files, then Max = Min.

If a Max value is not given for Line Sequential files, then Max = Min, and Min = 0.

General Rules

The runtime will write as many records as needed.

If the data written is smaller than the Min value, then

the runtime will pad the output with SPACES up to min-length in the line-sequential output file, and the runtime will pad the output with LOW-VALUES up to min-length in the sequential (binary sequential) output file.

-t <file> | <dir>

Causes a program listing to be output into `<file>` or `<directory>`. When using a directory name, standard naming conventions are used, and the output is written to the named directory with a `.lst` extension.

The listing file produced by the `-t` compiler flag includes memory mapping information. At the end of the listing, in lines that are commented, the list file reports the size and offsets of the data fields in the Working-Storage Section.

The listing file produced by the `-t` compiler flag preserves comments, and can be compiled by the COBOL-IT compiler.

When a compilation string designates no configuration file, and when not using the `-E` compiler flag, the settings of the `default.conf` file are printed to the listing file that is named, or implied when using the `-t` compiler flag.

When a compilation string designates a configuration file, using the `conf=xxx.conf` compiler flag,

then the settings of the named compiler configuration flag are printed in the listing file that is named or implied when using the `-t` compiler flag.

When using the `-b` compiler flag to combine several programs into a single library, and using the `-t [filename]` compiler flag, a single listing is created that includes all of the programs in the library.

`-use-extfh <NAME>`

Names an EXTFH File handler to be used, enabling the use of an external file system.

`-use-extsm <NAME>`

Runtime module to be used, enabling the use of an external sort handler.

`-v`

Produces verbose output. The output of the `-v` compiler flag displays, all of the steps, and intermediate programs created by the compilation.

`-x`

Builds an executable program.

The executable produced with the `-x` compiler flag includes the main function in the output. This option takes effect at the translation stage. If you compile with `-x -C`, you will see the main function at the end of the generated C file.

WARNING: When using `-x`, the first source module or object (`.o` on UNIX or `.obj` on windows) given on the command line **MUST** be the main module of the program. All other modules, libraries or object files may follow in any order.

Example:

```
cobc -x mymain.cob other1.cob ...
```

`--xdd-prefix=<dir>`

Causes the XDD file created with the `-fgen-xdd` compiler flag to be stored in the directory named in `<dir>`. The folder name described in `<dir>` must be followed by a slash “/” in Unix/Linux, or a back-slash “\” in Windows.

As an example:

```
>cobc -fgen-xdd -xdd-prefix=/my/doc/xdd/ hello.cbl causes all xdds generated in the compilation to be stored in the /my/doc/xdd/ folder.
```

-C

Interrupts the compilation after converting COBOL to C.

The output is saved in files with different extensions:

- .c Files with extension '.c' hold the main code
- .c.h Files with extension '.c.h' hold the COBOL field storage
- .c.d.h Files with extension '.c.d.h' hold debug information
- .c.l.h Files with extension '.c.l.h' hold temporary fields

-D <define>

Passes <define> to the “C” Compiler

-E

Interrupts the compilation after the preprocessing of the COBOL code, without doing any translation to “C”, compilation, assembly, or linking. The preprocessed code is output to stdout by default, and reproduces the preprocessed COBOL source code.

Output from the –E compiler flag retains the format (free or fixed) of the original source code.

Note- Using the -E compiler flag, the references to expanded copy files includes path information on the commented-out reference to the copy file. As an example, in the sample below, the program was compiled with –I .\copy –ext sl . Note that the path to the SELECT statement is preserved:

```
*"/copy/reswords.sl"  
*  
  SELECT reswords ASSIGN TO "reswords"  
  ORGANIZATION IS INDEXED  
  ACCESS IS DYNAMIC  
  RECORD KEY IS reserved-word  
  FILE STATUS IS reswords-stat.
```

-G

Produces debugging information in the output, allowing “C”-level debugging.

To perform “C” level debugging, use the COBOL-IT Developer Studio.

COBOL-IT translates COBOL to “C” and uses the host “C” compiler to compile the translated source. As preparation, compile your COBOL programs with the –G compiler flag. “C” modules should be compiled for debugging as well.

The –G COBOL compiler flag allows the COBOL program to be include information for the “C” debugger. This corresponds to the gcc –g compiler flag.

Using the Debug Attach functionality of the Developer Studio to Attach the COBOL Debugger to an Application, you can enter the COBOL Debugger, then start the “C” debugger, and proceed

your debuggeing with both the “C” and COBOL debuggers running.

The Eclipse IDE for C/C++ Developers, and “C” compiler are required for this exercise.

-I <path>[,ext1,ext2,..,extn][@<LibName>] | <command-file>

Where:

<path> is the path that the compiler will search for the COPY file.
ext1,ext2,..,extn are file extensions to be included when performing the search for the COPY file.
@ represents the “@” character, which is placed before <LibName>
<LibName> is the Library named in a COPY statement that contains a reference to a Library. For example:

COPY <filename> IN/OF LibName.

<command-file> is the name of an existing file. When –I is followed by the name of an existing file, that file is read, and each line is treated as a parameter of the –I compiler flag.

For more details, and rules governing the treatment of file extensions, and rules applying to the COPY IN/OF LIBNAME syntax, see Guidelines for Searching and Locating COPY files.

-L <directory>

Adds <directory> to the library search path.

-MF <file>

Writes dependency list into <file>.

-MT <target>

Names the target file used for the dependency list.

-O, -Os, -O2

Enables optimization. Note- ‘-O’, ‘-Os’ and ‘-O2’ are passed to the “C” compiler as is and used for “C”-level optimization. For more details on the usage of the optimization compiler flags, see the Guidelines for use of the Lib Optimizer below.

The -O and -O2 compiler flags now automatically enable the following compiler flags by default: -fbin-opt, -fcmp-opt, -fcmp-inline, -ffp-opt, -foptimize-move, -foptimize-move-call, -ffast-op, -findex-optimize, -fdecimal-optimize. For more details about each of these optimizing compiler

flags, see below.

-R <directory>

Adds <directory> to runtime library search path (if supported).

-S

Interrupts the compilation after after output of the assembly file. Translated C files are compiled By cc. The output is saved in a file with a .s extension.

The -S compiler flag is only available when using the gcc C compiler.

-Wc CC_opt

Passes CC_opt directly to the C Compiler.

CC_opt is a compiler flag, or string, that can be processed by the C compiler.

As an example, the command:

```
>cobc -Wc -O2 hello.cbl
```

Affects the manner in which the C compiler command-line is built, adding -O2 to the compile string.

To view the C compiler command-line that is built by the COBOL-IT compiler, add the -v compiler flag to the compile string.

```
>cobc -v hello.cbl
```

```
...
```

```
cobc:0: cl /c /I "C:\Cobol\CobolIT\include" /DCOB_HAS_THREAD /W0 /nologo /GF /MD /Fo"hello.obj" "C:\Users\ROBERT~1\AppData\Local\Temp\cob652217784_5.c" cob652217784_5.c
```

```
...
```

-Wl LD_opt

Passes LD_opt directly to the Linker.

LD_opt is an option, or string that can be processed by the Linker.

To view the link command-line that is built by the COBOL-IT compiler, add the -v compiler flag to the compile string.

```
>cobc -v hello.cbl
```

```
...
```

```
cobc:0: link /DLL /MANIFEST /out:"hello.dll" /nologo "hello.obj" "C:\Cobol\CobolIT\lib\libcobit_dll.lib" /DEFAULTLIB:MSVCRT.LIB
```

Guidelines for Searching and Locating COPY files

The COBOL-IT compiler can be directed to search for copy files in directories named by either the COBCPY, or COB_COPY_DIR environment variable, or with the use of the `-I` compiler flag. Copy file name resolution is refined with the use of the `-ext` compiler flag.

By default, the COBOL-IT compiler will search the current directory, and `$COBOLITDIR\copy` for named copy files, and if the copy files have no explicit file name extensions, the COBOL-IT compiler will search for default copy file extensions.

Default copy file extensions are:

- .CPY
- .COB
- .CBL
- .cpy
- .cob
- .cbl
- no extension

The COBOL-IT compiler will then check the environment variables COBCPY and COB_COPY_DIR for paths to add to the default search paths.

At the command line, you may add more directories to search with the `-I <directory>` compiler flag, and you may add more extensions to the default file extensions searched with the `-ext <extension>` compiler flag.

Example: Consider a case where a program, `myprog.cbl` has a copy file declared as follows:

```
COPY "customer.cpy".
```

And where "customer.cpy" is contained in a subdirectory called "copy".

```
>set COBCPY=.\copy  
>cobc myprog.cbl
```

Example: Consider a case where a program, `myprog.cbl` has a copyfile declared as follows:

```
COPY "customer".
```

And where "customer.fd" is contained in a subdirectory called "copy". .fd is not a default extension, so both the directory and the extension need to be given to the compiler, to find the file.

```
>set COBCPY=.\copy  
>cobc -ext=fd myprog.cbl    or  
  
>cobc -I .\copy -ext=fd myprog.cbl
```

Locating COPY books

The COBOL-IT compiler will search for copy files:

First- In the same folder as the source file

Second- In the folder named by the `-I` compiler option

Third- In the folder named by the `COBCPY` environment variable

Fourth- if there has been no other indication in the environment or through the `-I` flag, in the folder `$COBOLITDIR/copy` on a Unix/Linux-based system and `%COBOLITDIR%\copy` on a Windows-based machine.

Note that the listing file will expand the copy file, and indicate which file was located:

```
SCREEN SECTION.  
*#1 "../COPY/SAMPLE1X.CPY"
```

Resolving COPY book names

Resolving the statement

```
COPY "MYFILE".
```

The file is located if the compiler locates `myfile` followed by any of the following extensions: `.CPY`, `.cpy`, `.CBL`, `.cbl`, `.COB`, `.cob`, any extension passed with `-ext` compiler option or with no extension at all.

Resolving the statement

```
COPY "MYFILE.CPY".
```

The file is located if the compiler locates the file `"myfile.cpy"`.

Note:

The copy file should not have the same name as the object file.

The target of the `COPY` command is case-sensitive on UNIX systems.

-I <directory> compiler flag causes `<directory>` to be searched for copy files

Example: `cobc -I .\copy program1.cbl`

-ext< extension> compiler flag causes `<extension>` to be included in search for copy files.

Example: `cobc -I .\copy -ext ws program1.cbl`

For systems where copy file searches are case sensitive:

`-ffold-copy-lower` Fold COPY subject to lower case (Default no transformation)

`-ffold-copy-upper` Fold COPY subject to upper case (Default no transformation)

Using -fcurdir-include compiler flag

The -fcurdir-include compiler flag causes COPY files to first be searched for in the current directory. (before the -I <Path>). The COPY search is performed for files with default extensions, and with extensions described with the -ext compiler flag. This is the default behavior of the compiler.

The -fno-curdir-include compiler flag causes the search for a COPY file to not search for COPY file in the current directory, unless that directory is named by a -I compiler flag, or by a COB_COPY_DIR, or COBCPY environment variable.

Using -I <path>[,ext1,ext2,..,extn][@<LibName>] / <command-file> compiler flag:

The following rules apply to the treatment of file extensions:

If file extensions are specified, then the compiler will limit its COPY file search to files with these extensions inside the specified <path>.

If no file extensions are specified, then standard extensions and those specified with the -ext compiler flag are used for the COPY file search.

File extensions must be named when the COPY <path> is the current directory (“.”).

To specify that files with no extension should be included in the COPY file search, use a single dot “.”.

For example, the following string would check for files with no extension, and with the .cpy extension:

```
>cobc -I /opt/mycopys,..,cpy
```

Note that if your file is declared with an extension, for example:

```
COPY test.cpy
```

Then, for the purposes of the COPY file search, it is not necessary to apply any extensions. In this case, you would want to check for files with no extension, as follows:

```
>cobc -I /opt/mycopys,.
```

That is- you would follow <path> by a comma, and then a single dot “.”, indicating that files with no extension should be included in the COPY file search.

The following rules apply to cases where a COPY <filename> IN/OF LibName statement is being interpreted: Consider the case where source code contains the line:

COPY MyCpy IN/OF LIBA.

Note that the default behavior of the compiler in this case is to look in the current directory for a subdirectory LIBA, and to search that directory for files called MyCpy with all of the standard file extensions. If not found, the compiler would continue the search in all <path> specified by the -I compiler flag, and would ignore the LibraryName.

If no @<LibName> is specified in the -I compiler command, then the default behavior of the compiler is assumed.

If @<LibName> is specified in the -I compile command, then the default behavior of the compiler is not applied. Instead, the <path> named in the -I compiler command is searched for the COPY file.

If the COPY file is not located in the <path>, then the search fails.

Examples:

>cobc -I /opt/copy looks for COPY files with standard extensions in /opt/copy

>cobc -I /opt/copy,.cpy looks for COPY files with the .cpy extension in /opt/copy

>cobc -I /opt/copy,..,cpy looks for COPY files with no extension or with the .cpy extension in /opt/copy

>cobc -I /opt/copy,..,cpy@LIBA looks for COPYfiles with no extension or with the .cpy extension in /opt/copy when resolving a COPY file described as IN/OF LIB, for example:

COPY [filename] IN/OF LIBA

>cobc -I <command-file> reads <command-file> and interprets each line as a command in the -I command string.

Guidelines for enforcing bounds-checking

The use of the “-debug” compiler flag enables all bounds-checking. Effectively, this is the equivalent of making the following settings in the compiler configuration file (default.conf, for example):

```
#Bound Pointer and base vars  
EC-BOUND-PTR: yes
```

```
#Field Subscript check Field(idx)  
EC-BOUND-SUBSCRIPT:yes
```

```
#Field Ref check Field(offset:len)  
EC-BOUND-REF-MOD:yes
```

#Non-numeric data check for PIC 9/USAGE COMP-3 target data items.
EC-DATA-INCOMPATIBLE:yes

If you want to enable/disable bounds checking for individual items you can do this via the configuration file. When compiling with `-debug`, the default values for the above-mentioned compiler configuration flags would all be set to “yes”. Re-setting a flag to “no” would override the `-debug` compiler configuration flag.

In the absence of the `-debug` compiler flag, the default values for the above-mentioned compiler configuration flags would all be set to “no”. Re-setting a flag to “yes” would override the default behavior, and enable a specific bounds-checking behavior.

otherwise the default is for all bounds checking to be disabled.

Guidelines for optimizing performance

The COBOL-IT Compiler Suite provides guidelines for optimizing your generated code. You can cause large modules to be split into separate C functions. The “C” Compiler can then more effectively optimize the resulting code.

-O compiler flags

- The `-Os` compiler flag causes large modules to be split, and optimized C code to be produced.
- The `-Os -O` sequence of compiler flags causes the “C” Compiler to optimize to reduce object size when compiling the optimized C code.
- The `-O -Os` sequence of compiler flags causes the “C” Compiler to optimize to maximize execution speed, when compiling the optimized C code. Note that objects optimized for performance may be larger in size.
- The `-O` compiler flag causes optimized C code to be produced, and causes the “C” Compiler to optimize to maximize execution speed. However, with very large modules, some “C” Compilers may fail. The `-Os` compiler flag can be used to break the large modules down, and avoid this problem.

Optimizations enabled with the `-O` compiler flag

Compiler Flag	Default	What it does	Comments
bin-opt:[yes/no]	No	Enables binary operation optimization.	bin-opt :yes is set by default with use of the <code>-O</code> compiler flag. If you wish to disable bin-opt when using <code>-O</code> , user the <code>-fno-bin-opt</code> compiler flag.
decimal-optimize : [yes/no]	No	Enables optimization of the conversion of	decimal-optimize :yes is set by default with

		decimal-encoded numeric values to binary in COMPUTE statements.	use of the <code>-O</code> compiler flag. If you wish to disable decimal-optimize when using <code>-O</code> , use the <code>-fno-decimal-optimize</code> compiler flag.
<code>index-optimize : [yes/no]</code>	No	Enables optimization of MOVE and IF statements containing references to variables using indexes. As an example, MOVE data-item(1,3) to current-item. Or IF data-item(1,3) < 100 perform inventory-trigger.	Index-optimize is set to yes by default with the use of the <code>-O</code> compiler flag. If you wish to disable index-optimize when using <code>-O</code> , use the <code>-fno-index-optimize</code> compiler flag.
<code>binary-truncate:no</code>	Yes	Enables optimization of mathematical operations where a numeric field is described with a decimal notation, and contains an integer value, with no decimals, (for example, PIC 99V99 VALUE 10.). Applies to the performance of the ADD, SUBTRACT MULTIPLY and DIVIDE verbs.	binary-truncate is set to yes by default. To achieve optimizations from binary-truncate when compiling with <code>-O</code> , set binary-truncate:no.
<code>notrunc:yes</code>	No	Same as binary-truncate: no	Notrunc is set to no by default. To achieve optimizations from notrunc when compiling with <code>-O</code> , set notrunc:yes.

COB_OPTSIZE_FLAG

The environment variable COB_OPTSIZE_FLAG can be used to name flags to be used in the C compilation phase.

- Usage: To cause the `-Os` flag to be used by the “C” Compiler

```
export COB_OPTSIZE_FLAG=-Os
```

The CALL statement

In applications with large numbers of CALL statements, significant performance improvements can be gained by optimizing the performance of the CALL statement. This is an overview of the compiler configuration file options, compiler flags and runtime environment variables that can improve the performance of the CALL statement. In some cases, more details can be found at the documentation point of the flag.

external-link:[function-name]	Compiler Configuration Flag. Causes [function-name] to be declared as an external non-COBOL symbol. Causes CALL “function-name” to generate more efficient code.
-fauto-load-symb	Compiler Flag. Provides additional control, as regards static symbol definition, by causing the static.symb and user.symb files to be automatically loaded with the compiler configuration file. static.symb is provided with the compiler distribution, and should not be changed by the user. Static.symb includes symbols declared by Pro*Cob and Tuxedo.
-fcall-opt	Compiler Flag. Stores the address of a symbol locally in module memory. Enables CALL statement optimization. Programs containing CANCEL statements should not be compiled with -fcall-opt.
module-load-priority:yes COB_LOAD_PRIORITY	Compiler Configuration Flag. Affects resolution of target of CALL statement. Normal sequence is (first) check linked library, and (then) check shared library. This reverses the sequence. Module-load-priority:yes corresponds to the runtime environment variable COB_LOAD_PRIORITY=Y
static-link:[function-name]	Causes [function-name] to be linked statically. Improves the performance of the CALL statement.
-x	Compiler Flag. Generates native executable.
COB_CALL_CASE=xul COB_LOAD_CASE=xul	Runtime environment variables. COB_CALL_CASE and COB_LOAD_CASE runtime environment variables should be used together. Used together, they provide the user with control over how the target of a CALL statement is resolved.

The PERFORM statement

In applications where PERFORM statements execute a very high number of times, significant performance improvements can be seen by optimizing the PERFORM.

-freturn-opt	Optimizes PERFORM return code.
--------------	--------------------------------

Resolving File Names

The process of resolving a file name can be time consuming where files as declared in a

SELECT phrase may have no extension, and full path-name. This performance penalty can be eliminated by using filename mapping.

External-mapping:yes	Allows files declared as EXTERNAL to be resolved using environment variables.
Filename-mapping:yes	Allows file names to be resolved at runtime using environment variables.

Removing debug-oriented compiler flags

Debug-oriented compiler flags have performance penalties. When your code is well-tested, these compiler flags may no longer be needed, and can be removed to achieve better performance.

-fcheckpoint	Enables setting of checkpoints. Program state is saved at checkpoints, and can be reloaded. For more details on the usage of the -fcheckpoint flag,
-debug	Turns on exception checking
-debugdb=<debugDB>	Stores metadata for debugging in SQLite3 database.
-fdebug-exec	Used for debugging of EXEC SQL statements.
Exception-checking (EC-xxx) compiler configuration flags. As an example: EC-SIZE:yes	Enabled with -debug
-fmem-info	Stores memory information, for analysis in the eventual cause of a crash.
-fprofiling	Adds counters to total statistics for reports on where your application is spending the most time.
-fsource-location	Generates source location code, enabling information to be dumped on source location when runtime aborts. Enabled by -g.
-fstack-check	Enables stack checking debug function.
-ftrace -fsimpletrace -ftraceall	The tracing compiler flags. Cause output to be written to an output file during the runtime execution.
-ftrap-unhandled-exception	Provides additional information when runtime aborts.
-g	Causes debugger metadata to be stored in the compiled object file or, if -DebugDB compiler flag is used, in an SQLite3 database.
-G	Produces debugging information, for purposes of debugging programs written in "C".
COB_ERROR_FILE	Used when debugging are set to capture information for debugging purposes.
COB_FILE_TRACE	Causes data to be written to the COB_ERROR_FILE whenever there is a file I/O operation executed.
COB_DUMP	No longer required after your functionality tests have been

	completed. Creates an output file for the memory dump created when a runtime aborts.
--	--

Optimizing performed at installation (Windows)

During the Windows install, the user is asked what version of Visual C compiler they are using. The compiler and runtime are compiled with the indicated version. This provides optimal performance.

Optimizing compiler flags set by default

The COBOL-IT Compiler Suite uses the following optimizations by default. These can be disabled by adding the no- prefix to the compiler flag. As an example, to disable the `-fcmp-opt` compiler flag, use the `-fno-cmp-opt` compiler flag.

<code>-fcmp-opt</code>	The <code>-fcmp-opt</code> compiler flag activates optimizations when comparing literals with variables. To disable, use the <code>-fno-cmp-opt</code> compiler flag.
<code>-ffast-figurative-move</code>	Fast MOVE of figurative constant . To disable, use the <code>-fno-fast-figurative-move</code> compiler flag.
<code>-ffast-op</code>	Fast operation on numeric DISPLAY/COMP-3. To disable, use the <code>-fno-fast-op</code> compiler flag.
<code>-foptimize-move</code>	Optimizes MOVE operations performed by INITIALIZE statement when target fields are USAGE DISPLAY NUMERIC or USAGE NATIONAL. To disable, use the <code>-fno-optimize-move</code> compiler flag.

Guidelines for use of `-preprocess=cmd`

- The `-preprocess=cmd` compiler flag implements the COBOL-IT integrated pre-processor. Preprocessors (like Oracle procob) may now be called by the compiler after the COBOL preprocessing of COPY/REPLACING clauses has been completed. Usage of the `-preprocess` compiler flag allows for use of the `-fdebug-exec` compiler flag, which permits the user to run the original source code (before precompiling) in the debugger.
- Note- When using `-preprocess=cmd` the COBOL-IT Preprocessor ignores INCLUDE statements, as these must be processed by the external preprocessor.
- Usage: `-preprocess=cmd [filename]`
 - [filename] is a COBOL source file that needs to be precompiled before being compiled by the COBOL-IT COBOL compiler. A common case is a COBOL program containing ESQLE statements, allowing it to interact with an Oracle® database. Such a program would need to be precompiled by the Oracle® precompiler procob.
 - cmd is the external preprocessor command that take 2 parameters :The first parameter is the input file name. The second parameter is the output file name. Consider the sample script citprocob.sh, provided in the Linux/Unix distributions:

- `citprocob.sh`

- In Linux/Unix distributions, a sample script called `citprocob.sh` is located in `$COBOLITDIR/bin/`, which executes the command:

```
procob format=TERMINAL iname=$1 oname=$2.
```

With this script, you can execute the command:

```
cobc -preprocess=citprocob.sh testsql.pco
```

 to cause the Oracle precompiler to run, and preprocess a `pco` file, producing an intermediate output file.

Note that while the command file takes two parameters, only one parameter is supplied to the `-preprocess` compiler flag. The second parameter is provided by the compiler `cobc`, as it generates a target file, and supplies it with a name.

- Sample Usage:

- In Linux/Unix:

```
>cobc -conf=oraconf.conf -x -preprocess=citprocob.sh  
procobdemo.pco $ORACLE_HOME/precomp/lib/cobsqlintf.o -L  
$ORACLE_HOME/lib/ -l cIntsh
```

where `citprocob.sh` contains the line:

```
procob format=TERMINAL iname=$1 oname=$2
```

- In Windows:

```
>cobc -conf=myconf.conf -b -preprocess=cobcmakel  
procobdemo.pco -L %ICLIBHOME% -l %SQLLIB_lib%
```

where the following runtime environment variables are set:

```
set ICHOME=C:\COBOL\INSTANTCLIENT_11_2  
set ICLIBHOME=%ICHOME%\sdk\lib\msvc  
set PCBCFG=%ICHOME%\precomp\admin\pcbcfg.cfg  
set PROCOB=%ICHOME%\sdk\procob.exe  
set SQLLIB_lib=orasql11.lib
```

and where `cobcmakel.bat` contains the line:

```
%PROCOB% config=%PCBCFG% ireclen=132 iname=%1 oname=%2
```

Special cases:

To allow the compiled object to be run in the debugger, add the `-g` compiler flag. Note that the use of `-preprocess` provides debugging of original source code, and precompiled code is not displayed in the debugger.

```
cobc -g -conf=myconf.conf -b -preprocess=cobcmakel
```



```
procobdemo.pco -L %ICLIBHOME% -l %SQLLIB_lib%
```

If you prefer debugging original source code, while maintaining the ability to trace generated source code, add the `-fdebug-exec` compiler flag:

```
cobc -g -fdebug-exec -conf=myconf.conf -b
```

```
-preprocess=cobcmake1 procobdemo.pco -L %ICLIBHOME% -l  
%SQLLIB_lib%
```

Compiler -f Flags

`-f` compiler flags describe a context in which the compiler should generate code in a prescribed way. As such, they are used to enable optimizations, to enforce behaviors compliant with other COBOLs, to create thread-safe code, and to enable the use of utilities such as the debugger, tracing, memory dumps, and profiling.

All flags are provided in the form : *-f`flag-name`*

To enable the flag, use: *-f`flag-name`*

To disable the flag, use: *-fno-`flag-name`*

The `-f` convention can also be applied to compiler configuration file entries, enabling compiler configuration file entries to be entered on the command line.

Consider the case of the `align-8:[y/n]` compiler configuration flag as an example:

`align-8:y` is expressed on the command line as follows:

```
cobc -falign-8 sample.cbl
```

`align-8:n` is expressed on the command line as follows:

```
cobc -fno-align-8 sample.cbl
```

`-f` Flags that can be set on the command line are:

-f77-opt

Optimizes the use of integers stored in `USAGE DISPLAY` or `PACKED` fields in level-77 data items. The 77-opt optimizations are enabled by use of the `-O` compiler flag.

-faccept-with-auto

Causes the `WITH AUTO` clause to be assumed by default on a field-level `ACCEPT` statement.

When not compiling with `-facept-with-auto`, the WITH TAB clause is assumed by default on a field-level ACCEPT statement.

-facept-with-update

Causes field-level ACCEPT statements to be interpreted as containing the “WITH UPDATE” clause.

Equivalent to compiler configuration file setting:

accept-with-update: yes

-falign-8

Aligns 01-level and 77-level data on 8 byte boundaries.
Default is 4- byte boundaries.

On HP Itanium based system this flag is always enabled

-fall-external-call

Internal use only. Causes all CALL statements to be considered EXTERNAL.

-fall-external-link

Causes the targets of the CALL statement to all be assumed to be external-links.
This can improve performance at runtime by optimizing the resolution of the CALL statement.

-falloc-unused-linkage

Causes the compiler to allocate static memory for level 01 fields in the Linkage Section that are not used in either a USING clause or an ENTRY clause.

If the `-falloc-unused` linkage compiler flag is not used, and level 01 fields in the Linkage Section are not used in either a USING clause or an ENTRY clause, these fields are initialized to NULL, and no memory is allocated for them.

Note that usage of of a field for which no static memory has been allocated will provoke a Memory Fault. For cases such as described above, where static memory has not been allocated at compile time, it is possible to programmatically allocate static memory to an unused linkage field using the SET [linkage field] to ADDRESS OF [data-pointer] statement, and avoid the Memory Fault condition.

-fas400-like

Causes the LIKE clause to act compatibly with the AS400 implementation of the LIKE clause. The `-fas400-like` compiler flag causes a field declared with the LIKE clause to be described as a PIC X (other field’s byte size).

Equivalent to compiler configuration file setting:

as400-like: yes

-fauto-load-symb

Provides additional control, as regards static symbol definition, by causing the `static.symb` and `user.symb` files to be automatically loaded with the compiler configuration file.

`static.symb` is provided with the compiler distribution, and should not be changed by the user. `Static.symb` includes symbols declared by Pro*Cob and Tuxedo. Note that it is overwritten when the compiler is updated.

`user.symb` is user-definable and may be placed in the `config` directory of the COBOL-IT installation, or in the current directory. If `user.symb` is missing, no error is generated.

Both `static.symb` and `user.symb` files may include `static-link` declarations.

-fautolock

Sets default for SELECT to LOCK MODE IS AUTOMATIC

-fauto-sprwr

Causes SPCRW2 to run automatically when needed before any `-pre-process` script (default).

-fbdb

Activates the usage of Oracle Berkeley DB isam files

-fbinary-byteorder-big-endian

Sets binary-byteorder to big-endian.

Corresponds to compiler configuration flag:
binary-byteorder: big-endian

-fbinary-byte-order-native

Sets binary-byteorder to native.

Corresponds to compiler configuration flag:
binary-byteorder: native

-fbin-opt

Enables the use of CPU integers when manipulating USAGE COMP and USAGE COMP-5 data elements. The `-bin-opt` optimizations are enabled by use of the `-O` compiler flag.

-fbin-opt-strict

Causes `-fbin-opt` binary operation optimization to be strictly respected.

Corresponds to compiler configuration flag:
bin-opt-strict:yes

-fcall-comp5-as-comp

The `-fcall-comp5-as-comp` compiler flag affects the behavior of the CALL statement. On little-endian platform (intel Linux, Windows) when a call USING clause contains a literal, the `-fcall-comp5-as-comp` compiler flag causes the literal to be copied as a COMPUTATIONAL value, rather than as a COMP-5 value.

Example: In the statement:

CALL "subprogram" USING 1234.

The literal 1234 is passed as a COMP value when using `-fcall-comp5-as-comp`. Otherwise, the literal 1234 is passed as a COMP-5 value.

Equivalent to: `call-comp5-as-comp:yes` in config file

-fcall-lowercase

Affects the handling of literals that are the target of a CALL statement. As an example, consider the literal "MyProg" in the statement: CALL "MyProg". In this case, the `-fcall-lowercase` compiler flag causes all of the characters in the literal "MyProg" to be converted to lowercase.

-fcall-opt

Enables CALL statement optimization Programs containing CANCEL statements should not be compiled with `-fcall-opt`.

-fcall-uppercase

Affects the handling of literals that are the target of a CALL statement. As an example, consider the literal "MyProg" in the statement: CALL "MyProg". In this case, the `-fcall-uppercase` compiler flag causes all of the characters in the literal "MyProg" to be converted to uppercase.

-fcarealia-sign

Use CA Realia sign coding for Usage Display

Digit	Sign Digit Character for:					
	Positively-signed values			Negatively-signed values		
	-fsign-ascii	-fsign-ebcdic	-fcarealia-sign	-fsign-ascii	-fsign-ebcdic	-fcarealia-sign
0	0(30)	{(7B)	0(30)	p(70)	}(7D)	0(30)
1	1(31)	A(41)	1(31)	q(71)	J(4A)	!(21)
2	2(32)	B(42)	2(32)	r(72)	K(4B)	"(22)

3	3(33)	C(43)	3(33)	s(73)	L(4C)	#(23)
4	4(34)	D(44)	4(34)	t(74)	M(4D)	\$(24)
5	5(35)	E(45)	5(35)	u(75)	N(4E)	%(25)
6	6(36)	F(46)	6(36)	v(76)	O(4F)	&(26)
7	7(37)	G(47)	7(37)	w(77)	P(50)	'(27)
8	8(38)	H(48)	8(38)	x(78)	Q(51)	((28)
9	9(39)	I(49)	9(39)	y(79)	R(52))(29)

-fC-cmd-line

When used with `-x`, causes the program to receive command line parameters as though they were given in C. In this case, command line parameters are read as they would be through a “C” interface. For example: `(int argc , char **argv)`

-fC-data-init

(Internal use only)

Controls if the C data structure created by the compiler is initialized in the source (at compilation time) or at runtime. This should not be changed.

-fcheckpoint

Enables setting of checkpoints. Program state is saved at checkpoints, and can be reloaded. For more details on the usage of the `-fcheckpoint` flag, see the **Guidelines for use of Checkpoints** below.

-fcics

Generates CICS-compliant code.

-fcmp-inline

Causes comparisons to be inlined in the C code instead of through calls to the runtime library when possible.

-fcmp-opt

Activates optimizations when comparing literals with variables.

The `-fcmp-opt` compiler flag is set by default. Use `-fno-cmp-opt` to disable the functionality.

The `-fcmp-opt` compiler flag corresponds to the compiler configuration flag :
`cmp-opt :yes`

-fcobol-lines

When compiling with `gcc`, line directives corresponding to COBOL source code line numbers are added to the “C” source code.

-fcompat-display-to-int

Provides compatibility with older versions of COBOL-IT Compiler Suite (prior to version 3.10.5) as regards display to int functionality.

-fcompute-ibm

Causes arithmetic expressions (like $a+B*c$) in COMPUTE statements, and comparisons to use IBM COBOL defined rules for determining the number of decimals used in intermediate results.

-fno-compute-ibm causes the maximum number of decimals (machine-dependent, up to 37) to be used in intermediate results of arithmetic expressions in COMPUTE statements and comparisons.

For details, please consult IBM documentation.

-fcompute-ibm-trunc

When using *-fcompute-ibm* compiler flag, or when *compute-ibm* compiler configuration flag is set to yes, causes intermediate results to be truncated (default).

Corresponds to compiler configuration flag:

compute-ibm-trunc:yes

-fcontinuation-line

Allows a hyphen in column 7, with no following text, to be recognized as not being a continuation line. When compiling with *-fcontinuation-line*, a hyphen in column 7, with no following text in quotes is not recognized as a continuation line.

Note: A hyphen in column 7, with following text, in quotes, is always recognized as a continuation line.

For rules on the handling of continuation lines, see Line Continuations in the COBOL-IT COBOL Reference Manual.

-fcopy-default-leading

The *-fcopy-default-leading* compiler flag affect the behavior of the COPY REPLACING statement.

When using the *-fcopy-default-leading* compiler flag, and when using the `==xxx==` notation in a COPY REPLACING statement, the LEADING phrase is assumed by default. The LEADING phrase indicates that only the LEADING characters identified will be replaced if they match text in the copy file.

-fcopy-exec-replace

The *-fcopy-exec-replace* compiler flag affects the behavior of the COPY REPLACING statement. When using the *-fcopy-exec-replace* compiler flag, and when a COPY REPLACING `== xxx ==` statement is performed, text inside EXEC / END-EXEC blocks are also replaced if applicable.

-fcopy-mark

Adds mark for begin/end of COPY In listing and preprocessed file

Note: The copy marks are:

```
*++SCOPY .\copy/sample.CPY ( beginning of COPY file )  
[ COPY file is listed here ]  
*--SCOPY .\copy/sample.CPY ( end of COPY file )
```

-fcopy-partial-replace

The `-fcopy-partial-replace` compiler flag increases the compatibility of the COPY REPLACING behavior with Micro Focus compiler behaviors.

When a pattern like `COPY FIC1 REPLACING == WJXX- == BY == WJ03- ==` is processed :
If this flag is on, the preprocessor uses a partial replacement as defined by MF and ANSI2002 standard.

If it is off (the default) the IBM mainframe and ANSI85 standard is used.

The `-fcopy-partial-replace` compiler flag corresponds to the compiler configuration flag :
`copy-partial-replace :yes`

-fctree

Activates the usage of C-tree isam files

-fctree-field-numbering

Causes the CTREE XDD generator to generate a prefix F <field-number> before field names.
Use with `-fgen-xdd` compiler flag.

-fctree-no-full-qualification

Affects the behavior of the `-fgen-xdd` compiler flag, causing it to not generate the fully qualified data names in the XDD description of the file. When using the `-fctree-no-full-qualification` compiler flag with the `-fgen-xdd` compiler flag,

the field name generated is :
`xxx_<Field_Name>`

Where:

`xxx` is a unique number (position in the structure),
`<Field_Name>` is the name of the data field, without any other prefix or suffix

Use with `-fgen-xdd` compiler flag.

-fcurdir-include

Causes COPY files to first be searched for in the current directory, before locations described with the `-I <Path>`, or with environment variables.

The COPY search is performed for files with default extensions, and with extensions described with the `-ext` compiler flag. This is a default behavior.

The `-fno-curdir-include` compiler flag causes the search for a COPY file to not search for COPY file in the current directory, unless that directory is named by a `-I` compiler flag, or by a `COB_COPY_DIR`, or `COBCPY` environment variable.

-fdebugdb

The `-fdebugdb` compiler flag, when used with `-g`, store all debugging information into a file name `<modulename>.dbd`. The runtime checks for the `<debugdb>` file using the name given at compilation time. If not found, the runtime will then check for the filename in the `COB_LIBRARY_PATH` location.

This is different from `debugdb=<filename>` where you have to specify a unique Debug db for the whole project

Equivalent to: `debugdb:yes` in the compiler configuration file.

-fdebug-exec

Affects the tracing of Exec statements when debugging code that has been compiled with the integrated pre-processor (`-preprocess`). When using the Integrated Preprocessor Interface, the default behavior of the debugger is to `-not-` trace (display) the code generated by the external preprocessor. Only the original source EXEC statements are shown. The `-fdebug-exec` compiler flag enables the tracing (debugging) of the generated code.

-fdebugging-line

Enables support for debugging lines. (Source lines that contain 'D' in indicator column)

-fdebug-parser

Allows for the debugging of the parser. (maintainer use only)

-fdecimal-optimize

Optimizes the conversion from DISPLAY/COMP-3 to binary values in COMPUTE statements. When several COMPUTE statement are in the same paragraph, the compiler will minimize the conversions from DISPLAY/COMP-3 to binary values for fields that are used (and not modified) in different statements in the same paragraph. `-fdecimal-optimize` is enabled by using either the `-O` or `-O2` compiler flags.

Equivalent to: `decimal-optimize:yes` in config file

-fdisam

Activates the usage of the DISAM indexed file engine.

-fdisplay-dos

Causes DISPLAY statements to use CR/LF.

Corresponds to configuration file setting:
display-dos: yes

-fdisplay-ibm

Affects the output of the DISPLAY Statement for numeric fields to be more compatible with IBM mainframe.

-fdiv-check

Enables the checking of divide operations when binary optimizations are turned on with the use of the `-fbin-opt`, or `-O` compiler flags. The effect is to cause divide-by-0 operations to generate an exception.

-febcdic-charset

The `-febcdic-charset` compiler flag requires a dedicated license.

Causes the COBOL-IT Compiler and Runtime to store and manage data in the EBCDIC encoding format. Source code is stored in ASCII format.

The `-febcdic-charset` compiler flag causes modules to be created that apply the EBCDIC character set to file operations, and internal data storage. When using the `-febcdic-charset` compiler flag, all of the programs in a runtime unit must be compiled with the `-febcdic-charset` compiler flag. If a main entry program that is not compiled with the `-febcdic-charset` compiler flag CALLs a program that is compiled with the `-febcdic-charset` compiler flag, or conversely, if a main entry program compiled with the `-febcdic-charset` compiler flag CALLs a program that is not compiled with the `-febcdic-charset` compiler flag, the CALL will fail, and the COBOL-IT runtime will abort.

-femulate-vms

Causes spaces to be stripped from filenames and adds suffix '.DAT' if needed.

Corresponds to compiler configuration file setting:
emulate-vms: yes

-fexclusivelock

Causes all files with no LOCK MODE clause in their SELECT statement to be declared Implicitly as LOCK MODE is EXCLUSIVE. For details on other `-f` compiler flags related to the treatment of LOCK MODE, see Guidelines for modifying default handling of the LOCK MODE.

-fexec-check

Used with `-fsyntax-only`, checks the EXEC SQL/CICS/DLI syntax

-fexpand-exec-copy

The `-fexpand-exec-copy` compiler flag causes the compiler to expand COBOL COPY statements inside EXEC ... END-EXEC blocks. This applies to both EXEC SQL and EXEC CICS blocks.

Equivalent to: expand-exec-copy:yes in config file

-fexpand-sql-include

Used with -E, expands 'EXEC SQL INCLUDE <File name> END-EXEC' in the -E output.

-ffast-figurative-move

Fast MOVE of figurative constant (default)

-ffast-op

Enables the runtime to use faster operations when manipulating data items declared as USAGE DISPLAY or USAGE COMP-3.

-ffcdreg

The -ffcdreg compiler flag allows a user of an EXTFH compliant data source to directly read and write the File Control Description (FCD) through which information passes to and from an EXTFH-compliant data source. When the -ffcdreg compiler flag is used the compiler will generate an error if -use-extfh is not used.

As background, EXTFH makes use of a standardized File Control Description (FCD), through which information passes to and from the EXTFH-compliant data source.

An FCD is created for each file that is mapped to an EXTFH-compliant data source.

It can be useful inside a program to directly read and write the FCD. The FCDREG compiler directive was developed for this purpose, and the COBOL-IT implementation of this functionality is the -ffcdreg compiler flag. When you compile with the -ffcdreg compiler flag, a register is created for each [filename] which is named "FH--FCD of [filename]". Note that there are two hyphens in the name "FH--FCD". By describing the FCD structure, and positioning the beginning of the structure at the address of "FH--FCD of [filename]", individual elements within the structure can be read and written.

Note- The FCD structure is described in a copy file called XFHFCD.CPY, which is included in the \$COBOLITDIR\copy directory in Windows, and the \$COBOLITDIR/share/config directory on UNIX/Linux-based systems.

For example:

1- Include a reference to the FCD in your Linkage Section, as follows:

```
LINKAGE SECTION.  
01 FCD.  
COPY "XFHFCD.CPY".
```

2- Sync the address of FCD with the address of FH--FCD OF FIL1.

PROCEDURE DIVISION.

...

SET ADDRESS OF FCD TO ADDRESS OF FH--FCD OF FIL1.

3- After performing the SET statement above, the fields in XFHFCD.CPY can be read and written.

-ffdclear

Causes the record to be INITIALIZED after each WRITE.

-ffile-auto-external

Default: On

This functionality may be disabled using the compiler flag '-fno-file-auto-external'. However, when disabling this functionality, be aware, that if you have separate programs sharing the same EXTERNAL file that also have file-var fields, then changes made between the programs will not automatically be shared.

The `-ffile-auto-external` compiler flag affects the way that the compiler treats variables describing file-names for files described as EXTERNAL.

When a file is declared as EXTERNAL, if the file-name is indicated as a variable name, in an ASSIGN DYNAMIC [file-var] clause, then file-var should be declared as EXTERNAL. Note that variables declared as EXTERNAL must be declared as level 01 or level 77.

If [file-var] is not declared as EXTERNAL, then the default behavior of the COBOL-IT Compiler is to implicitly declare an external variable name, and assign it a name derived from the FD named in the SELECT clause.

The convention used is as follows:

Consider the statement:

```
SELECT myfile ASSIGN DYNAMIC file-var...
```

```
...
```

```
77 file-var  pic x(8) value "customer".
```

In this case, file-var is not declared as EXTERNAL, so COBOL-IT issues the following warning:

'file-var' declared implicitly EXTERNAL AS 'FE_file_myfile_ASSIGN' (-fno-file-auto-external to disable).

Creating the implicit name based on the file name guarantees that the programmer will be able to give different names to [file-var] in different programs, and that they will nonetheless share the same value.

If file-var is declared explicitly as EXTERNAL, then this condition does not apply.

-ffold-copy-lower

Folds COPY file names to lower case

-ffold-copy-upper

Folds COPY file names to upper case

-ffp-opt

Causes COMP-2 operations to be inlined in C, and maximizes the use of the CPU Floating Point unit.

-ffree-thread-safe-data

When used with `-thread-safe`, causes the data in a module to be freed after a CANCEL event that is not a FULL-CANCEL.

Corresponds to compiler configuration file setting:
`free-thread-safe-data: yes`

-ffunctions-all

Allows use of intrinsic functions without the FUNCTION keyword

-ffunctions-all-intrinsic

Some functions do not require FUNCTION keyword

-fgcc

Generates gcc-compliant C code. The `-fgcc` compiler flag is enabled when `COB_CC=gcc`.

Default : off for all platforms except Linux

Default: on for Linux platforms

-fgcc-bug

When using a gcc compiler on very large source files, the gcc compiler could enter an infinite loop. This bug is avoided by using the `-gcc-bug` compiler flag.

-fgcc-goto

Generates gcc-computed goto code. The `-fgcc-goto` compiler flag is enabled when using the `-fgcc` compiler flag, or when `COB_CC=gcc`

-fgcc-O-bug

When using `-O`, some versions of gcc generate incorrect code. This bug is avoided by using the `-gcc-O-bug` compiler flag.

-fgcos-mode

Causes the compiler to more closely emulate GCOS operations.

-fgen-xdd

Generate c-TreeACE .xdd file

-fglobal-typedef

Causes TYPEDEFS to be GLOBAL for all nested programs. If not set, TYPEDEFS are local to the current program.

-fibm-listing-macro

Enables IBM listing extensions (TITLE, SKIP1/2/3, EJECT ...) (default)

-fibm-mainframe

Causes the compiler and runtime to operate in an IBM Mainframe compatible mode.

-fibm-sync

Applies SYNC attribute to group item if first elementary field is described with the SYNC attribute. (default).

To turn off this behavior, use the `-fno-ibm-sync` compiler flag.

The `-fno-ibm-sync` compiler flag does not cause the SYNC attribute to be applied to a group item if the first elementary field in the group item is described with the SYNC attribute.

-fimplicit-init

Initializes the COBOL runtime system at runtime start-up.

-finclude-main

Causes main symbol to be included in module object when compiled with `-c`.

Previous behavior

Previously, the compiler generated a wrapper C module `<module-name>_main.c` with the main symbol that called the first module (`.o` or `.cob`) given as a parameter at the `-x` command line.

The C module was compiled into a .o and stored in a static “.a” library. Then this library was appended at link time.

New behavior

When compiling with the compiler flag combination **-c -finclude-main**, the compiler inserts the main symbol into the object of the first COBOL module named on the command line. When compiling with the compiler flag **-x**, and at least one COBOL module has been compiled, the **-finclude main** compiler flag is implied, and no static “.a” library is produced. If no COBOL modules are compiled, the **Previous Behavior** is used.

Example :

```
>cobc -c -finclude-main myprog.cob mysub.cob
```

You may the link the program with:

```
>cobc -x -flink-only myprog.o mysub.o
```

This will generate a myprog executable. No static “.a” library is produced, and the main symbol of myprog.o is used.

Example:

```
>cobc -x myprog.cob mysub.cob
```

This will generate a myprog executable. No static “.a” library is produced.

-fincomplete-subscript

Affects the behavior of MOVEs to table items.

Consider a data item declared as 01 TABLE OCCURS 10 PIC X.

Causes the phrase MOVE SPACE TO TABLE to be equivalent to MOVE ALL SPACE TO TABLE.

-findex-optimize

Improves performance where indexes in tables are evaluated and USAGE DISPLAY fields are used as indexes. In these cases, the index values are cached in a C integer field to improve performance.

As an example, consider the usage of the follow code:

```
...  
01 IxD A PIC 999 USAGE DISPLAY.  
01 IxD B PIC 999 USAGE DISPLAY.  
...  
MOVE FLD-ARRAY(IxD A, IxD B) TO ...  
MOVE FLD-ARRAY(IxD A, IxD B) TO ...  
IF (FLD-ARRAY(IxD A, IxD B) ...
```

Use of the **-findex-optimize** compiler flag will benefit the performance of these MOVE and IF statements by keeping the actual value of the index in a binary C cache, thus avoiding conversion

from DISPLAY (or COMP-3) to a binary value each time the index is evaluated in a statement. The `-findex-optimize` flag is enabled by use of the `-O` or `-O2` compiler flags.

-finitialize-fd

Causes records declared in the FD section to be initialized when the program is initially loaded in memory.

-finitialize-opt

Using the `-finitialize-opt` compiler flag optimizes the implementation of the initial field initialization at runtime startup and the execution of the INITIALIZE statement by grouping field initializations wherever possible.

-fkeep-copy-statement

In listing and preprocessed file, keep COPY statements.

-fkeep-org-src-line

For use with the integrated pre-processor (`-preprocess`). Causes errors to be reported on the original source line.

-fkeep-unused

Causes memory to be allocated for the field tree of level-01 and level-77 data items that are declared which contain sub-fields and in which none of these sub-fields is used.

The `-fno-keep-unused` compiler flag causes memory to not be allocated for the field tree of level-01 and level-77 data items that are declared which contain sub-fields and in which none of these sub-fields is used.

-fline-seq-dos

Affects the writing of the record delimiter at the end of each record in a line sequential file. When compiling with `-fline-seq-dos`, the record delimiter is set to `<CR><LF>`.

-flink-only

Causes the `main()` symbol to not be generated, when used with `-x`. For use when the program entry point (`main`) is provided by an external object or library.

-flisting-sources

Informs the compiler that source is the result of program listing option (`-t <file>`).

-floosy-comment

Causes the compiler to allow a `*` in column 8 to be used to mark a comment.

-fls-expand-tab

Causes the READ of a LINE SEQUENTIAL file to expand the TAB character to 8 spaces (default)

-fmain

Generates main() symbol when used with -x (default)

-fmain-as-object

Generates main() symbol as object not in library (unix only) (default)

-fmainframe-vb

The `-fmainframe-vb` compiler flag causes WRITES and READS of Variable Blocked files to assume formats compatible with the Mainframe Z/OS COBOL Format.

-fmakesyn-patch-preprocess

Causes the makesyn compiler flag to change the output of a pre-processed file.

-fmanuallock

Causes all files with no LOCK MODE clause in their SELECT statement to be declared Implicitly as LOCK MODE is MANUAL unless a SHARING clause in the SELECT statement or in the OPEN statement indicates otherwise. For details on other `-f` compiler flags related to the treatment of LOCK MODE, see Guidelines for modifying default handling of the LOCK MODE.

-fmem-info

Enables Dump of Working-Storage when runtime aborts. The `-fmem-info` compiler flag functionality is enabled by the `-g` compiler flag, and by the `-debug` compiler flag.

-fmfcomment

Treats lines with '*' or '/' in column 1 as comments.

-fmf-compat-parser

Increases compatibility of syntax parser with the Micro Focus syntax parser. (default).

Causes COBOL-IT to match certain Micro Focus behaviors. These include :

Parsing of line continuation characters

Relaxed syntax check on RECORD CONTAINS phrase in the FD

Allowing level-66 and level-88 data names to have the same name as a paragraph or section.

-fmf-ctrl-escaped-parser

Syntax parser is MF compatible with control character escaped by 0 (default).

-fmf-file-optional

Affects the file-status codes returned on files declared as `OPTIONAL` and `OPEN` in `EXTEND`.

The `-fmf-file-optional` compiler flag causes files declared as `OPTIONAL` and `OPEN` in `EXTEND` to return file-status code "05" if the file is created and file-status code "00" if the file exists. The `-fmf-file-optional` compiler flag improves consistency with Micro Focus behaviors.

Alternatively, the runtime returns file-status code 00 in either case.

The `-fmf-file-optional` compiler flag corresponds to setting `mf-file-optional:yes` in the compiler configuration file.

-fmf-gnt

Causes shared objects generated by the compiler to be created with the `.gnt` extension.

Note that the generated object IS NOT compatible with the `.gnt` objects produced by Micro Focus. This option is only used to reduce change in existing compilation scripts by causing object code to be generated with the same extensions.

-fmf-hostnumcompare

Provides compatibility with Micro Focus in cases where the `HOST-NUMCOMPARE` directive is used. The `-fmf-hostnumcompare` compiler flag affects comparisons of `USAGE DISPLAY` numeric data items when one of the numeric data items in the comparison contain non-numeric data.

When compiling with `-fmf-hostnumcompare`, the field containing numeric data is redefined as an alphanumeric item of the same length, and this redefined data item is compared with the non-numeric value of the other numeric data item.

When not compiling with `-fmf-hostnumcompare` (the default), the contents of the field containing numeric data are moved to an intermediate alphanumeric data item that is the same size as the field containing nonnumeric data before the comparison is performed. The content of this intermediate alphanumeric item is then compared to the non-numeric value of the other numeric data item.

-fmf-int

Causes shared objects generated by the compiler to be created with the `.int` extension.

Note that the generated object IS NOT compatible with the `.int` objects produced by Micro Focus. This option is only used to reduce change in existing compilation scripts by causing object code to be generated with the same extensions.

-fmf-relativefile

The `-fmf-relativefile` compiler flag causes the runtime to assume the Micro Focus format for relative files for both `READ` and `WRITE` operations.

When using the `-fmf-relativefile` compiler flag, the end-of-record marker for relative files is consistent with the setting of the compiler configuration flag `line-seq-dos`.

When `line-seq-dos:yes`, the end of record setting is `CR/LF`

When `line-seq-dos:no`, the end of record setting is `LF`

-fmodule-name-entry

Generates source module as alternate entry (default)

-fmodule-uppercase

Causes the output file name to be created in upper-case, when used with the `-m` compiler flag.

-fmove-all-edited

Causes MOVE ALL “X” to an edited field to take care of the PICTURE.

Corresponds to compiler configuration file setting:

move-all-edited: yes

-fno-cbl-error-proc

Prevents the execution of CBL_ERROR_PROC.

Corresponds to compiler configuration file setting:

no-cbl-error-proc: yes

-fno-realpath

Causes file names to NOT be extended to a fully qualified path.

By default, when processing file names, the compiler retrieves the fully qualified path (from the root) and processes the compilation using that extended name. That full name is also stored as the source file name for debugging purposes.

-fnostrip

Causes objects and object and executable files to NOT be stripped.

Stripping an object or an executable is the action of removing system level debugging information

-fnotrunc

Causes truncation of binary fields to NOT be made according to the PICTURE clause while performing intermediate computations.

-fnull-param

Causes an extra NULL pointers to be passed as the last argument on CALL statements.

-fnumeric-compare

Causes the comparison of a numeric field with a PIC X field to interpret the value of the PIC X field using its numeric value.

-fnumval-validate

The `-fnumval` compiler flag validates argument 1 of the NUMVAL function.

-fobj-cit

The `-fobj-cit` compiler flag causes compiled object to be generated with a `.cit` extension instead of `.dll` (windows) or `.so` (unix/linux). The COBOL-IT runtime recognizes the `.cit` extension as an executable extension.

In a runtime session in which some of the programs have been compiled with the `-fobj-cit` compiler flag, the runtime, when resolving the names of programs/sub-programs, will search first for programs/sub-programs with the `.cit` extension. Subsequently, the runtime will search for programs/sub-programs with the `.so/.dll` extension.

In a runtime session in which no programs have been compiled with the `-fobj-cit` compiler flag, the runtime will search first for programs/sub-programs with the `.so/.dll` extension.

-fodo-slide

Affects data items that appear after a variable-length table in the same record; that is, after an item with an OCCURS DEPENDING clause, but not subordinate to it.

If the `odo-slide` compiler flag is set, these items always immediately follow the table, whatever the current size of the table. Note that the internal addresses of these data items change as the table's size changes.

If the `odo-slide` compiler flag is not set, these items have fixed addresses, and begin after the end of the space allocated for the table at its maximum length.

Note : The `mf.conf` configuration file, which contains compiler configuration flags designed to match Micro Focus default behaviours, includes the setting `odo-slide : no`.

-foptimize-move

Causes MOVE operations to be optimized by `-fmem-info` where the source and target fields have identical declarations. (Default is on).

-foptimize-move-call

Causes MOVE operations to be optimized by pre-selecting the internal runtime library routines used for the MOVE when possible.

-foptional-file

Causes all SELECT statements that do not specify OPTIONAL or NOT OPTIONAL to be considered OPTIONAL.

If the compiler flag is not present or if `-fno-optional-file` is specified, NOT OPTIONAL is the default value.

The implementation of `-foptional-file` is designed to be compatible with the OPTIONAL-FILE compiler directive supported by other COBOL compilers.

-fperform-osvs

Enhances compatibility with OSVS COBOL PERFORM statements

The \$SET perform type settings of COB370, ENTCOBOL, OSVS and VSC2 are emulated with the -fperform-osvs compiler flag.

The exit point of any currently executing perform is recognized if reached.

PERFORM statements with the same exit point can be nested to a depth of two (one inner and one outer). If they are nested deeper, they do not return correctly. The end of a section is regarded as a separate point from the end of its last paragraph.

The example below is included to illustrate a possible consequence of using the -fperform-osvs compiler flag, where an infinite loop results that would otherwise be avoided.

```

IDENTIFICATION DIVISION.
PROGRAM-ID.      TEST-PERFORM.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 SW            PIC X VALUE '1'.
01 ABORT-PRG    PIC X(3) VALUE 'NO'.
PROCEDURE DIVISION.
BEGIN.
    DISPLAY "BEGIN".
    PERFORM A THRU A-EX UNTIL ABORT-PRG = 'YES'.
    DISPLAY "END".
    STOP RUN.

C.
    DISPLAY "I AM IN C".

A.
    IF SW = '1'
        PERFORM D THRU D-EX
    ELSE
        MOVE 'YES' TO ABORT-PRG.

A-EX.
    EXIT.

B.
    DISPLAY "I AM IN B".
    MOVE '0' TO SW.
    GOTO C.

D.
    IF SW = '1'
        GOTO B.

D-EX.
    EXIT.
  
```

When not using the -fperform-osvs compiler flag, , the source above produces the following result :

```

BEGIN

I AM IN B

I AM IN C

END
  
```

-fprepro_cut_line

When preparing a file for preprocessing, cuts line to 72 columns (default).

Corresponds to compiler configuration file setting:

prepro_cut_line: yes

-fprinter-crlf

Files declared with ASSIGN TO PRINTER file names are generated with compatibility for DOS printers. This will change the End Of Record to CR/LF (instead of LF)

-fprofiling

Generates paragraph profiling code. The output produced by the profiler includes separate Counts for CPU and real elapsed times. For more details on using COBOL-IT's built-in Profiler, see Guidelines for use of Profiler below.

-fprotect-linkage

Generates code at the entry point of a program containing a USING xxx clause.

This allows for the passing of parameters that are NULL pointers. In these cases, where NULL pointers are passed, the compiler creates a "fake" field of the same definition in WORKING-STORAGE, and substitutes it as a reference for the parameter. Doing this will avoid a SIGVEC error if NULL pointers passed through linkage are targets of a READ or WRITE statement.

-fraw-by-value

CALL BY VALUE [PIC X Fld] does not convert [PIC X Fld] to numeric COMP-5 (default).

-fraw-pic9-display (Internal use only)

DISPLAY PIC 9(X) (no sign, no decimal) as it is in memory.

-fread-into-copy

Causes a READ INTO statement to COPY data rather than perform a MOVE.

Corresponds to compiler configuration file setting:

read-into-copy: yes

-fready-trace

The -fready-trace compiler flag enables paragraph tracing between READY TRACE and RESET TRACE procedural COBOL statements. In the interval between the READY TRACE and RESET TRACE statements, paragraph tracing output is written to the console in the format:

PROGRAM-ID: [program-id]: [paragraph name]

-frecmode-f

Causes all unspecified RECORDING MODE clauses to be interpreted as RECORDING MODE F.

-frecmode-v

Causes all unspecified RECORDING MODE clauses to be interpreted as RECORDING MODE V.

-frecord-depending-iso

Causes a RECORD DEPENDING ON <FIELD> clause to be handled in an ISO-compatible manner. More specifically, the `-frecord-depending-iso` compiler flag causes files declared with a RECORD DEPENDING ON <FIELD> clause, without any FROM or TO value, to assume a FROM and TO value of the maximum record size.

The `-frecord-depending-iso` compiler flag corresponds to the `record-depending-iso:yes` compiler configuration flag.

-fregion0

Causes the program to always switch to region 0 when executing. The compiler flag `-fregion0` lets you specify that the module will always execute in region 0 even if called from another region. When called from another region, the module will switch to region 0 on entry and switch back to the calling region at exit.

-frelativefile-bigendian

Causes the record header of relative files to be stored in BigEndian format.

-freplace-additive

Allows for the use of the REPLACE ADD verb, which has the effect of nesting a REPLACE statement inside an existing REPLACE statement. Nested REPLACE statements are executed before outer REPLACE statements in COBOL-IT's precompile phase. Note that a REPLACE stack can be cleared with the REPLACE OFF statement.

-freturn-opt

Generates optimized PERFORM return code. The `-freturn-opt` compiler flag is ignored when using the `-fgcc` compiler flag.

-fround-fp

Controls the way COMP-1 or COMP-2 are "moved" into non-COMP-1 or COMP-2 target fields when the target field has fewer decimal places than the source field.

If the `-fround-fp` compiler flag is used, the value is rounded to the number of decimal of the target field. Otherwise, the value is truncated.

-frw-after-preprocess

Causes SPCRW2 to be run after the `-preprocess` script. By default SPCRW2 is run before the `-preprocess` script.

-frw-mode-nopf

Is equivalent to setting MODE NOPF for a Report Section report. MODE NOPF causes the Report Writer to emulate an external print driver that does not generate printer control characters, like FF. When using the `-frw-mode-nopf` compiler flag, the report file is written as a standard LINE SEQUENTIAL file.

-frw-mode-nopf-dos

Is equivalent to setting MODE NOPF for a Report Section report. MODE NOPF causes the Report Writer to emulate an external print driver that does not generate printer control characters, like FF. When using the -frw-mode-nopf-dos compiler flag, the report file is generated with fixed-length lines that are padded with SPACES and use the CR/LF record delimiter.

-fsafe-linkage

Generates code at the entry point of a program containing a USING xxx clause. This allows for the omission of parameters. Doing this will avoid a SIGVEC being returned by the debugger when all linkage parameters are not provided.

-fsequential-line

Causes all non-qualified SEQUENTIAL files to be declared as LINE SEQUENTIAL. Files declared as RECORD SEQUENTIAL are not affected.

-fshare-all-autolock

Causes all files with a SHARE WITH ALL clause in their SELECT statement to be declared implicitly as LOCK MODE IS AUTOMATIC. For details on other -f compiler flags related to the treatment of LOCK MODE, see Guidelines for modifying default handling of the LOCK MODE.

-fshare-all-default

The -fshare-all-default compiler flag causes all files to be declared implicitly as SHARE WITH ALL.

-fshare-all-default *may be* used in conjunction with :

-fshare-all-autolock Causes all files with a SHARE WITH ALL clause in their SELECT statement to be declared implicitly as LOCK MODE IS AUTOMATIC.

-fshare-all-manulock Causes all files with a SHARE WITH ALL clause in their SELECT statement to be declared implicitly as LOCK MODE IS MANUAL.

-fshare-all-default *should not be* used in conjunction with:

-fexclusivelock Causes all files with no LOCK MODE clause in their SELECT statement to be declared implicitly as LOCK MODE is EXCLUSIVE.

-fmanuallock Causes all files with no LOCK MODE clause in their SELECT statement to be declared implicitly as LOCK MODE is MANUAL unless a SHARING clause in the SELECT statement or in the OPEN statement indicates otherwise.

For details on other -f compiler flags related to the treatment of LOCK MODE, see Guidelines for modifying default handling of the LOCK MODE.

-fshare-all-manulock

Causes all files with a SHARE WITH ALL clause in their SELECT statement to be declared implicitly as LOCK MODE IS MANUAL. For details on other -f compiler flags related to the treatment of LOCK MODE, see Guidelines for modifying default handling of the LOCK MODE.

-fsign-ascii

Corresponds to the SIGN "ASCII" directive. Causes numeric DISPLAY items that include signs to be interpreted according to the ASCII sign convention. (default on ASCII machines)

Digit	Sign Digit Character for:					
	Positively-signed values			Negatively-signed values		
	-fsign-ascii	-fsign-ebcdic	-fcarealia-sign	-fsign-ascii	-fsign-ebcdic	-fcarealia-sign
0	0(30)	{(7B)	0(30)	p(70)	}(7D)	0(30)
1	1(31)	A(41)	1(31)	q(71)	J(4A)	!(21)
2	2(32)	B(42)	2(32)	r(72)	K(4B)	"(22)
3	3(33)	C(43)	3(33)	s(73)	L(4C)	#(23)
4	4(34)	D(44)	4(34)	t(74)	M(4D)	\$(24)
5	5(35)	E(45)	5(35)	u(75)	N(4E)	%(25)
6	6(36)	F(46)	6(36)	v(76)	O(4F)	&(26)
7	7(37)	G(47)	7(37)	w(77)	P(50)	'(27)
8	8(38)	H(48)	8(38)	x(78)	Q(51)	((28)
9	9(39)	I(49)	9(39)	y(79)	R(52))(29)

-fsign-ebcdic

Corresponds to the SIGN "EBCDIC" directive. Causes numeric DISPLAY items that include signs to be interpreted according to the EBCDIC sign convention. (default on EBCDIC machines)

Digit	Sign Digit Character for:					
	Positively-signed values			Negatively-signed values		
	-fsign-ascii	-fsign-ebcdic	-fcarealia-sign	-fsign-ascii	-fsign-ebcdic	-fcarealia-sign
0	0(30)	{(7B)	0(30)	p(70)	}(7D)	0(30)
1	1(31)	A(41)	1(31)	q(71)	J(4A)	!(21)
2	2(32)	B(42)	2(32)	r(72)	K(4B)	"(22)
3	3(33)	C(43)	3(33)	s(73)	L(4C)	#(23)
4	4(34)	D(44)	4(34)	t(74)	M(4D)	\$(24)

5	5(35)	E(45)	5(35)	u(75)	N(4E)	%(25)
6	6(36)	F(46)	6(36)	v(76)	O(4F)	&(26)
7	7(37)	G(47)	7(37)	w(77)	P(50)	'(27)
8	8(38)	H(48)	8(38)	x(78)	Q(51)	((28)
9	9(39)	I(49)	9(39)	y(79)	R(52))(29)

-fsign-leading

Makes SIGN IS LEADING the default.

-fsign-separate

Makes SIGN IS SEPARATE the default.

-fsimple-trace

Generates trace output at runtime for executed SECTION/PARAGRAPHS.

-fsource-location

Generates source location code, enabling information to be dumped on source location when the runtime aborts. The `-fsource-location` compiler flag is enabled by the `-g` compiler flag, and by the `-debug` compiler flag.

-fsplit-debug-mark

DEBUG marks respect max 72 characters (default)

-fstack-check

Enables stack checking debug function. The stack checking debug function allows the user to trace back through the stack of calling programs to the currently running line of source in a program. The `-fstack-check` compiler flag is enabled by the `-g` compiler flag, and by the `-debug` compiler flag.

-fstatic-call

Causes static C function calls to be generated for the CALL statement.

This implies that all CALL'ed programs are C function that are linked with the current program. When using the `-fstatic-call` compiler flag, no external dynamic resolution is performed at runtime.

-fstrict-compare-low

Causes display of numeric variables containing low values not equal to zero or spaces when compared.

Corresponds to compiler configuration setting:
strict-compare-low: yes

-fstrict-record-contains

Causes the RECORD CONTAINS clause to be strictly respected. (This is the default.)
Note that this compiler flag contains underscores, instead of hyphens.

When an FD contains a RECORD CONTAINS xx CHARACTERS clause, such as:
RECORD CONTAINS 70 CHARACTERS

but the actual record size described has fewer characters than are named in the clause, COBOL-IT now sends the stated number of characters in the RECORD CONTAINS clause for both MIN-RECORD-SIZE and MAX-RECORD-SIZE.

Note that in the case above, when using VBISAM, which does not rely on EXTFH handling, COBOL-IT would detect the smaller actual record size, and pass it through as the MIN-RECORD-SIZE.

This behavior increases compatibility with the EXTFH implementation of the c-Tree ISAM file system. If you encounter an EXTFH-compliant file system that requires that the non-EXTFH default behavior described above, then you may compile with the

`-fno-strict_record_contains` compiler flag, or set :

`strict-record-contains:no` in the compiler configuration file .

-fsyntax-only

Performs syntax error checking only. Output is limited to results of syntax check.

-fthread-safe

Generate thread-safe executables. For more details, see Guidelines on operating in a thread-safe environment below.

When implementing the COBOL-IT Region Interface, required to provide region isolation. For more details, see the chapter on the COBOL-IT Region Interface.

-ftrace

Generates trace output at runtime, listing the SECTION/PARAGRAPH names as they are executed.

-ftrace-ts

Generates trace code with timestamp (Executed SECTION/PARAGRAPH)

-ftrace-upon-sysout

Causes trace output to be written upon SYSOUT. Default is to write trace output upon SYSERR.

-ftraceall

Generates trace output at runtime, listing SECTION/PARAGRAPH/STATEMENTS names as they are executed.

-ftrap-unhandled-exception

is useful in cases where certain EC compiler configuration file flags are set to yes, yet ON EXCEPTION/ON SIZE ERROR/ON OVERFLOW language is not present in the COBOL program. In these cases, using the `-ftrap-unhandled-exception` compiler flag causes the information made available to the user to be enhanced when the program aborts.

As an example, in a case where there is a compiler configuration flag setting of :
EC-SIZE:yes

and where this phrase does not contain an ON SIZE ERROR clause, the program would abort in cases where a SIZE ERROR was triggered. In combination with `-ftrap-unhandled-exception:yes`, all size error events will be captured.

As another example, where there is a compiler configuration flag setting of:
EC-SIZE-ZERO-DIVIDE:yes

In combination with `trap-unhandled-exception:yes`, setting `EC-SIZE-ZERO-DIVIDE:yes` will capture all division by zero error events if no ON SIZE ERROR clause is present.

Note- this applies to the following EC- compiler configuration flags:

```
EC-IMP-ACCEPT :yes
# For Accept exception
EC-IMP-DISPLAY :yes
# For Display exception
EC-SIZE : yes
#For Arithmetic exception
EC-OVERFLOW : yes
# For String/Unstring exception
```

Note that all of the EC- compiler configuration flags can be set to yes using the `-debug` compiler flag. You may wish that your error procedure be always called on any exception, and thereby ensure that your server will handle it and not crash. In these cases, you should use the `-debug` compiler flag together with the `-ftrap-unhandled-exception` flag.

For details on how to install and uninstall error procedures, see the COBOL-IT Library Routines documentation for the `CBL_ERROR_PROC` library routine. `CBL_ERROR_PROC` installs or uninstalls an error procedure, which is run when a program-ending error occurs. The Error Routine allows the user to register procedures that will automatically be executed either when a program-ending error occurs.

-ftruncate-listing

Causes output of the `-t <file>` compiler flag to be truncated at column 76

-funstring-use-move

The `-funstring-use-move` compiler flag affects the behavior of the `UNSTRING` verb. When using the `-funstring-use-move` compiler flag, if the target of an `UNSTRING INTO` operation is described as `PIC 9`, then the operation will be performed using a `MOVE` operation instead of raw copy operation. Then rules defined by the `move-picx-to-pic9` compiler configuration flag are used for conversion.

-futf-8

Instructs the compiler that the source file, and literals are UTF-8 encoded. The `-futf-8` compiler flag can be used with, or without the `-codepage` compiler flag.

If the `-futf-8` compiler flag is used and the `-codepage` compiler flag is not specified, then `-codepage UTF-8` is assumed.

```
>cobc -futf-8 <source-file>
```

If, however, you wish to compile your source with another codepage (for example, the `LATIN1` codepage), you should explicitly include that codepage declaration.

```
>cobc -futf-8 -codepage latin1 <source-file>
```

-futf16-le

Causes fields declared as `PIC N` to be stored as UTF16-LE (Little Endian). Note that by default, fields declared as `PIC N` are stored as UTF16-BE (Big Endian).

Note- Big-Endian refers to a convention for the storage of integers in memory in which the most significant bytes are stored in the bytes with the lower addresses. The integer 256 is stored in Big-Endian format, as follows: 00000001 00000000.

Little-Endian refers to a convention of or the storage of integers in memory in which the least significant bytes are stored in the bytes with the lower addresses. The integer 256 is stored in Little-Endian format, as follows: 00000000 00000001.

“Endianness” is determined by the Processor of your computer. A simple rule of thumb is that if you have an x86 processor, your platform has a little-endian data storage convention. Otherwise, you most likely have a big-endian data storage convention. Check with your system administrator if you have any questions.

Used in conjunction with the compiler configuration file setting:

ls-utf16: yes

-fvalidate-dep-on

Causes the setting of `DEPENDING ON` to be set at runtime.

Corresponds to compiler configuration file setting:

validate-dep-on: yes

-fvalidate-only

Compiles source, no output produced, EXEC are ignored.

-fvalue-of-id-priority

Gives priority to the literal or data element named in the VALUE OF FILE-ID clause in the FD, causing the target of the VALUE OF FILE-ID clause in the FD to override the target of the ASSIGN clause for the file.

-fvalue-size-is-auto

Causes the CALL ..USING BY VALUE : default SIZE IS clause to be set to AUTO (current default is SIZE IS 4).

-fvbisam

Forces use of the VBISAM Extfh indexed file engine.

This is the default setting in version 3.x and prior versions of COBOL-IT. However, in the future release of COBOL-IT version 4.x, VBISAM will be deprecated and D-ISAM will become the default. At that time, continued use of VBISAM files will require that the VBISAM Extfh indexed file engine be activated either by using the -vbisam compiler flag, or with the use of the COB_EXTFH=vbisamextfh runtime environment variable setting.

-fvms-error-handler

Causes the default file error handler to always abort (emulation of VMS behavior)

Corresponds to compiler configuration file setting:

vms-error-handler: yes

-fxparse-event

The -fxparse-event compiler flag causes the XML PARSE statement to generate START-OF-DOCUMENT and END-OF-DOCUMENT XML-EVENTS.

Guidelines for handling Linkage Section parameters

There are a number of scenarios where different behaviors could be desired, with respect to the handling of Linkage Section parameters. These include:

- * The allocation of memory to fields that are declared in a USING clause but not used
- * The passing of parameters that are null pointers
- * The omission of parameters, which may be subsequently used.

The -f compiler flags related to the handling of Linkage Section parameters are:

-falloc-unused-linkage Causes the compiler to allocate static memory for level 01 fields in the Linkage Section that are not used in either a USING clause or an ENTRY clause.

-fprotect-linkage Generates code at the entry point of a program containing a USING xxx clause.

-fsafe-linkage Generates code at the entry point of a program containing a USING xxx clause. This allows for the omission of parameters. Doing this will avoid a SIGVEC being returned by the debugger when all linkage parameters are not provided.

Guidelines for modifying default handling of the LOCK MODE

There are several `-f` compiler flags designed to allow the user to create implicit declarations for LOCK MODE in files.

The `-f` compiler flags related to the treatment of LOCK MODE are:

- fexclusivelock** Causes all files with no LOCK MODE clause in their SELECT statement to be declared implicitly as LOCK MODE is EXCLUSIVE.
- fmanuallock** Causes all files with no LOCK MODE clause in their SELECT statement to be declared implicitly as LOCK MODE is MANUAL unless a SHARING clause in the SELECT statement or in the OPEN statement indicates otherwise.
- fshare-all-autolock** Causes all files with a SHARE WITH ALL clause in their SELECT statement to be declared implicitly as LOCK MODE IS AUTOMATIC.
- fshare-all-manulock** Causes all files with a SHARE WITH ALL clause in their SELECT statement to be declared implicitly as LOCK MODE IS MANUAL.
- fshare-all-default** Causes all files to be declared implicitly as SHARE WITH ALL.

Guidelines for use of Checkpoints

COBOL-IT supports the CHECKPOINT verb, which can be used to enable checkpoint processing in programs compiled with the `-fcheckpoint` compiler flag.

- Usage:

```
CHECKPOINT [checkpoint prefix] CONTINUE [GIVING field]
or
CHECKPOINT [checkpoint prefix] EXIT [RETURNING|WITH] [return value]
[GIVING field]
```

- CHECKPOINT saves the current status of a program (Field Value, call stack , perform stack) into a file named <checkpoint prefix><program id>.ctx.
- When used with CONTINUE, the runtime saves the status and continues execution.
- When used with EXIT RETURNING *value*, the runtime saves the status, and exits the

program (and all calling parent programs) returning the value.

- When called back with checkpoint file, the runtime reloads the program status and continues the execution at the statement just following the CHECKPOINT line. If a GIVING clause is specified, the field is set to "1" when reloading and to "0" when execution is continuing.
- Reloading a checkpoint is done by using the `--reload` compiler flag, or by using the `--reload` compiler flag together with the `--checkpoint` compiler flag, and the `<checkpoint prefix>`:

```
>cobc --reload myprog
or
>cobc --reload --checkpoint <checkpoint prefix>myprog
```

Guidelines for use of Profiler

COBOL-IT provides a profiling utility that allows you to analyze where your programs are spending time by providing output, in Excel format, on the number of times a paragraph is executed, and both CPU and elapsed time spent in each paragraph.

The time is expressed in a platform-dependent unit, named "Ticks" as provided by the runtime environment of the "C" Compiler at hand. Please check the clock function for more information about this.

Because of the coarseness of this unit, some of the times measured as described above may be zero, while the paragraph has been executed one or more times.

By default, on program exit, the COBOL_IT runtime generates a file named `cob_profiling_<PID>_final.xls`, where [PID] is the PID number. This file is a tab separated text file, and can be opened directly with a spreadsheet like OpenOffice Calc or Microsoft Excel.

To enable the profiling utility, compile your program with the `--fprofiling` compiler flag.

Example:

```
>cobc -fprofiling sample.cbl
>cobcrun sample
>cob_profiling_11344_final.xls
```

Returning Profiling Dumps prior to the program exit:

Dumping Profiling Data at the Module Level

You can generate separate output files for each module in your COBOL runtime session by setting the runtime environment variable `COB_PROFILING_EACH_MODULE=Y`. In this case an XLS file is output at the exit of each module in the application using the naming convention

[MODULE]_[PID]_profile.xls.

Using the PRAGMA statement to produce Profiling Reports

The PRAGMA statement provides internal compiler control, for profiling. The first literal is the command sent to the compiler.

Format 1

```
PRAGMA "PROFILING" { literal-1 } ...
```

Format 2

```
PRAGMA "DUMP" { literal-1 } ...
```

Syntax

1. literal-n is a character string.

General Rules

1. The PRAGMA "PROFILING" statement causes programs compiled with `-fprofiling` to report time measurements from the current PRAGMA statement to the next PRAGMA, or to the end of the program.
2. The PRAGMA "PROFILING" statement should be entered in column 8, as in the Code Sample below.
3. The PRAGMA "DUMP" statement causes a profiling report to be generated when executed. Profiling reports produced by the PRAGMA "DUMP" statement will overwrite previous PRAGMA "DUMP" files with the same name.

PRAGMA "PROFILING" Code Sample

```
PROCEDURE DIVISION.  
.  
.  
PRAGMA "PROFILING" "STEP1".  
.  
.  
PRAGMA "PROFILING" "STEP2".
```

PRAGMA "DUMP" Code Sample

```
PROCEDURE DIVISION.  
.  
.  
PRAGMA "DUMP" "REPORT".  
.  
.
```

Info profiling debugging command

The cobcdb compiler command `>info profiling` causes a profiling dump to be produced,

dumping profiling information at the current point in the program. Profiling information is dumped in the .xls file format.

Attaching a program compiled with -fprofiling to a running process

A program compiled using the -fprofiling compiler flag may be attached exactly like a program to debug (with cobcdb -p <pid>) even if the program is not compiled for debugging .

Guidelines for thread-safe programs

The -fthread-safe compiler flag causes thread-safe code to be generated.

Take care to use the -fthread-safe compiler flag when interoperating with Java. All programs in an application that interoperates with Java should be compiled with -fthread-safe, even those not directly calling, or called by Java. For more information on Interoperability between COBOL-IT and Java, see the Chapter on **COBOL/Java Interoperability**.

Thread safe internal structures now use the pthread library, which is available in AIX, Linux, HP-UX, Itanium, Sun Solaris and Windows operating environments.

Thread memory release and support for a program monitor

COBOL-IT supports Thread memory release and support for a program monitor taking control of the abort & exit of the COBOL-IT runtime. A typical “program monitor” should use the following scheme to call COBOL programs:

```
/* Allocate Runtime Data memory for this thread */
cit_runtime_t * constrtd = cob_get_rtd();

int (*func)();
int res;

/*initialize COBOL Thread runtime */
cob_init(rtd, argc, argv);

/* Prepare the Runtime to exit with a long jump instead of exit(x)*/
res = cob_setjmp(rtd);
if ( res == 0 ) {
    /* Find the COBOL Program */
    func = cob_resolve(rtd, "prog");
    if ( func ) {
        /* Call it and all other needed*/
        func();
    }
    /* if the Program goes Here the the COBOL exit normally */
    /* with a GO BACK */
    ...
}
```

```
} else if ( res > 0 && res < 127) {
    /* COBOL Program exit normally through STOP RUN */
    /* res hold the RETURN-CODE */
    ...

} else {
    /* COBOL Program exit on runtime abort */
    /* res hold the error code */
    ...

}

/* Before leaving the Thread Release Runtime Data */
cob_rtd_tidy ();
```

Compiler `-w` Flags

`-w` compiler flags are warning flags.

All `-w` compiler flags are provided in the form : `-Wwarning-name`

To enable the flag, use: `-Wwarning-name`

Example:

```
cobc -Wall sample.cbl
```

To disable the flag, use: `-Wno-warning-name`

Example:

```
cobc -Wno-all sample.cbl
```

Warnings are evaluated in the command line order.

For example, use the command:

```
>cobc -Wall -Wno-archaic
to enable all warnings but the archaic features.
```

Note that `cobc -w` when used with no warning flag has the effect of disabling all warnings.

Example:

```
cobc -w sample.cbl
```

The Warning Compiler flags are:

-w

Disables all warnings.

-Wall

Enables all warnings.

-Warchaic

Warns if archaic features are used.

-Wbdb

Warns about bdb license. (default).

-Wcall-params

Warns if non 01/77 items are used for CALL parameters (NOT set with Wall).

-Wconstant

Warns if inconsistent constant is used.

-Wimplicit-define

Warns of implicitly defined data items.

-Winformation

Warns information about 'not recommended' code. –Winformation is applied by default.

-Wlinkage

Warns of dangling LINKAGE items. (NOT set with -Wall).

-Wobsolete

Warns if obsolete features are used.

-Wparentheses

Warns of lack of parentheses around AND within OR.

-Wredefinition

Warns if incompatible redefinition of data items are used. -Wredefinition is defined by default.

-Wstrict-typing

Warns of type mismatch strictly.

-Wsuggestion

Warns suggestions about 'not recommended' code. –Wsuggestion is applied by default.

-Wterminator

Warns of lack of scope terminator, such as END-XXX. (NOT set with -Wall)

-Wtruncate

Warns of possible field truncations. -Wtruncate is applied by default.

-Wunreachable

Warns of unreachable statements.

Compiler Configuration File

The Compiler Configuration File describes compiler behaviors, and is checked by the compiler `cobc` at compile time.

The default compiler configuration file, `default.conf`, is located in `$COBOLITDIR/config` on Windows-based systems, and on `$COBOLITDIR/share/COBOL-it/config` on UNIX/Linux-based systems.

When modifying compiler configuration flags included in any of the compiler configuration files provided at installation time, care must be taken when upgrading the COBOL-IT compiler, as these files will be overwritten at installation time. This problem can be avoided by renaming files which have been modified, and using the `-conf=<myconf.conf>` compiler flag.

COBOL-IT also provides a number of sample compiler configuration files, which contain settings that enhance compatibility with different COBOLs.

These include

COBOL	Configuration file
Bs2000	<code>bs2000.conf</code>
COBOL 2002	<code>COBOL2002.conf</code>
COBOL 85	<code>COBOL85.conf</code>
IBM	<code>ibm.conf</code>
Micro Focus	<code>mf.conf</code>
MVS	<code>mvs.conf</code>

The compiler can be directed to use any of these files by using the `-std` compiler flag. The `-std` compiler flag will recognize the filename (without extension) of any of the provided compiler configuration files as a parameter, as follows:

```
>cobc -std=bs2000 sample.cbl
```

You may create your own compiler configuration file, and name it whatever you wish.

To direct the compiler to reference a renamed compiler configuration file, locate your renamed compiler configuration file in the `$COBOLITDIR/config` on Windows-based systems, and on `$COBOLITDIR/share/config` on UNIX/Linux-based systems., and use the `-conf` compiler flag as follows:

```
>cobc -conf=myconf.conf
```

To facilitate the creation of your own compiler configuration file, you may use the “include” phrase to indicate that you wish to include all of the settings of a named compiler configuration file, as follows:

```
include "default.conf"
```

```
assign-clause=mf  
binary-byteorder=native
```

Brief descriptions, and valid settings of supported compiler configuration flags are documented below:

77-opt:[yes/no]

Default is `77-opt:no`.

When set to yes,

Optimizes the use of integers stored in `USAGE DISPLAY` or `PACKED` fields in level 77 data items. The 77-opt optimizations are enabled by use of the `-O` compiler flag.

accept-but-ignore-comp6-signed:[yes/no]

Default is `accept-but-ignore-comp6-signed: no`

Affects compilation of data items described as signed, with usage `COMP-6`.

When set to yes,

`COMP-6` defined as signed (`PIC S9`) are compiled as non signed.

accept-with-auto:[yes/no]

Default is `accept-with-auto:no`.

When set to yes,

Causes the `WITH AUTO` clause to be assumed by default on a field-level `ACCEPT` statement.

When set to no,

The `WITH TAB` clause is assumed by default on a field-level `ACCEPT` statement.

accept-with-update:[yes/no]

Default is `accept-with-update:no`

When set to yes,

All field-level `ACCEPT` statements are interpreted as containing the “`WITH UPDATE`” clause.

align-8:[yes/no]

Default is `align-8:no*`.

* On HP Itanium based systems this flag is always enabled

When set to yes,

Aligns 01-level and 77-level data on 8 byte boundaries.

When set to no,

Aligns 01-level and 77-level data on 4-byte boundaries.

all-external-call:[yes/no]

Default is `all-external-call:no`

Internal use only. Causes all CALL statements to be considered EXTERNAL. This should not be changed.

all-external-link:[yes/no]

Default is `all-external-link:no`

When set to yes,

Causes the targets of the CALL statement to all be assumed to be external-links.

This can improve performance at runtime by optimizing the resolution of the CALL statement.

alloc-unused-linkage:[yes/no]

Default is `alloc-unused-linkage:no`.

When set to yes,

Causes the compiler to allocate static memory for level 01 fields in the Linkage Section that are not used in either a USING clause or an ENTRY clause.

When set to no,

If level 01 fields in the Linkage Section are not used in either a USING clause or an ENTRY clause, these fields are initialized to NULL, and no memory is allocated for them.

Note that usage of of a field for which no static memory has been allocated will provoke a Memory Fault. For cases such as described above, where static memory has not been allocated at compile time, it is possible to programmatically allocate static memory to an unused linkage field using the SET [linkage field] to ADDRESS OF [data-pointer] statement, and avoid the Memory Fault condition.

as400-like :[yes/no]

Default is `as400-like:no`

Causes the LIKE clause to act compatibly with the AS400 implementation of the LIKE clause.

When set to yes, a field declared with the LIKE clause is assumed to be described as PIC X (other field's byte size).

assign-clause: [COBOL2002 / mf / ibm / external]

Default is `assign-clause: mf`

Improves compatibility with named COBOL on compilation of ASSIGN clause.

When set to external,

Targets of all all ASSIGN causes are considered to be EXTERNAL.

auto-initialize:[yes/no]

Default is `auto-initialize:yes`

Affects automatic initialization behavior.

When set to yes, working-storage is automatically initialized when the program is loaded into memory.

auto-load-symb:[yes/no]

Default is `auto-load-symb:yes`

When set to yes,

Provides additional control, as regards static symbol definition, by causing the `static.symb` and `user.symb` files to be automatically loaded with the compiler configuration file.

`static.symb` is provided with the compiler distribution, and should not be changed by the user. `Static.symb` includes symbols declared by Pro*Cob and Tuxedo. Note that it is overwritten when the compiler is updated.

`user.symb` is user-definable and may be placed in the config directory of the COBOL-IT installation, or in the current directory. If `user.symb` is missing, no error is generated.

Both `static.symb` and `user.symb` files may include static-link declarations.

autolock:[yes/no]

Default is `autolock:no`.

When set to yes,

Sets default for SELECT to LOCK MODE IS AUTOMATIC

bdb:[yes/no]

Default is `bdb:no`.

When set to yes,

Activates the usage of Oracle Berkeley DB isam files

bin-opt:[yes/no]

Default is `bin-opt:no`.

When set to yes,

Enables binary operation optimization. The `-bin-opt` optimizations are enabled by use of the `-O` compiler flag.

bin-opt-strict:[yes/no]

When set to yes,

Causes `-fbin-opt` binary operation optimization to be strictly respected.

binary-byteorder: native / big-endian

Default is `binary-byteorder:big-endian`

Defines the binary byte order of USAGE COMPUTATIONAL data items.

On little-endian platforms like x86, `binary-byteorder` should be set to “native”.

Note- Big-Endian refers to a convention for the storage of integers in memory in which the most significant bytes are stored in the bytes with the lower addresses. The integer 256 is stored in Big-Endian format, as follows: 00000001 00000000.

Little-Endian refers to a convention of or the storage of integers in memory in which the least significant bytes are stored in the bytes with the lower addresses. The integer 256 is stored in Little-Endian format, as follows: 00000000 00000001.

The Native option provides the ability to use the “Endianness” of the native processor.

“Endianness” is determined by the Processor of your computer. A simple rule of thumb is that if you have an x86 processor, your platform has a little-endian data storage convention. Otherwise, you most likely have a big-endian data storage convention. Check with your system administrator if you have any questions.

binary-size: 2-4-8 / 1-2-4-8 / 1--8

Default is `binary-size:1-2-4-8`

Describes relationship between PIC description and storage bytes of a binary data item.

Binary byte size defines the allocated bytes according to number associated with PIC 9 clause.

Value:	signed	unsigned	bytes
	-----	-----	-----
'2-4-8'	1 - 4	1 - 4	2
	5 - 9	5 - 9	4
	10 - 18	10 - 18	8
'1-2-4-8'	1 - 2	1 - 2	1
	3 - 4	3 - 4	2
	5 - 9	5 - 9	4
	10 - 18	10 - 18	8
'1--8'	1 - 2	1 - 2	1
	3 - 4	3 - 4	2
	5 - 6	5 - 7	3
	7 - 9	8 - 9	4
	10 - 11	10 - 12	5
	12 - 14	13 - 14	6
	15 - 16	15 - 16	7
	17 - 18	17 - 18	8

binary-truncate:[yes/no]

Default is `binary-truncate:yes`

Governs the behavior of the runtime when binary data is truncated.

When set to yes, the default value of yes corresponds to the behavior of the TRUNC compiler directive. The TRUNC compiler directive causes truncation to be governed by the number of digits in the PICTURE clause, when moving data into data items described as COMP, BINARY, or COMP-4.

When set to no, the value of no corresponds to the behavior of the NOTRUNC compiler directive. The NOTRUNC compiler directive causes truncation to be governed by the capacity of allocated storage when moving data into data items described as COMP, BINARY, or COMP-4.

bitfield-first-is-lsb: [yes/no]

Default is `bitfield-first-is-lsb:no`

call-comp5-as-comp:[yes/no]

Default is `call-comp5-as-comp:no`.

When set to yes, affects the behavior of the CALL statement. On little-endian platform (intel Linux, Windows) when a call USING clause contains a literal. When set to yes, causes the literal to be copied as a COMPUTATIONAL value, rather than as a COMP-5 value.

Example: In the statement:

CALL "subprogram" USING 1234.

The literal 1234 is passed as a COMP value when call-comp5-as-comp is set to yes.

Otherwise, the literal 1234 is passed as a COMP-5 value.

call-lowercase: [yes/no]

Default is `call-lowercase:no`

The call-lowercase compiler configuration flag affects the handling of literals that are the target of a CALL statement. As an example, consider the literal "MyProg" in the statement: CALL "MyProg".

When set to yes, all of the characters in the literal "MyProg" are converted to lowercase.

When set to no (the default), this setting is ignored.

call-opt: [yes/no]

Default is `call-opt:no`.

When set to yes,

Enables CALL statement optimization. Programs containing CANCEL statements should **not** be compiled with call-opt:yes.

call-uppercase: [yes/no]

Default is `call-uppercase:no`

The call-uppercase compiler configuration flag affects the handling of literals that are the target of a CALL statement. As an example, consider the literal "MyProg" in the statement: CALL "MyProg".

When set to yes, all of the characters in the literal “MyProg” are converted to uppercase.
When set to no (the default), this setting is ignored.

carealia-sign: [yes/no]

Default is `carealia-sign: no`

When set to yes, the `carealia-sign` compiler configuration flag causes signed (PIC S9) variables to be stored according to CA-REALIA sign storage conventions. For more detail, see `-fcarealia-sign` compiler flag documentation.

C-cmd-line:[yes/no]

Default is `C-cmd-line:no`.

When set to yes,

When used with `-x`, causes the program to receive command line parameters as though they were given in C. In this case, command line parameters are read as they would be through a “C” interface. For example: `(int argc , char **argv)`

C-data-init:[yes/no] [Internal use only]

Default is `C-data-init:no`

Controls if the C data structure created by the compiler is initialized in the source (at compilation time) or at runtime. This should not be changed.

check-linkage-bound: [yes/no]

Default is `check-linkage-bound: yes`.

checkpoint: [yes/no]

Default is `checkpoint:no`.

When set to yes,

Enables setting of checkpoints. Program state is saved at checkpoints, and can be reloaded. For more details on the usage of the checkpoint flag, see the **Guidelines for use of Checkpoints**.

cics: [yes/no]

Default is `cics: no`.

When set to yes,

The compiler generates CICS-compliant code.

cmp-inline:[yes/no]

Default is `cmp-inline:no`

When set to yes,

Causes comparisons to be inlined in the C code instead of through calls to the runtime library when possible.

cmp-opt:[yes/no]

Default is `cmp-opt:yes`

The `cmp-opt` compiler flag activates optimizations when comparing literals with variables. When set to `no`, this functionality is disabled.

cobol-lines: [yes/no]

Default is `cobol-lines: no`.

When set to `yes`,

When compiling with `gcc`, line directives corresponding to COBOL source code line numbers are added to the “C” source code.

codepage: <codepage-id>

Names the default codepage to be used.

Example: `codepage: 1252`

See compiler option `-codepage` for more details.

comp5-byteorder: [native/big-endian]

Default is `comp5-byteorder: native`

Sets the byte ordering used for COMP-5 data.

When set to `native`, a little-endian byte ordering is set.

When set to `big-endian`, a big-endian byte ordering is set.

compat-display-to-int:[yes/no]

Default is `compat-display-to-int:no`

When set to `yes`, provides compatibility with older versions of COBOL-IT Compiler Suite (prior to version 3.10.5) as regards `display to int` functionality.

complex-odo:[yes/no]

Default is `complex-odo:no`

The compiler flag `complex-odo` is now obsolete. `OCCURS DEPENDING ON` is always considered to be `complex`.

compute-ibm:[yes/no]

Default is `compute-ibm: no`.

Beginning with version 3.7.20 the `-fcompute-ibm` compiler flag is used by default.

When set to `yes`,

Causes arithmetic expressions (like `a+B*c`) in `COMPUTE` statements, and comparisons to use IBM COBOL defined rules for determining the number of decimals used in intermediate results. For details, please consult IBM documentation.

When set to `no`,

Causes the maximum number of decimals (machine-dependent, up to 37) to be used in intermediate results of arithmetic expressions in COMPUTE statements and comparisons.

compute-ibm-trunc: [yes/no]

When compute-ibm is set to yes, causes intermediate results to be truncated (default).

console-is-sysfile: [yes/no]

Default is `console-is-sysfile:no`

When set to yes,

Causes DISPLAY UPON CONSOLE statements to be redirected to <file> when the compiler flag `-sysout=<file>` is used. This allows users to maintain a pre-2.11 behavior, which was determine to be defective.

constant: "key=value"

Provides a way to define constants that can be tested for purposes of conditional compilation. When using the constant "key=value" compiler flag, the conditional compilation below will test true.

```
$if key=value  
$else  
$end
```

continuation-line

Default is `continuation-line:no`

When set to yes, allows a hyphen in column 7, with no following text, to be recognized as not being a continuation line. When set to yes, a hyphen in column 7, with no following text in quotes is not recognized as a continuation line.

Note: A hyphen in column 7, with following text, in quotes, is always recognized as a continuation line.

For rules on the handling of continuation lines, see Line Continuations in the COBOL-IT COBOL Reference Manual.

copy-default-leading:[yes/no]

Default is `copy-default-leading:no`

When set to yes, affects the behavior of the COPY REPLACING statement.

When copy-default-leading is set to yes and when using the `==xxx==` notation in a COPY REPLACING statement, the LEADING phrase is assumed by default. The LEADING phrase

indicates that only the LEADING characters identified will be replaced if they match text in the copy file.

copy-exec-replace[yes/no]

Default is `copy-exec-replace: no`.

When set to yes, affects the behavior of the COPY REPLACING statement. When set to yes, and when a COPY REPLACING == xxx == statement is performed, text inside EXEC / END-EXEC blocks are also replaced if applicable.

copy-mark: [yes/no]

Default is `copy-mark: no`.

When set to yes,

Adds mark for begin/end of COPY In listing and preprocessed file

Note: The copy marks are:

*++SCOPY .\copy/sample.CPY (beginning of COPY file)

[COPY file is listed here]

*--SCOPY .\copy/sample.CPY (end of COPY file)

copy-partial-replace:[yes/no]

Default is `copy-partial-replace:no`

When set to yes,

The copy-partial-replace compiler configuration flag increases the compatibility of the COPY REPLACING behavior with Micro Focus compiler behaviors.

When a pattern like COPY FIC1 REPLACING == WJXX- == BY == WJ03- == is processed :

If this flag is on, the preprocessor uses a partial replacement as defined by MF and ANSI2002 standard.

If it is off (the default) the IBM mainframe and ANSI85 standard is used.

The copy-partial-replace compiler compiler configuration flag corresponds to the compiler flag :
`-fcopy-partial-replace`

crtstatus-map:[cit-value] [user-value]

Allows the user to create their own CRT STATUS Map, as described below.

When `crt-status-var` is declared as UNSIGNED-INT, the runtime copies `user-value` directly into the `crt-status-var`. For this case, if you wish to remap the first four function keys, from CIT-Values of 1001 through 1004 to single digits 1 through 4, you would add the entries:

`crtstatus-map: 1001 1`

`crtstatus-map: 1002 2`

`crtstatus-map: 1003 3`

`crtstatus-map: 1004 4`

The use of hex notation (x00 through x127) to describe characters in user-value is supported. For this case, if you wish to remap the first four function keys, from CIT-Values of 1001 through 1004 to single digits 1 through 4, you would add the entries:

```
crtstatus-map: 1001 x31
crtstatus-map: 1002 x32
crtstatus-map: 1003 x33
crtstatus-map: 1004 x34
```

Hex notations permit the user to use values that fall outside of the normal alphanumeric range. More than one character can be represented using hex notation. For example, if you wish to map the cit-value of 1004 to the user-value of 1234 using hex notation, you would add the entry:

```
crtstatus-map 1004 x31323334
```

In the example above, in the description of the user-value, after the “x”, the values “31”, “32”, “33”, “34” are concatenated to represent the string “1234”. When using hex notations, it is recommended that the receiving field be described as PIC X(4).

If no crtstatus-map is defined , CRT STATUS values are converted to PIC 9(4) and copied into the crt-status-var.

ctree: [yes/no]

Default is `ctree:no`.

When set to yes,
Activates the usage of C-tree isam files .

ctree-field-numbering: [yes/no]

Default is `ctree-field-numbering:no`

When set to yes,
Causes the CTREE XDD generator to generate a prefix F <field-number> before field names.
Use with `-fgen-xdd` compiler flag.

ctree-no-full-qualification: [yes/no]

Default is `ctree-no-full-qualification:no`.

When set to yes,
Affects the behavior of the gen-xdd compiler flag, causing it to not generate the fully qualified data names in the XDD description of the file. When using the `ctree-no-full-qualification` compiler flag with the `gen-xdd` compiler flag, the field name generated is :
`xxx_<Field_Name>`

Where:

xxx is a unique number (position in the structure),

<Field_Name> is the name of the data field, without any other prefix or suffix.

Use with the gen-xdd compiler flag.

curdir-include: [yes/no]

Default is `curdir-include:yes`.

When set to yes,

Causes COPY files to first be searched for in the current directory, before locations described with the `-I <Path>`, or with environment variables. The COPY search is performed for files with default extensions, and with extensions described with the `-ext` compiler flag. This is a default behavior.

When set to no,

Causes the search for a COPY file to not search for COPY file in the current directory, unless that directory is named by a `-I` compiler flag, or by a `COB_COPY_DIR`, or `COBCPY` environment variable.

datacompress: <integer>

Default is `datacompress:0`.

- To enable compression in VBISAM, you must add :

```
$SET DATACOMPRESS "x"
```

before the SELECT statement.

"x" is a numeric integer literal, ranging from 1 to 9. Higher values produce better compression, at the expense of performance. A setting of 1 provides the best performance, a setting of 9 the best compression.

All users of an indexed file created with compression must re-compile their programs with the `DATACOMPRESS` compiler directive and use the same compression setting.

To disable compression (after having enabled it)

```
$SET NODATACOMPRESS
```

debug-exec: [yes/no]

Default is `debug-exec:no`.

When set to yes,

Affects the tracing of Exec statements when debugging code that has been compiled with the integrated pre-processor (`-preprocess`). When using the Integrated Preprocessor Interface, the default behavior of the debugger is to `-not-` trace (display) the code generated by the external preprocessor. Only the original source EXEC statements are shown. The `-fdebug-exec` compiler flag enables the tracing (debugging) of the generated code.

debugging-line:[yes/no]

Default is `debugging-line:no`.

When set to yes,
Enables support for debugging lines. (Source lines that contain 'D' in indicator column)

debug-parser: [yes/no]

Default is `debug-parser:no`.

When set to yes,
Allows for the debugging of the parser. (maintainer use only).

decimal-optimize:[yes/no]

Default is `decimal-optimize:no`

When set to yes, optimizes the conversion from DISPLAY/COMP-3 to binary values in COMPUTE statements. When several COMPUTE statement are in the same paragraph, the compiler will minimize the conversions from DISPLAY/COMP-3 to binary values for fields that are used (and not modified) in different statements in the same paragraph. Decimal-optimize is set to yes by default when using either the -O or -O2 compiler flags.

defaultbyte:[any integer]

Default is `defaultbyte:0`

Corresponds to the DEFAULTBYTE directive. Sets the character used to initialize undeclared Working-Storage.

If specified , the integer value is the ASCII Value of default byte that will be used to fill memory of data items that have been declared in Working-Storage, and which do not have an VALUE declared. For example, to set the DEFAULTBYTE to an ASCII SPACE, the setting would be:
`defaultbyte: 32`.

Note that the compiler configuration flag `defaultbyte:[integer]` requires that the `use-defaultbyte:yes` compiler configuration flag also be set.

defaultcall:[any integer]

Default is `defaultcall:0`

Designates default call-convention used when no CALL-CONVENTION is mentioned in a CALL statement.

The integer value for `defaultcall` is a numeric literal representing the call convention .
The call convention number is a 16-bit number defined as follows:

Bit	Meaning
0-1	Unsupported
2	= 0 - RETURN-CODE is updated on exit = 1 - RETURN-CODE is not updated on exit

3	= 0 –CALL is resolved dynamically at run time = 1 –CALL is resolved statically at compilation time
4-5	Unsupported
6	Windows Only = 0 –CALL use standard C call conventions = 1 –CALL use “STDCALL” WINAPI call conventions (used to call Windows standard API)

Typical values are :

- 4 Do not modify RETURN-CODE
- 72 Windows API Call.

disam:[yes/no]

Default is `disam:yes`

When set to yes,
Activates the usage of DISAM files .

display-dos:[yes/no]

Default is `display-dos:no`

When set to yes,
Causes DISPLAY statements to use CR/LF.

display-ibm:[yes/no]

Default is `display-ibm:no`

When set to yes,
Affects the output of the DISPLAY Statement for numeric fields to be more compatible with IBM mainframe.

displaynumeric-edited-mf50: [yes/no]

Default is `displaynumeric-edited-mf50:yes` .

`displaynumeric-mf50:yes`

Used with :

`move-spaces-to-display-numeric:yes`

causes numeric-edited fields to be allowed to be initialized to spaces.

displaynumeric-mf50:[yes/no]

Default is `displaynumeric-mf50: no`

Enhances compatibility with Micro Focus behaviors for MOVE and INITIALIZE of data items described as USAGE DISPLAY NUMERIC.

When set to yes, the following are implied:

`move-picx-to-pic9:mf50`

`move-spaces-to-displaynumeric:yes`

When set to yes, compatibility with Micro Focus MOVE, INITIALIZE behaviours are enhanced.

Consider an example:

...

```
77 NUMERIC-FLD PIC 99.
```

...

```
INITIALIZE NUMERIC-FLD TO SPACE (NUMERIC_FLD: "<sp><sp>")
```

```
MOVE ZERO TO NUMERIC-FLD. (NUMERIC-FLD: "<sp>0\`")
```

div-check: [yes/no]

Default is `div-check:yes`.

When set to yes,

Enables the checking of divide operations when binary optimizations are turned on with the use of the `-fbn-opt`, or `-O` compiler flags. The effect is to cause divide-by-0 operations to generate an exception.

ebcdic-charset: [yes/no]

Default is `ebcdic-charset:no`.

The `-febcdic-charset` compiler flag requires a dedicated license.

When using the `-febcdic-charset` compiler flag, the COBOL-IT Compiler and Runtime store and manage data in the EBCDIC encoding format. Source code is stored in ASCII format.

When set to yes,

The `-febcdic-charset` compiler flag causes modules to be created that apply the EBCDIC character set to file operations, and internal data storage. When using the `-febcdic-charset` compiler flag, all of the programs in a runtime unit must be compiled with the `-febcdic-charset` compiler flag. If a main entry program that is not compiled with the `-febcdic-charset` compiler flag CALLs a program that is compiled with the `-febcdic-charset` compiler flag, or conversely, if a main entry program compiled with the `-febcdic-charset` compiler flag CALLs a program that is not compiled with the `-febcdic-charset` compiler flag, the CALL will fail, and the COBOL-IT runtime will abort.

emulate-vms [yes/no]

Default is `emulate-vms:no`.

When set to yes,

Causes spaces to be stripped from filenames and adds suffix `'.DAT'` if needed.

exception checking

Runtime exception checking is enabled when compiling with `-debug`, for the following compiler configuration flags. For details about the Runtime Exception Checking flags, see the file `exception.def`, which is located in `$COBOLITDIR\include\libcob`, in your distribution.

EC-ALL:[yes/no]
EC-ARGUMENT:[yes/no]
EC-ARGUMENT-FUNCTION:[yes/no]
EC-ARGUMENT-IMP:[yes/no]
EC-BOUND:[yes/no]
EC-BOUND-IMP:[yes/no]
EC-BOUND-ODO:[yes/no]
EC-BOUND-OVERFLOW:[yes/no]
EC-BOUND-PTR:[yes/no]
EC-BOUND-REF-MOD:[yes/no]
EC-BOUND-SET:[yes/no]
EC-BOUND-SUBSCRIPT:[yes/no]
EC-BOUND-TABLE-LIMIT:[yes/no]
EC-DATA:[yes/no]
EC-DATA-CONVERSION:[yes/no]
EC-DATA-IMP:[yes/no]
EC-DATA-INCOMPATIBLE:[yes/no]
EC-DATA-INFINITY:[yes/no]
EC-DATA-INTEGRITY:[yes/no]
EC-DATA-NEGATIVE-INFINITY:[yes/no]
EC-DATA-NOT_A_NUMBER:[yes/no]
EC-DATA-PTR-NULL:[yes/no]
EC-FLOW:[yes/no]
EC-FLOW-GLOBAL-EXIT:[yes/no]
EC-FLOW-GLOBAL-GOBACK:[yes/no]
EC-FLOW-IMP:[yes/no]
EC-FLOW-RELEASE:[yes/no]
EC-FLOW-REPORT:[yes/no]
EC-FLOW-RETURN:[yes/no]
EC-FLOW-SEARCH:[yes/no]
EC-FLOW-USE:[yes/no]
EC-FUNCTION:[yes/no]
EC-FUNCTION-NOT-FOUND:[yes/no]
EC-FUNCTION-PTR-INVALID:[yes/no]
EC-FUNCTION-PTR-NULL:[yes/no]
EC-I-O:[yes/no]
EC-I-O-AT-END:[yes/no]
EC-I-O-EOP:[yes/no]
EC-I-O-EOP-OVERFLOW:[yes/no]
EC-I-O-FILE-SHARING:[yes/no]
EC-I-O-IMP:[yes/no]
EC-I-O-INVALID-KEY:[yes/no]
EC-I-O-LINAGE:[yes/no]

EC-I-O-LOGIC-ERROR:[yes/no]
EC-I-O-PERMANENT-ERROR:[yes/no]
EC-I-O-RECORD-OPERATION:[yes/no]
EC-IMP:[yes/no]
EC-IMP-ACCEPT:[yes/no]
EC-IMP-DISPLAY:[yes/no]
EC-LOCALE:[yes/no]
EC-LOCALE-IMP:[yes/no]
EC-LOCALE-INCOMPATIBLE:[yes/no]
EC-LOCALE-INVALID:[yes/no]
EC-LOCALE-INVALID-PTR:[yes/no]
EC-LOCALE-MISSING:[yes/no]
EC-LOCALE-SIZE:[yes/no]
EC-OO:[yes/no]
EC-OO-CONFORMANCE:[yes/no]
EC-OO-EXCEPTION:[yes/no]
EC-OO-IMP:[yes/no]
EC-OO-METHOD:[yes/no]
EC-OO-NULL:[yes/no]
EC-OO-RESOURCE:[yes/no]
EC-OO-UNIVERSAL:[yes/no]
EC-ORDER:[yes/no]
EC-ORDER-IMP:[yes/no]
EC-ORDER-NOT-SUPPORTED:[yes/no]
EC-OVERFLOW:[yes/no]
EC-OVERFLOW-IMP:[yes/no]
EC-OVERFLOW-STRING:[yes/no]
EC-OVERFLOW-UNSTRING:[yes/no]
EC-PROGRAM:[yes/no]
EC-PROGRAM-ARG-MISMATCH:[yes/no]
EC-PROGRAM-ARG-OMITTED:[yes/no]
EC-PROGRAM-CANCEL-ACTIVE:[yes/no]
EC-PROGRAM-IMP:[yes/no]
EC-PROGRAM-NOT-FOUND:[yes/no]
EC-PROGRAM-PTR-NULL:[yes/no]
EC-PROGRAM-RECURSIVE-CALL:[yes/no]
EC-PROGRAM-RESOURCES:[yes/no]
EC-RAISING:[yes/no]
EC-RAISING-IMP:[yes/no]
EC-RAISING-NOT-SPECIFIED:[yes/no]
EC-RANGE:[yes/no]
EC-RANGE-IMP:[yes/no]
EC-RANGE-INDEX:[yes/no]
EC-RANGE-INSPECT-SIZE:[yes/no]
EC-RANGE-INVALID:[yes/no]
EC-RANGE-PERFORM-VARYING:[yes/no]
EC-RANGE-PTR:[yes/no]
EC-RANGE-SEARCH-INDEX:[yes/no]

EC-RANGE-SEARCH-NO-MATCH:[yes/no]
EC-REPORT:[yes/no]
EC-REPORT-ACTIVE:[yes/no]
EC-REPORT-COLUMN-OVERLAP:[yes/no]
EC-REPORT-FILE-MODE:[yes/no]
EC-REPORT-IMP:[yes/no]
EC-REPORT-INACTIVE:[yes/no]
EC-REPORT-LINE-OVERLAP:[yes/no]
EC-REPORT-NOT-TERMINATED:[yes/no]
EC-REPORT-PAGE-LIMIT:[yes/no]
EC-REPORT-PAGE-WIDTH:[yes/no]
EC-REPORT-SUM-SIZE:[yes/no]
EC-REPORT-VARYING:[yes/no]
EC-SCREEN:[yes/no]
EC-SCREEN-FIELD-OVERLAP:[yes/no]
EC-SCREEN-IMP:[yes/no]
EC-SCREEN-ITEM-TRUNCATED:[yes/no]
EC-SCREEN-LINE-NUMBER:[yes/no]
EC-SCREEN-STARTING-COLUMN:[yes/no]
EC-SIZE:[yes/no]
EC-SIZE-ADDRESS:[yes/no]
EC-SIZE-EXPONENTIATION:[yes/no]
EC-SIZE-IMP:[yes/no]
EC-SIZE-OVERFLOW:[yes/no]
EC-SIZE-TRUNCATION:[yes/no]
EC-SIZE-UNDERFLOW:[yes/no]
EC-SIZE-ZERO-DIVIDE:[yes/no]
EC-SORT-MERGE:[yes/no]
EC-SORT-MERGE-ACTIVE:[yes/no]
EC-SORT-MERGE-FILE-OPEN:[yes/no]
EC-SORT-MERGE-IMP:[yes/no]
EC-SORT-MERGE-RELEASE:[yes/no]
EC-SORT-MERGE-RETURN:[yes/no]
EC-SORT-MERGE-SEQUENCE:[yes/no]
EC-STORAGE:[yes/no]
EC-STORAGE-IMP:[yes/no]
EC-STORAGE-NOT-ALLOC:[yes/no]
EC-STORAGE-NOT-AVAIL:[yes/no]
EC-USER:[yes/no]
EC-VALIDATE:[yes/no]
EC-VALIDATE-CONTENT:[yes/no]
EC-VALIDATE-FORMAT:[yes/no]
EC-VALIDATE-IMP:[yes/no]
EC-VALIDATE-RELATION:[yes/no]
EC-VALIDATE-VARYING:[yes/no]
EC-XML:[yes/no]
EC-XML-CODESET:[yes/no]
EC-XML-CODESET-CONVERSION:[yes/no]

EC-XML-COUNT:[yes/no]
EC-XML-DOCUMENT-TYPE:[yes/no]
EC-XML-IMPLICIT-CLOSE:[yes/no]
EC-XML-INVALID:[yes/no]
EC-XML-NAMESPACE:[yes/no]
EC-XML-RANGE:[yes/no]
EC-XML-STACKED-OPEN:[yes/no]

Compiling with the `-debug` compiler configuration flag enables all of the exception checks.

When not compiling with `-debug`, you can enable specific exception checks by setting the associated compiler configuration flag to `yes` in the compiler configuration file.

exclusivelock: [yes/no]

Default is `exclusivelock:no`.

Causes all files with no `LOCK MODE` clause in their `SELECT` statement to be declared implicitly as `LOCK MODE IS EXCLUSIVE`. For details on other compiler flags related to the treatment of `LOCK MODE`, see Guidelines for modifying default handling of the `LOCK MODE`.

exec-check: [yes/no]

Default is `exec-check:no`.

When set to `yes`,
Used with `-syntax-only`, checks the `EXEC SQL/CICS/DLI` syntax.

exit-program-forced:[yes/no]

Default is `exit-program-forced:yes`

Affects behavior of the `EXIT PROGRAM` statement.

When set to `yes`, the execution of the `EXIT PROGRAM` statement is forced, creating compatibility with non-ISO-standard behavior used by some COBOL compilers.

Note that ISO standards state that the `EXIT PROGRAM` should be ignored if the currently running program was not `CALL`'ed by a COBOL main program. However, many proprietary COBOL compilers never ignore the `EXIT PROGRAM` statement

When set to `no`, the `EXIT PROGRAM` statement behaves according to ISO standards, which state that the `EXIT PROGRAM` should be ignored if the currently running program was not `CALL`'ed by a COBOL main program.

expand-exec-copy:[yes/no]

Default is `expand-exec-copy:no`.

When set to `yes`, causes the compiler to expand COBOL `COPY` statements inside `EXEC ... END-EXEC` blocks. This applies to both `EXEC SQL` and `EXEC CICS` blocks.

expand-sql-include:[yes/no]

Default is `expand-sql-include:no`.

When set to yes,

Used with `-E`, expands 'EXEC SQL INCLUDE <File name> END-EXEC' in the `-E` output.

external-link: <function name>

Causes [function name] to be declared as an external non-COBOL symbol.

Usage:

external-link: functionName

Using the compiler configuration flag `external-link` will cause the code "CALL 'function-name'" to generate more efficient code.

external-mapping:[yes/no]

Default is `external-mapping:yes`

Allows files declared as EXTERNAL to be resolved using environment variables.

When set to yes, file names of files declared as EXTERNAL are resolved at run time using environment variables. See 'filename-mapping' for detail about name-mapping.

fast-figurative-move: [yes/no]

Default is `fast-figurative-move:yes`.

When set to yes,

Optimizes the performance of the MOVE of figurative constants (default).fast-op: [yes/no]

Default is `fast-op:no`

When set to yes,

Enables the runtime to use faster operations when manipulating data items declared as USAGE DISPLAY or USAGE COMP-3. Fast-op is set to yes by default when using either the `-O` or `-O2` compiler flag.

fcdreg: [yes/no]

Default is `fcdreg:no`.

When set to yes,

Corresponds to the FCDREG compiler directive. This compiler flag provides functionality in applications that are using the EXTFH file system interface. For more detail see the description of the `-ffcdreg` compiler flag.

fdclear:[yes/no]

Default is `fdclear:no`

When set to yes,
Causes the record to be INITIALIZED after each WRITE.

file-auto-external:[yes/no]

Default is `file-auto-external:yes`.

The `-file-auto-external` compiler flag affects the way that the compiler treats variables describing file-names for files described as EXTERNAL. For more detail, see the description of the `-ff-auto-external` compiler flag.

filename-mapping:[yes/no]

Default is `filename-mapping: yes`

Allows file names to be resolved at runtime using environment variables.

When set to yes, file names are resolved at run time, checking for 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"

When set to no, the value of the ASSIGN clause is treated as the file name.

first-tab-width:[any integer]

Default is `first-tab-width:8`.

Allows the user to define the size of the first tab.

Notes- This is done to support the RM/COBOL standard that defines the first tab as 8 characters, and following tabs as 6 characters. To mimic this standard, you would set `tab-width: 6`, and `first-tab-width: 8`.

flat-extfh: <DRIVER NAME>***flat-extfh-lib: <library to use for this extfh driver>***

The configuration file flags `flat-extfh` and `flat-extfh-lib` enable the usage of EXTFH drivers for Sequential and Relative files.

Note: When used, they should be used together.

fold-copy-lower: [yes/no]

Default is `fold-copy-lower:no`.

When set to yes,

Folds COPY file names to lower case.

fold-copy-upper:[yes/no]

Default is `fold-copy-upper:no`.

When set to yes,
Folds COPY file names to upper case.

fp-opt:[yes/no]

Default is `fp-opt:no`

When set to yes,
Causes COMP-2 operations to be inlined in C, and maximizes the use of the CPU Floating Point unit.

free-thread-safe-data: [yes/no]

Default is `free-thread-safe-data:no`

When set to yes,
When used with `-thread-safe` compiler flag, frees data in modules after a CANCEL event that is not a FULL-CANCEL

fstatus-map:[cit-status] = [custom-status]

Allows COBOL-IT to map file IO statuses to custom values.

Example: `fstatus-map: 22 = 67`

In the example above, when the file system returns status 22, the value will be translated to 67 before returning to the application. This flag may be repeated as many as needed for all expected translation.

full-cancel:[yes/no]

Default is `full-cancel: no`

Affects the behavior of the CANCEL statement which, by default, causes a “Logical Cancel” to be implemented.

When set to yes,

All CANCEL statements performed in the running program cause a “Full Cancel” to be implemented.

This behavior can also be achieved by setting the runtime environment variable `COB_FULL_CANCEL` to Y.

Clarifications on the difference between a “Logical Cancel” and a “Full Cancel”:

In a “Logical Cancel”, the Working-Storage Section is reset to its initial values. Working-Storage initial values are the values set the first time the module was loaded in memory.

In a “Full Cancel”, the Working-Storage Section is reset to its initial values, and the module binary is unloaded, if possible, from memory. Unloading the module binary from memory is only possible

if the module is not being used in another region/thread, and it has been loaded by a CALL STATEMENT (not by a preload of a shared library).

functions-all:[yes/no]

Default is functions-all:no.

When set to yes,
Allows use of intrinsic functions without the FUNCTION keyword.

Note : The mf.conf configuration file, which contains compiler configuration flags designed to match Micro Focus default behaviours, includes the setting functions-all: yes. Micro Focus users that require the use of the FUNCTION keyword should re-set this option to functions-all: no.

functions-all-intrinsic:[yes/no]

Default is functions-all-intrinsic:no.

When set to yes,
Allows use of intrinsic functions without the FUNCTION keyword

gcc:[yes/no]

Default : no for all UNIX platforms
Default: yes for Linux platforms

When set to yes,
Generates gcc-compliant C code. The `-fgcc` compiler flag is enabled when `COB_CC=gcc`.

gcc-O-bug:[yes/no]

Default is gcc-O-bug:no.

When set to yes,
When using `-O`, some versions of gcc generate incorrect code. This bug is avoided by using the `-gcc-O-bug` compiler flag.

gcc-bug:[yes/no]

Default is gcc-bug:no.

When set to yes,
When using a gcc compiler on very large source files, the gcc compiler could enter an infinite loop. This bug is avoided by using the `-gcc-bug` compiler flag.

gcc-goto:[yes/no]

Default is gcc-goto:no

When set to yes,
Generates gcc-computed goto code. The `-fgcc-goto` compiler flag is enabled when using the `-fgcc` compiler flag, or when `COB_CC=gcc`

gcos-mode:[yes/no]

Default is `gcos-mode:no`.

When set to yes,
Causes the compiler to more closely emulate GCOS operations.

gen-xdd:[yes/no]

Default is `gen-xdd:no`.

When set to yes,
Causes the compiler to generate a c-TreeACE .xdd file

global-typedef:[yes/no]

Default is `global-typedef:yes`.

When set to yes,
Causes TYPEDEFS to be GLOBAL for all nested program. If not set, TYPEDEFS are local to the current program.

ibm-listing-macro:[yes/no]

Default is `ibm-listing-macro:yes`.

When set to yes,
Enables IBM listing extensions (TITLE, SKIP1/2/3, EJECT ...) (default)

ibm-mainframe:[yes/no]

Default is `ibm-mainframe:no`

When set to yes, causes the compiler and runtime to operate in an IBM Mainframe compatible mode.

ibm-sync:[yes/no]

Default is `ibm-sync:yes`.

When set to yes,
Applies SYNC attribute to group item if first elementary field is described with the SYNC attribute. (default).

When set to no,
The SYNC attribute is not applied to a group item if the first elementary field in the group item is described with the SYNC attribute.

identifier-length:<max-length>

Default is `identifier-length: 0`

The identifier-length compiler configuration flag allows compatibility to be achieved with other

compilers, as regards the maximum length of a variable name.

When set to 0

COBOL-IT has no limit on the length of a variable name, and will never test for this, or generate an error.

When set to a positive, non-zero value

COBOL-IT will generate an error if the length of a variable name exceeds the the max-length named by identifier-length.

To achieve compatibility with the IBM COBOL compiler, for example, which generates an error when a variable name exceeds 30 characters, set `identifier-length: 30`.

ignore-global-in-local-storage: [yes/no]

Default is `ignore-global-in-local-storage:no`

ignore-with-rollback: [yes/no]

Default is `ignore-with-rollback:no`

implicit-init: [yes/no]

Default is `implicit-init:no`

When set to yes,

Initializes the COBOL runtime system at runtime start-up.

include-main: [yes/no]

Default is `include-main:no`

When set to yes,

Causes main symbol to be included in module object when compiled with `-c`.

For more details, see the documentation of the `-finclude-main` compiler flag.

incomplete-subscript:[yes/no]

Default is `incomplete-subscript: yes`

Affects the behavior of MOVEs to table items.

When set to yes:

Consider a data item declared as `01 TABLE OCCURS 10 PIC X`.

The phrase `MOVE SPACE TO TABLE` is equivalent to `MOVE ALL SPACE TO TABLE`.

index-optimize:[yes/no]

Default is `index-optimize:no`

Improves performance where indexes in tables are evaluated and USAGE DISPLAY fields are used as indexes. In these cases, the index values are cached in a C integer field to improve performance.

As an example, consider the usage of the follow code:

```
...  
01 IxdA PIC 999 USAGE DISPLAY.  
01 IxdB PIC 999 USAGE DISPLAY.  
...  
MOVE FLD-ARRAY(IxdA, ixdB) TO ...  
MOVE FLD-ARRAY(IxdA, ixdB) TO ...  
IF (FLD-ARRAY(IxdA, ixdB) ...
```

When set to yes, benefits the performance of these MOVE and IF statements by keeping the actual value of the index in a binary C cache, thus avoiding conversion from DISPLAY (or COMP-3) to a binary value each time the index is evaluated in a statement. `index-optimize` is set to yes by default when using the `-O` or `-O2` compiler flags.

indirect-redefines:[yes/no]

Default is `indirect-redefines: yes`

Enables the REDEFINES of a variable that REDEFINES another variable.

When set to yes, redefines of a redefining variable is allowed.

When set to no, redefines of a redefining variable is not allowed.

Example of a redefines of a redefining variable:

```
10 VALA PIC 99.  
10 VALB PIC XX REDEFINES VALA.  
10 VALC PIC 99 REDEFINES VALB.
```

initcall:<program-name>

The `initcall` compiler configuration flag names modules to be called immediately before the first statement of a program is executed.

initialize-fd:[yes/no]

Default is `initialize-fd:no`.

When set to yes,

Causes records declared in the FD section to be initialized when the program is initially loaded in memory.

initialize-filler:[yes/no]

Default is `initialize-filler: no`

Affects whether the INITIALIZE statement does or does not initialize fields declared as FILLER.

When set to yes,

When performing an INITIALIZE statement, FILLER fields defined in elementary fields are initialized to the default value or to data of the VALUE clause (depending on *initialize-to-value*).

When set to no,

When performing a INITIALIZE statement, FILLER fields are left unchanged.

initialize-opt:[yes/no]

Default is `initialize-opt:no`

When set to yes,

Optimizes the implementation of the initial field initialization at runtime startup and the execution of the INITIALIZE statement by grouping field initializations wherever possible.

initialize-pointer:[yes/no]

Default is `initialize-pointer: yes`

Affects whether elementary fields described as USAGE INDEX, USAGE POINTER, USAGE PROGRAM POINTER are initialized by the INITIALIZE verb.

When set to yes,

Elementary fields described as USAGE INDEX, USAGE POINTER, USAGE PROGRAM-POINTER are initialized by the INITIALIZE verb.

When set to no,

Elementary fields described as USAGE INDEX, USAGE POINTER, USAGE PROGRAM-POINTER are not initialized by the INITIALIZE verb.

initialize-to-value:[yes/no]

Default is `initialize-to-value: no`

Affects the behavior of the INITIALIZE statement.

When set to yes, when performing an INITIALIZE statement :

If a data element contains a VALUE clause, then the INITIALIZE statement causes the data element to be INITIALIZE'd to the value defined in it's VALUE clause.

If a data element contains no VALUE clause, then the INITIALIZE statement causes the data element to be INITIALIZE'd using the defaults.

isam-extfh: <DRIVER NAME>

isam-extfh-lib: <library to use for this extfh driver>

The configuration file flags `isam-extfh` and `isam-extfh-lib` enable the usage of EXTFH drivers for Indexed ISAM files.

Note: When used, they should be used together.

Usage: `isam-extfh: <DRIVER NAME>`

`isam-extfh-lib: <library to use for this extfh driver>`

keep-copy-statement:[yes/no]

Default is `keep-copy-statement:no`.

When set to yes,
Causes listings and preprocessed files to keep COPY statements.

keep-org-src-line:[yes/no]

Default is `keep-org-src-line:yes`.

When set to yes,
For use with the integrated pre-processor (`-preprocess`). Causes errors to be reported on the original source line.

keep-unused:[yes/no]

Default is `keep-unused:yes`

When set to yes,
Causes memory to be allocated for the field tree of level-01 and level-77 data items that are declared which contain sub-fields and in which none of these sub-fields is used.

When set to no,
Causes memory to not be allocated for the field tree of level-01 and level-77 data items that are declared which contain sub-fields and in which none of these sub-fields is used.

key-dup-always-22:[yes/no]

Default is `key-dup-always-22: no`

Forces the runtime to return a file status of 22 on a duplicate key condition.

When set to yes,
When adding a record to an INDEXED file that is open in OUTPUT mode, if a duplicate key condition is detected, the runtime will return a file status 22.

keycompress: [integer between 0 and 9]

Default is `keycompress:0`.

- To enable key compression in VBISAM, you must add :

```
$SET KEYCOMPRESS "x"
```

before the SELECT statement.

“x” is a numeric integer literal, ranging from 1 to 9. Higher values produce better compression, at the expense of performance. A setting of 1 provides the best performance, a setting of 9 the best compression.

All users of an indexed file created with compression must re-compile their programs with the KEYCOMPRESS compiler directive and use the same compression setting.

To disable compression (after having enabled it)

```
$SET NOKEYCOMPRESS
```

larger-redefines-ok:[yes/no]

Default is `larger-redefines-ok: yes`

Allows a larger variable to redefine a smaller variable

When set to yes,
Larger variables may redefine smaller variables.

When set to no,
Larger variables may not redefine smaller variables.

line-seq-dos:[yes/no]

Default is `line-seq-dos: no`

Determines the end-of-record delimiter used on line sequential files.

When set to yes,
Line sequential records are terminated by a CR/LF

When set to no,
Line sequential records are terminated by a CR

line-seq-mf:[yes/no]

Default is `line-seq-mf: yes`

Affects the storage of bytes with values less than 0x20 in line sequential files.

When set to yes,

Line sequential files preface bytes with values less than 0x20 with 0x00. Thus, when you write the bytes X"1F", X"20" to a line sequential file, they will be recorded in the file as : X"00" X"1F" X"20".

line-seq-notrunc:[yes/no]

Default is `line-seq-notrunc:no`

The `line-seq-notrunc` compiler configuration flag affects the behavior of the runtime when a line sequential record is read that is longer than the declared record length.

When set to yes, the part of the record that exceeds the declared record length is returned as the next record.

When set to no, (the default), the record is truncated.

line-seq-recording-mode:[yes/no]

Default is `line-seq-recording-mode: no`

Affects the interpretation of the RECORDING MODE F clause on line sequential files.

When set to yes,
The record is written to disk including all trailing spaces.

When set to no,
Trailing spaces are removed from the record before writing it to disk.

line-seq-unix:[yes/no]

Default is `line-seq-linux:no`.

Obsolete

link-only: [yes/no]

Default is `link-only:no`.

When set to yes,
Causes the `main()` symbol to not be generated, when used with `-x`. For use when the program entry point (`main`) is provided by an external object or library.

listing-sources: [yes/no]

Default is `listing-sources:no`.

When set to yes,
Informs the compiler that source is the result of program listing option (`-t <file>`).

local-storage-guard: 8 (internal use only)

Default is `local-storage-guard: 8`

This setting is for internal use only, and should not be changed.

loosy-comment[yes/no]

Default is `loosy-comment:no`

When set to yes, the compiler allows a `*` in column 8 to be used to mark a comment.

ls-expand-tab:[yes/no]

Default is `ls-expand-tab:yes`

Implements the expansion of the tab character in line sequential files in a manner that is compatible with Micro Focus COBOL.

When set to yes,
Tab characters not preceded by a 0 are expanded to spaces. The tab stop is fixed to 8 characters.

ls-ignore-record-size:[yes/no]

Default is `ls-ignore-record-size:no`

Determines whether or not trailing spaces are stripped from a line sequential file before writing it to disk.

When set to yes,

The RECORD SIZE clause for LINE SEQUENTIAL files is ignored. Trailing white spaces are stripped before WRITEing a record to disk.

When set to no,

The RECORD SIZE clause for LINE SEQUENTIAL files is used to determine the number of bytes written to disk on a WRITE statement. Trailing spaces are not stripped.

ls-utf16:[yes/no]

Default is `ls-utf16:no`

Activates support for UTF16 storage, which affects the reading and writing of line sequential files.

When set to yes,

Line sequential file are read/stored in UTF16 format. When UTF16 Storage is active, end-of-record is read/written in UTF16 format in accordance with the setting of the `-utf16-le` compiler flag.

When used, the `-utf16-le` compiler flag causes fields declared as PIC N to be stored as UTF16-LE (Little Endian). Note that by default, fields declared as PIC N are stored as UTF16-BE (Big Endian). X'000A' is the end-of-record for BE storage and X'0A00' is the end-of-record for LE storage.

Note that the `ls-utf16` compiler configuration flag may be set in source using the meta comment `$SET` before the `SELECT` statement:

```
$SET LSUTF16           Activate UTF16 storage
$SET NOLSUTF16        Deactivate UTF16 storage
```

main:[yes/no]

Default is `main:yes`.

When set to yes,

Generates `main()` symbol when used with `-x` (default).

main-as-object:[yes/no]

Default is `main-as-object:yes`.

When set to yes,

Generates `main()` symbol as object not in library (unix only) (default)

mainframe-vb:[yes/no]

Default is `mainframe-vb:no`.

When set to yes,
Causes WRITES and READs of Variable Blocked files to assume formats compatible with the Mainframe Z/OS COBOL Format.

makesyn: oldvalue=newvalue

Provides a way to make a reserved word a synonym for another reserved word. The first word, represented by “oldvalue” becomes a synonym of the second word, represented by “newvalue”. A common usage is to make COMP a synonym of COMP-5.

The entry:

makesyn: comp=comp-5 corresponds to the compiler flag `-makesyn`, used as follows:

```
>cobc -makesyn comp=comp-5 hello.cbl
```

The `-makesyn oldvalue=newvalue` compiler flag provides compatibility with the `MAKESYN` directive.

A COBOL verb or field-name may be used as “old-value”. The COBOL-IT compiler will replace all instances of this “old-value” with the “new-value” when compiling.

CAUTION- While this provides an equivalent capability to the implementation of the `MAKESYN` directive in other COBOLs, the order of the parameters is reversed. COBOL-IT requires that the “old-value” be listed first, and followed by the “new-value”.

makesyn-patch-preprocess: [yes/no]

Default is `makesyn-patch-preprocess: no`

When set to yes,

Causes the `makesyn` compiler flag to change the output of pre-processed files

manuallock: [yes/no]

Default is `manuallock: no`.

When set to yes,

Causes all files with no `LOCK MODE` clause in their `SELECT` statement to be declared implicitly as `LOCK MODE` is `MANUAL` unless a `SHARING` clause in the `SELECT` statement or in the `OPEN` statement indicates otherwise. For details on other `-f` compiler flags related to the treatment of `LOCK MODE`, see Guidelines for modifying default handling of the `LOCK MODE`.

max-literal-expand: 32 (internal use only)

Default is `max-literal-expand: 32`

This setting is for internal use only, and should not be changed.

mem-info: [yes/no]

Default is `mem-info: no`.

When set to yes,
Enables Dump of Working-Storage when runtime aborts.

mem-info:yes is enabled by the `-g` compiler flag and by the `-debug` compiler flag.

mfcomment: [yes/no]

Default is `mfcomment:yes`.
Treats lines with '*' or '/' in column 1 as comments.

mf-compat-parser: [yes/no]

Default is `mf-compat-parser:yes`.

When set to yes,
Causes COBOL-IT to match certain Micro Focus behaviors. These include :
Parsing of line continuation characters
Relaxed syntax check on RECORD CONTAINS phrase in the FD
Allowing level-66 and level-88 data names to have the same name as a paragraph or section.

When set to no,
The compiler generates errors in these situations.

mf-ctrl-escaped-parser: [yes/no]

Default is `mf-ctrl-escaped-parser:yes`.

When set to yes,
Syntax parser is MF compatible with control character escaped by 0 (default).

mf-file-optional:[yes/no]

Default is `mf-file-optional:no`

When set to yes,
Affects the file-status codes returned on files declared as OPTIONAL and OPEN in EXTEND.

The `mf-file-optional:yes` compiler configuration flag causes files declared as OPTIONAL and OPEN EXTEND to return file-status code "05" if the file is created and file-status code "00" if the file exists. The `mf-file-optional:yes` compiler flag improves consistency with Micro Focus behaviors.

When set to no,
Files declared as OPTIONAL and OPEN EXTEND return file-status code "00" in both the case where the file did not exist, and was created, and the case where the file did exist.

The `-mf-file-optional` compiler configuration flag corresponds to setting `-fmf-file-optional` compiler flag.

mf-gnt: [yes/no]

Default is `mf-gnt:no`.

When set to yes,

Causes shared objects generated by the compiler to be created with the `.gnt` extension.

Note that the generated object IS NOT compatible with the `.gnt` objects produced by Micro Focus. This option is only used to reduce change in existing compilation scripts by causing object code to be generated with the same extensions.

mf-hostnumcompare:[yes/no]

Default is `mf-hostnumcompare:no`.

The `mf-hostnumcompare` compiler configuration entry provides compatibility with Micro Focus in cases where the `HOST-NUMCOMPARE` directive is used. The `mf-hostnumcompare` compiler configuration entry affects comparisons of `USAGE DISPLAY` numeric data items when one of the numeric data items in the comparison contain non-numeric data.

When set to yes, the field containing numeric data redefined as an alphanumeric item of the same length, and this redefined data item is compared with the non-numeric value of the other numeric data item.

When set to no (the default), the contents of the field containing numeric data are moved to an intermediate alphanumeric data item that is the same size as the field containing nonnumeric data before the comparison is performed. The content of this intermediate alphanumeric item is then compared to the non-numeric value of the other numeric data item.

mf-int:[yes/no]

Default is `mf-int:no`.

When set to yes,

Causes shared objects generated by the compiler to be created with the `.int` extension.

Note that the generated object IS NOT compatible with the `.int` objects produced by Micro Focus. This option is only used to reduce change in existing compilation scripts by causing object code to be generated with the same extensions.

mf-relativefile :[yes/no]

Default is `mf-relativefile :no`

Allows for compatibility with Micro Focus relative files.

When set to yes, the COBOL-IT runtime assumes the Micro Focus format for relative files for both `READ` and `WRITE` operations.

When set to yes, the end-of-record marker for relative files is consistent with the setting of the compiler configuration flag `line-seq-dos`.

When `line-seq-dos:yes`, the end of record setting is `CR/LF`

When `line-seq-dos:no`, the end of record setting is `LF`

module-load-priority:[yes/no]

Default is `module-load-priority: no`

Affects the manner in which the runtime resolves the target of a CALL statement, so that shared libraries are searched before linked symbols.

When set to `no`,

To resolve “myprog” in the phrase CALL “myprog”, the default behavior is first to look for the symbol “myprog” in the linked library. Then if the symbol is not found, the runtime searches for a shared library (`myprog.cit`, `myprog.dll` (Windows) or `myprog.so` (Linux/UNIX)) and searches for the symbol `myprog` in the shared library.

When set to `yes`,

The search order is reversed. The runtime first looks for the shared library and if the symbol is not found, it then looks into the linked symbols.

The same effect can be achieved at runtime by setting the `COB_LOAD_PRIORITY` environment variable as follows:

```
export COB_LOAD_PRIORITY=1
```

The compiler option `-fthread-safe` enables this option.

module-name-entry: [yes/no]

Default is `module-name-entry:yes`.

When set to `yes`,

Generates source module as alternate entry (default)

module-uppercase: [yes/no]

Default is `module-uppercase:no`.

When set to `yes`,

Causes the output file name to be created in upper-case, when used with the `-m` compiler flag.

move-all-edited: [yes/no]

Default is `move-all-edited:no`.

When set to `yes`,

Causes MOVE ALL "X" TO an edited field to take care of the picture.

move-high-low-to-displaynumeric [error/zero/value]

Default is `move-high-low-to-displaynumeric:value`

The `move-high-low-to-displaynumeric` compiler configuration flag affects the behavior of MOVE HIGH-VALUES TO (usage display numeric data item) *and* MOVE LOW-VALUES TO (usage display numeric data item).

When set to

error: trigger error at compilation "Invalid MOVE statement"

zero: move zeroes to display numeric item

value: move high- or low-values to display numeric item

move-picx-to-pic9:[cit / mf50 / mf40 / mvs / raw / iso / none]

Default is `move-picx-to-pic9: none`

Defines the runtime behaviour when moving alphanumeric (PIC X(n) USAGE DISPLAY), to non-signed display numeric data items (PIC 9(n) USAGE DISPLAY).

When set to "cit"

The target field is filled with '0's. Then the source's alphanumeric field value is checked for validity as a numeric value. If the check passes, the value is transferred truncated according to the rules associated with a COBOL MOVE from display numeric to display numeric.

When set to "mf50"

The source field value is copied, with truncation on the high end of the data item. No validation is done.

When set to "mf40"

The source field is copied, right-justified. Upper half-bytes are replaced with 0x30. As a result, a MOVE PIC X(3) VALUE "ABC" to a PIC 9(3) field stores "123" in the PIC 9(3) field. This is the Micro Focus 4.0 default behavior.

When set to "mvs"

The source field value is copied with truncation at the high end of the data item. Then the last character is checked. If that last character is a valid EBCDIC sign character , that last character is replaced by the corresponding value (0-9).

When set to "raw"

The data is copied with no conversion and no validation, and with RIGHT-JUSTIFY. It is equivalent to a MOVE of a PIC X field to a PIC X field where both PIC X fields are RIGHT-JUSTIFIED.

When set to "iso" (when displaynumeric-mf50:yes)

When MOVEing a PIC X field padded with SPACES to a PIC 9 field, the padding SPACES are converted to 0s. As an example, when MOVEing a PIC XX VALUE "2" field to a PIC 99 field, the PIC 99 field would store the target as "02".

move-spaces-to-comp3:[error/space/zero]

Default is `move-spaces-to-comp3: zero`

The `move-spaces-to-comp3` compiler configuration flag affects the behavior of the compiler and runtime when SPACES are moved to a COMP-3 data item.

The default setting is “zero”. When set to zero, the compiler does not issue an error. At runtime ZEROES are moved to the COMP-3 data item.

When set to space, the compiler does not issue an error. At runtime, SPACES are moved to the COMP-3 data item.

When set to error, the compiler issues an error : Error: Invalid MOVE statement.

move-spaces-to-displaynumeric:[yes/no/error]

Default is `move-spaces-to-displaynumeric: error`

Enables MOVE SPACES to a numeric USAGE DISPLAY data item (PIC 9(n) USAGE DISPLAY)

When set to yes,

The MOVE of SPACES to a PIC 9 USAGE DISPLAY field is allowed.

move-to-group-separated:[yes/no]

Default is `move-to-group-separated: no`

Affects the behaviour of a MOVE of SPACES or ZEROES to a group item, as when compiling the instruction:

```
MOVE SPACE TO GRP-XXX  
MOVE ZERO TO GRP-XXX
```

where GRP-XXX is a group item.

When set to yes,

The MOVE is generated for each field and sub-field of the group

When set to no,

The MOVE treats the group-item as a single alphanumeric data item.

name:[any string]

Default value: "COBOL-IT"

Purely used for purposes of commentary.

`no-cbl-error-proc: [yes/no]`

Default is `no-cbl-error-proc: no`

When set to yes,

Prevents the execution of CBL_ERROR_PROC.

no-realpath: [yes/no]

Default is `no-realpath:no`.

When set to yes,
Causes file names to NOT be extended to a fully qualified path.

By default, when processing file names, the compiler retrieves the fully qualified path (from the root) and processes the compilation using that extended name. That full name is also stored as the source file name for debugging purposes.

non-ibm-5.2-syntax: [(ok or yes)/(error or no)/warning]

Default is `non-ibm-5.2-syntax: ok`

When set to ok or yes, non-ibm 5.2 syntax is not checked, and no action is taken.

When set error or no, non-ibm 5.2 syntax generates a compilation error.

When set to warning, non-ibm-5.2 syntax generates a warning, but the compilation continues.

nostrip: [yes/no]

Default is `nostrip:no`.

When set to yes,
Causes objects and object and executable files to NOT be stripped.
Stripping an object or an executable is the action of removing system level debugging information

notrunc: [yes/no]

Default is `notrunc:no`

When set to yes,
Causes truncation of binary fields to NOT be made according to the PICTURE clause while performing intermediate computations.

not-reserved:[any reserved word]

Allows the removal of words from the reserved word list.

As an example, if you want the reserved word INCLUDE to not be considered a reserved word by the compiler, make an entry in the compiler configuration file as follows:

```
not-reserved:INCLUDE
```

null-param: [yes/no]

Default is `null-param:no`

When set to yes,
Causes an extra NULL pointers to be passed as the last argument on CALL statements.

numeric-compare: [yes/no]

Default is `numeric-compare:no`.

When set to yes,

Causes the comparison of a numeric field with a PIC X field to interpret the value of the PIC X field using its numeric value.

numeric-group: [yes/no/warning]

Default is `numeric-group:yes`

Determines the behavior of the compiler when an IS/IS NOT NUMERIC clause is applied to a group item.

When set to yes, the compiler supports the syntax, and the clause does not produce a compiler error.

When set to no, compiler aborts, with the error message:

Error: IS NUMERIC not allowed on group

When set to warning, the compiler continues, but produces a warning:

Warning: IS NUMERIC not allowed on group

numval-validate: [yes/no]

Default is `numval-validate: no`

When set to yes,
Validates argument 1 of the NUMVAL function.

obj-cit: [yes/no]

Default is `obj-cit:no`.

When set to yes, causes compiled object to be generated with a cit extension instead of .dll (windows) or .so (unix/linux). The COBOL-IT runtime recognizes the .cit extension as an executable extension. The default behavior of the CALL statement has been changed, so that in a CALL "myprog" statement, the runtime will look first for a compiled object with a .cit extension, before searching for a .dll (Windows) or a .so (Linux/UNIX).

odo-slide: [yes/no]

Default is `odo-slide: no`

When set to yes,

Affects data items that appear after a variable-length table in the same record; that is, after an item with an OCCURS DEPENDING clause, but not subordinate to it.

If the odo-slide compiler flag is set, these items always immediately follow the table, whatever the current size of the table. Note that the internal addresses of these data items change as the table's size changes.

If the odo-slide compiler flag is not set, these items have fixed addresses, and begin after the end of the space allocated for the table at its maximum length.

Note : The mf.conf configuration file, which contains compiler configuration flags designed to match Micro Focus default behaviours, includes the setting `odo-slide : no`.

If you are using the mf.conf configuration file, and your code depends on `odo-slide` being set to yes, please add `odo-slide : yes` to your configuration file or add `-fodo-slide` to your compilation command line.

optimize-move: [yes/no]

Default is `optimize-move: yes`

When set to yes,

Causes MOVE operations to be optimized that are performed by the INITIALIZE verb in cases where target fields are USAGE DISPLAY NUMERIC, or USAGE NATIONAL.

Causes MOVE operations to be optimized by `-fmem-info` where the source and target fields have identical declarations.

When set to no,

The optimizations are deactivated.

optimize-move-call:[yes/no]

Default is `optimize-move-call:no`

When set to yes,

Causes MOVE operations to be optimized by pre-selecting the internal runtime library routines used for the MOVE when possible. `optimize-move-call` is set to yes by default when using either the `-O` or `-O2` compiler flag.

optional-file: [yes/no]

Default is `optional-file: no`

When set to yes,

Causes all SELECT statements that do not specify OPTIONAL or NOT OPTIONAL to be considered OPTIONAL.

pack-comp-4:[yes/no]

Default is `pack-comp-4: no`

Affects the storage of COMP-4 data items.

When set to yes,

COMP-4 fields are stored in the minimal possible amount of data space. This is the equivalent of applying “`binary-size = 1 - - 8`” to the COMP-4 data items.

When set to no,

COMP-4 is considered equivalent to USAGE COMPUTATIONAL with memory storage calculated accordingly.

perform-osvs:[yes/no]

Default is `perform-osvs: no`

Enhances compatibility with OSVS COBOL PERFORM statements

The \$SET perform type settings of COB370, ENTCOBOL, OSVS and VSC2 are emulated with the setting perform-osvs:yes.

When set to yes,

The exit point of any currently executing perform is recognized if reached.

PERFORM statements with the same exit point can be nested to a depth of two (one inner and one outer). If they are nested deeper, they do not return correctly. The end of a section is regarded as a separate point from the end of its last paragraph.

The example below is included to illustrate a possible consequence of setting perform-osvs to 'yes', where an infinite loop results that would otherwise be avoided.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. TEST-PERFORM.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 SW PIC X VALUE '1'.  
01 ABORT-PRG PIC X(3) VALUE 'NO'.  
PROCEDURE DIVISION.  
BEGIN.  
    DISPLAY "BEGIN".  
    PERFORM A THRU A-EX UNTIL ABORT-PRG = 'YES'.  
    DISPLAY "END".  
    STOP RUN.  
C.  
    DISPLAY "I AM IN C".  
A.  
    IF SW = '1'  
        PERFORM D THRU D-EX  
    ELSE  
        MOVE 'YES' TO ABORT-PRG.  
A-EX.  
    EXIT.  
B.  
    DISPLAY "I AM IN B".  
    MOVE '0' TO SW.  
    GOTO C.  
D.  
    IF SW = '1'  
        GOTO B.  
D-EX.  
    EXIT.
```

When perform-osvs is set to 'no', the source above produces the following result :

```
BEGIN  
  
I AM IN B  
  
I AM IN C  
  
END
```

prepro_cut_line: [yes/no]

Default is `prepro_cut_line: yes`.

When set to yes,

When preparing file for preprocess, cuts output to 72 columns (default)

When set to no,

When preparing file for preprocess, does not cut output to 72 columns.

pretty-display:[yes/no]***printer-crlf:[yes/no]***

Default is `printer-crlf: no`.

When set to yes,

Files declared with ASSIGN TO PRINTER file names are generated with compatibility for DOS printers. This will change the End Of Record to CR/LF (instead of LF)

profiling:[yes/no]

Default is `profiling: no`.

When set to yes,

The compiler generates paragraph profiling code. The output produced by the profiler includes separate counts for CPU and real elapsed times. For more details on using COBOL-IT's built in Profiler, see Guidelines for use of Profiler below.

protect-linkage:[yes/no]

Default is `protect-linkage: no`.

When set to yes,

Generates code at the entry point of a program containing a USING xxx clause.

This allows for the passing of parameters that are NULL pointers. In these cases, where NULL pointers are passed, the compiler creates a "fake" field of the same definition in WORKING-STORAGE, and substitutes it as a reference for the parameter. Doing this will avoid a SIGVEC error if NULL pointers passed through linkage are targets of a READ or WRITE statement.

quote:[any single character]

Default is `quote: "`

Defines the value of the QUOTE reserved word

raw-by-value: [yes/no]

Default is `raw-by-value: yes`.

When set to yes,
CALL BY VALUE [PIC X Fld] does not convert [PIC X Fld] to numeric COMP-5 (default).

raw-compare: [yes/no]

Default is `raw-compare:yes`.

raw-pic9-display: [yes/no]

Default is `raw-pic9-display:no`.

When set to yes,
DISPLAY PIC 9(X) (no sign, no decimal) as it is in memory.

read-at-end-mf:[yes/no]

Default is `read-at-end-mf:no`

Affects the interpretation of the AT END/NOT AT END clauses used with a READ statement.

When set to yes,

For INDEXED files, when compiling a READ statement:

If an AT END and/or NOT AT END clause is defined in a READ statement and no NEXT or PREVIOUS clause is specified, then the NEXT clause is implied.

read-into-copy: [yes/no]

Default is `read-into-copy:no`

When set to yes,

Causes the READ INTO statement to COPY data rather than performing a MOVE.

ready-trace:[y/n]

Default is `ready-trace: no`

Enables paragraph tracing between READY TRACE and RESET TRACE procedural COBOL statements.

When set to yes,

In the interval between the READY TRACE and RESET TRACE statements, paragraph tracing output is written to the console in the format:

PROGRAM-ID: [program-id]: [paragraph name]

recmode-f:[yes/no]

Default is `recmode-f: no`.

When set to yes,

Causes all unspecified RECORDING MODE clauses to be interpreted as RECORDING MODE F.

recmode-osvs:[yes/no]

Improves compatibility with OSVS COBOL, as regards the RECORDING MODE phrase.

Default is `recmode-osvs: no`

When set to yes,

Compatibility with OSVS COBOL is improved, as regards the RECORDING MODE phrase.

recmode-v:[yes/no]

Default is `recmode-v: no`.

When set to yes,

Causes all unspecified RECORDING MODE clauses to be interpreted as RECORDING MODE V.

record-depending-iso:[yes/no]

Default is `record-depending-iso:no`

When set to yes,

The record-depending-iso compiler configuration flag causes a RECORD DEPENDING ON <FIELD> clause to be handled in an ISO-compatible manner. More specifically, the `rec0rd-depending-iso:yes` compiler configuration flag causes files declared with a RECORD DEPENDING ON <FIELD> clause, without any FROM or TO value, to assume a FROM and TO value of the maximum record size.

When set to no,

The clause is ignored.

redefine-identifier: [error / warning / ok]

Default is `redefine-identifier: ok`

The redefine-identifier compiler configuration entry has been added to the “Dialect Features” component of the compiler configuration file. The redefine-identifier compiler configuration flag affects compiler behavior when ambiguous identifiers exist in the source code.

As an example, if your Working-Storage Section contained two identical declarations:

```
77 field-1 pic x(10).
```

```
77 field-1 pic x(10).
```

When set to ok

The compiler does not generate an error.

When set to warning

The compiler generates a warning

When set to error

The compiler generates an error

region0: [yes/no]

Default is `region0:no`.

When set to yes,

Causes the program to always switch to region 0 when executing. Setting `region0:yes` lets you specify that the module will always execute in region 0 even if called from another region. When called from another region, the module will switch to region 0 on entry and switch back to the calling region at exit.

relativefile-bigendian:[yes/no]

Default is `relativefile-bigendian:no`

When set to yes,

Causes the record header of relative files to be stored in BigEndian format.

relax-bounds-check:[yes/no]

Default is `relax-bounds-check: yes`

Affects bounds-checking in reference-modified notations.

When set to yes,

In a reference-modified expression like `mystring (x:length)`, the variable `length` is not checked for an out-of-bounds condition.

relax-level-hierarchy:[yes/no]

Default is `relax-level-hierarchy: yes`

Affects compiler's handling of non-matching level numbers.

When set to yes,

Non-matching level numbers are allowed.

relaxed-syntax-check:[yes/no]

Default is `relaxed-syntax-check: yes`

Affects strictness of syntax checking rules applied by the compiler.

When set to yes,

Relaxed syntax checking rules are applied by the compiler.

replace-additive:[yes/no]

Default is `replace-additive:no`.

When set to yes,

Allows for the use of the REPLACE ADD verb, which has the effect of nesting a REPLACE statement inside an existing REPLACE statement. Nested REPLACE statements are executed before outer REPLACE statements in COBOL-IT's precompile phase. Note that a REPLACE stack can be cleared with the REPLACE OFF statement.

return-opt:[yes/no]

Default is `return-opt:no`.

When set to yes,
Generates optimized PERFORM return code. The `-freturn-opt` compiler flag is ignored when using the `-fgcc` compiler flag.

round-fp:[yes/no]

Default is `round-fp:no`.

When set to yes,

Controls the way COMP-1 or COMP-2 are “moved” into non-COMP-1 or COMP-2 target fields when the target field has fewer decimal places than the source field. If the `-fround-fp` compiler flag is used, the value is rounded to the number of decimal of the target field. Otherwise, the value is truncated.

rtncode-size: <integer>

Default is `rtncode-size:0`.

When set to a non-zero value, sets the size, and memory alignment of the return-code register.
Possible values are:

2	2 bytes. Align on 2-byte boundary
4	4 bytes. Align on 4-byte boundary
8	8 bytes. Align on 8-byte boundary

rw-after-preprocess:[yes/no]

Default is `rw-after-preprocess:no`

When set to yes,

Causes SPCRW2 to be run after the `-preprocess` script. By default SPCRW2 is run before the `-preprocess` script.

rw-mode-nopf:[yes/no]

Default is `rw-mode-nopf:no`

When set to yes,

Is equivalent to setting MODE NOPF for a Report Section report. MODE NOPF causes the Report Writer to emulate an external print driver that does not generate printer control characters, like FF. When set to yes, the report file is written as a standard LINE SEQUENTIAL file.

rw-mode-nopf-dos:[yes/no]

Default is `rw-mode-nopf-dos: no`

When set to yes,

Is equivalent to setting MODE NOPF for a Report Section report. MODE NOPF causes the Report Writer to emulate an external print driver that does not generate printer control characters, like FF. When set to yes, the report file is generated with fixed-length lines that are padded with SPACES and use the CR/LF record delimiter.

safe-linkage:[yes/no]

Default is `safe-linkage: no`.

When set to yes,

Generates code at the entry point of a program containing a USING xxx clause.

This allows for the omission of parameters. Doing this will avoid a SIGVEC being returned by the debugger when all linkage parameters are not provided.

screen-exceptions:[yes/no]

Default is `screen-exceptions: no`

Mimics the behavior of the environment variable COB_SCREEN_EXCEPTIONS.

When set to yes,

Causes the runtime to behave as if the environment variable COB_SCREEN_EXCEPTIONS=Y

Enables use of the Page Up, Page Down, Up Arrow, and Down Arrow keys on Field-level ACCEPT statements. Also enables use of the Esc key, if the environment variable COB_SCREEN_ESC=Y. When these keys are pressed, the COBOL-IT runtime will return the CRT Status values as described in the table below. Note that Page Up, Page Down, Up Arrow, and Down Arrow are enabled by default when ACCEPTing a Screen:

Condition	Key Pressed	Cit Value returned to CRT-STATUS-VAR
On ACCEPT <Screen> or On ACCEPT <Field> where the environment variable COB_SCREEN_EXCEPTIONS=Y or the compiler configuration flag screen-exceptions:Y	Page Up	2001
	Page Down	2002
	Up Arrow	2003
	Down Arrow	2004
When COB_SCREEN_ESC=Y	Escape Key	2005

	Print	2006
--	-------	------

screen-raw-keys:[yes/no]

Default is `screen-raw-keys: no`

Mimics the behavior of the environment variable `COB_SCREEN_RAW_KEYS`.

When set to yes,

Causes the runtime to behave as if the environment variable `COB_SCREEN_RAW_KEYS=Y`

Note- The “raw keys” are the Home, End, Insert, Delete, and Erase EOL keys. When one of these keys is pressed, the COBOL-IT runtime will return CRT Status values as described in the table below:

Condition	Key Pressed	Cit Value returned to CRT-STATUS-VAR
Where the environment variable <code>COB_SCREEN_RAW_KEYS=Y</code> Or the compiler configuration flag <code>screen-raw-keys:Y</code>	Home	2007
	End	2008
	Ins	2009
	Del	2010
	Erase EOL	2011

sequential-line:[yes/no]

Default is `sequential-line:no`.

When set to yes,

Causes all non-qualified SEQUENTIAL files to be declared as LINE SEQUENTIAL. Files declared as RECORD SEQUENTIAL are not affected.

share-all-autolock:[yes/no]

Default is `share-all-autolock:no`.

When set to yes,

Causes all files with a SHARE WITH ALL clause in their SELECT statement to be declared implicitly as LOCK MODE IS AUTOMATIC. For details on other `-f` compiler flags related to the treatment of LOCK MODE, see Guidelines for modifying default handling of the LOCK MODE.

share-all-default:[yes/no]

Default is `share-all-default:no`.

When set to yes,
Causes all files to be declared implicitly as `SHARE WITH ALL`.

For details on other compiler flags related to the treatment of `LOCK MODE`, see Guidelines for modifying default handling of the `LOCK MODE`.

share-all-manulock:[yes/no]

Default is `share-all-manulock:no`.

When set to yes,
Causes all files with a `SHARE WITH ALL` clause in their `SELECT` statement to be declared implicitly as `LOCK MODE IS MANUAL`.

For details on other compiler flags related to the treatment of `LOCK MODE`, see Guidelines for modifying default handling of the `LOCK MODE`.

sign-ascii:[yes/no]

Default is `sign-ascii:no`.

When set to yes,
Corresponds to the `SIGN "ASCII"` directive. Causes numeric `DISPLAY` items that include signs are interpreted according to the ASCII sign convention. (default on ASCII machines)

sign-ebcdic:[yes/no]

Default is `sign-ebcdic:no`.

When set to yes,
Corresponds to the `SIGN "EBCDIC"` directive. Causes numeric `DISPLAY` items that include signs are interpreted according to the EBCDIC sign convention. (default on EBCDIC machines)

sign-leading:[yes/no]

Default is `sign-leading:no`

When set to yes,
Makes `SIGN IS LEADING` the default.

sign-separate:[yes/no]

Default is `sign-separate:no`.

When set to yes,
Makes SIGN IS SEPARATE the default.

signed-comp6-as-comp3:[yes/no]

Default is `signed-comp6-as-comp3: no`
Affects treatment of data items described as comp-6.

When set to yes,
Signed comp-6 data items are treated as comp-3 data items.

simple-trace:[yes/no]

Default is `simple-trace:no`.

When set to yes,
Generates trace output at runtime for executed SECTION/PARAGRAPHS.

source-location:[yes/no]

Default is `source-location:no`.

When set to yes,
Generates source location code, enabling information to be dumped on source location when the runtime aborts.

`source-location:yes` is enabled by the `-g` compiler flag and by the `-debug` compiler flag.

split-debug-mark:[yes/no]

Default is `split-debug-mark:yes`.

When set to yes,
DEBUG marks respect max 72 characters (default)

spzero:[yes/no]

Default is `spzero: no`

Affects the handling of space characters in numeric fields.

When set to yes ,
Space characters moved to NUMERIC USAGE fields are converted to '0's.

stack-check:[yes/no]

Default is `stack-check:no`.

When set to yes,

Enables stack checking debug function. The stack checking debug function allows the user to trace back through the stack of calling programs to the currently running line of source in a program.

`stack-check:yes` is enabled by the `-g` compiler flag and by the `-debug` compiler flag.

static-call:[yes/no]

Default is `static-call:no`.

When set to yes,

Causes static C function calls to be generated for the CALL statement.

This implies that all CALL'ed programs are C function that are linked with the current program. When using the `-fstatic-call` compiler flag, no external dynamic resolution is performed at run-time.

static-link:[function-name]

Improves performance of the CALL statement for statically linked routines.

The string value represents a symbol to link statically when encountered as an argument of a CALL. This improves the performance of the call.

Mainly used for linked C function libraries

Example:

```
static-link: myfunc
```

Note:

Oracle symbols are provided in the file "oracle.symb", and can be included in your compiler configuration file as follows:

```
include "oracle.symb"
```

Tuxedo symbols are provided in the file "tuxedo.symb", and can be included in your compiler configuration file as follows:

```
include "tuxedo.symb"
```

sticky-linkage:[yes / no / fixed / variable]

Default is `sticky-linkage: no`

Affects allocation of linkage section items not listed as parameters

When set to yes, or fixed

The called program fills in any parameter not actually passed by the caller.

When set to no, or variable

Non-parameter linkage-section items remain allocated between invocations.

strict-compare-low: [yes/no]

Default is `strict-compare-low:no`

When set to yes,

Causes display of numeric variables containing low values not equal to zero or spaces when compared.

strict-record-contains:[yes/no]

Default is `strict-record-contains:yes`

The `strict-record-contains` compiler configuration entry affects the handling of the `RECORD CONTAINS xx CHARACTERS` clause in file systems using then `EXTFH` interface, when the actual record size has fewer characters than are named in the clause.

When set to yes (the default):

COBOL-IT sends the number of characters stated in the clause as both `MIN-RECORD-SIZE` and `MAX-RECORD-SIZE`.

When set to no:

COBOL-IT detects the smaller actual record size, and passes it through as the `MIN-RECORD-SIZE`, while passing the number of characters stated in the clause as the `MAX-RECORD-SIZE`.

Note that the `strict-record-contains` compiler configuration flag has no effect on `VBISAM` files, as they do not use the `EXTFH` interface.

synchronized-double-word-bound:[yes/no]

Default is `synchronized-double-word-bound:no`

Affects the alignment of data items declared as `USAGE BINARY`, which also contain the `SYNC` clause. The default behavior is consistent with the behaviors of the IBM and Micro Focus compilers. Specifically, when the `SYNC` clause is used, binary fields are aligned either on 2- or 4-byte boundaries, depending on the size of the field.

Fields of 1-2 bytes in size are aligned on 2-byte boundaries.

Fields greater than 2-bytes in size are aligned on 4-byte boundaries.

When set to yes,

Binary fields are aligned on 2- or 4- or 8-byte boundaries, depending on the size of the field. This was the default COBOL-IT behavior prior to version 3.7.9.

Fields of 1-2 bytes in size are aligned on 2-byte boundaries.

Fields of 3-4 bytes in size are aligned on 4-byte boundaries.

Fields greater than 4 bytes in size are aligned on 8-byte boundaries.

synchronized-propagate-to-occurs: [yes/no] (Internal Use Only)

Default is `synchronized-propagate-to-occurs: yes`

This setting is for internal use only, and should not be changed.

synchronized-propagate-to-occurs-with-group-size: [yes/no] (Internal Use Only)

Default is `synchronized-propagate-to-occurs-with-group-size: no`

This setting is for internal use only and should not be changed.

syntax-only:[yes/no]

Default is `syntax-only:no`.

When set to yes.

Performs syntax error checking only. Output is limited to results of syntax check.

syntax-support:[ok / archaic / obsolete / skip / ignore / unconformable / error]

Allows users to choose for the compiler to support, warn, or reject listed phrases which are included in some COBOL dialects. The meaning of the different syntax support options:

ok	Accept the dialect
archaic, obsolete, unconformable	Emit warning messages
skip	Construction is ignored
ignore	Produces a warning message
error	Produces a syntax error
unconformable	Produces an 'unconformable' error message

The phrases entered by default in the compiler configuration file with syntax support options :

```
# 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:  ok
synchronized-clause:      ok
goto-statement-without-name: obsolete
stop-literal-statement:   obsolete
support-debugging-line:   obsolete
padding-character-clause: obsolete
next-sentence-phrase:    archaic
eject-statement:          skip
entry-statement:          ok
move-noninteger-to-alphanumeric: error
```

<code>odo-without-to:</code>	<code>ok</code>
<code>move-spaces-to-displaynumeric:</code>	<code>error</code>
<code>synchronized-clause:</code>	<code>ok</code>
<code>synchronized-clause-on-group:</code>	<code>ok</code>
<code>synchronized-group-align-size:</code>	<code>skip</code>
<code>synchronized-occurs-align-size:</code>	<code>skip</code>

tab-width:[integer]

Default value: 8

Affects interpretation of tab characters in source code.

tally-register:[yes/no]

Default is `tally-register: yes`

Affects the creation of the internal TALLY register.

When set to `no`,

The creation of the TALLY register is disabled.

text-column:[integer]

Default is `text-column: 72`

Defines number of columns used for COBOL text

For Fixed source format only. Causes the compiler to ignore characters after the column marked by “`text-column + 6`”.

thread-safe:[yes/no]

Default is `thread-safe:no`.

When set to `yes`,

Generates thread-safe executables. For more details, see Guidelines on operating in a thread-safe environment below.

trace:[yes/no]

Default is `trace:no`.

When set to `yes`,

Generates trace output at runtime, listing the SECTION/PARAGRAPH names as they are executed.

trace-ts: [yes/no]

Default is `trace-ts:no`.

When set to `yes`,

Causes trace output to include timestamp.

trace-upon-systout:[yes/no]

Default is `trace-upon-sysout:no`.

When set to yes,

Causes trace output to be written upon `sysout`, instead of the default writing upon `syserr`.

traceall:[yes/no]

Default is `traceall:no`.

When set to yes,

Generates trace output at runtime, listing SECTION/PARAGRAPH/STATEMENTS names as they are executed.

trap-unhandled-exception:[yes/no]

Default is `trap-unhandled-exception:no`.

When set to yes,

Is useful in cases where certain EC compiler configuration file flags are set to yes, yet ON EXCEPTION/ONSIZED ERROR/ON OVERFLOW language is not present in the COBOL program. In these cases, using the `-ftap-unhandled-exception` compiler flag causes the information made available to the user to be enhanced when the program aborts.

For more details, see the documentation of the `-ftap-unhandled-exception` compiler flag.

truncate-listing:[yes/no]

Default is `truncate-listing:no`.

When set to yes,

Causes output of the `-t <file>` compiler flag to be truncated at column 76

unstring-use-move:[yes/no]

Default is `unstring-use-move:no`.

Affects the behavior of the UNSTRING verb.

When set to yes, if the target of an UNSTRING INTO operation is described as PIC 9, then the operation will be performed using a MOVE operation instead of raw copy operation. Then rules defined by the `move-picx-to-pic9` compiler configuration flag are used for conversion.

use-defaultbyte:[yes/no]

Default is `use-defaultbyte: no`

Enables defaultbyte use

When set to yes,

The defaultbyte compiler configuration file flag may be used.

utf16-le:[yes/no]

Default is `utf16-le:no`.

Determines whether fields declared as PIC N are stored in UTF16-LE (Little Endian) or UTF16-BE (Big Endian) format.

When set to yes,

Causes fields declared as PIC N to be stored in UTF16-LE (Little Endian) format.

When set to no,

Causes fields declared as PIC N are stored in UTF16-BE (Big Endian) format.

utf-8:[yes/no]

Default is `utf-8:no`.

When set to yes,

Instucts the compiler that the source file, and literals are UTF-8 encoded. The utf-8 compiler configuration file flag can be used with, or without the codepage compiler configuration flag.

If the utf-8 compiler configuration flag is used and the codepage compiler configuration flag is not specified, then codepage utf-8 is assumed.

If, however, you wish to compile your source with another codepage (for example, the LATIN1 codepage), you should explicitly include that codepage declaration:

```
codepage: latin1
```

validate-dep-on: [yes/no]

Default is `validate-dep-on:no`.

When set to yes,

Causes the value of DEPENDING ON to be checked at runtime.

validate-odo:[yes/no]

Default is `validate-odo:yes`.

validate-only:[yes/no]

Default is `validate-only:no`.

When set to yes, causes the compilation of source to ignore all EXEC statements, and produce no compiled objects. Compiler errors are produced, and can be captured in an error file, (using `-err`, for example.

value-of-id-priority: [yes/no]

Default is `value-of-id-priority:no`

When set to yes, the literal or data element that is the target of the VALUE OF FILE-ID clause in the FD overrides target of the ASSIGN clause for the file.

When set to no (the default), this setting is ignored.

value-size-is-auto: [yes/no]

Default is `value-size-is-auto: no`

When set to yes, the CALL ..USING BY VALUE : default SIZE IS clause will be AUTO (current default is SIZE IS 4).

variable-rec-pad-mf:[yes/no]

Default is `variable-rec-pad-mf: no`

Affects padding rules applied to variable length sequential records.

When set to yes, variable size RECORD SEQUENTIAL records (REC MODE 'V') are stored with padding characters at the end to ensure that the next record starts on a 4 byte boundary.

vbisam: [yes/no]

Default is `vbisam:no`

When set to yes, causes VBISAM to be used as the default indexed file system.

vms-error-handler: [yes/no]

Default is `vms-error-handler:no`

When set to yes,

Causes the default IO error handler to always abort (emulation of VMS behavior).

when-compiled-function-all :[yes/no]

Default is `when-compiled-function-all: no`

When set to yes, and when functions-all: yes, then WHEN-COMPILED can be called without using the FUNCTION keyword.

wnone:[yes/no]

Default is `wnone: no`

When set to yes, causes warnings turned on by default to be turned off.

xparse-event:[yes/no]

Default is `xparse-event:no`

When set to yes, causes the XML PARSE statement to generate START-OF-DOCUMENT and END-OF-DOCUMENT XML-EVENTS.

zero-length-trim:[yes/no]

Default is `zero-length-trim:yes`.

When set to `yes`, allows the `LENGTH` intrinsic function to return a zero-length of a string that has been operated on by the `TRIM` intrinsic function.

Compiler Environment Variables

The following environment variables are used by **cobc**, the COBOL-IT compiler, at compilation time.

COB_AR <program>

Default is `ar` on Unix/Linux-based systems and `lib.exe` on Windows-based systems. Designates archive program to be used by **cobc**.

COB_ARFLAGS <ar flags>

Designates archive flags used by **cobc**.

COB_CC <program>

Designates C compiler used by **cobc**.

COB_CFLAGS <cc flags>

Default is `-I$COBOLITDIR/include` on Unix/Linux-based systems and `-I%COBOLITDIR%\include` on Windows.

Designates C compiler flags used by **cobc**.
If defined, replaces the default settings used by **cobc** as the C compiler flags.

COB_CONFIG_DIR<directory>

Default is `$COBOLITDIR/share/COBOL-it/config` on Unix/Linux-based systems, and `%COBOLITDIR%/config` on Windows-based systems.

Designates the location of the compiler configuration file.

COB_CONSOLE_CP=<codepage-id>

The **COB_CONSOLE_CP**=<codepage-id> runtime environment variable, when defined, causes the `DISPLAY UPON CONSOLE/DISPLAY UPON SYSOUT` phrases to convert the codepage defined using the `-fcodepage` compiler flag to <codepage-id>.

As an example (on a Linux X Console):

```
>cobc -x myprog.cob -fcodepage latin1
export COB_CONSOLE_CP=UTF-8
./myprog
```

Will correctly display literals encoded in latin1 on the console.

COB_COPY_DIR<directory>

Default is \$COBOLITDIR/share/COBOL-it/copy on Unix/Linux-based systems, and %COBOLITDIR%\copy on Windows-based systems.

Designates the location of COPY files.

COB_EXTRA_FLAGS

Designates C compiler flags added when -O2 option is used or when -O is used without prior -Os.

COB_LDADD <ld flags>

Sets an additional link command to be used by cobc.

Example:

```
COB_LDADD= -L/my/ownlibpath/ -lmylibs; export COB_LDADD
```

COB_LDFLAGS <ld flags>

Designates linker flags used by cobc. If defined, cobc does not compute the needed linker flags.

COB_LIBS <libs>

Default is -L/opt/COBOL-it/lib -lcob -lgmp -ldb on Unix/Linux-based systems, and /LIBPATH:C:\COBOL\COBOLIT\lib\ libcobit_dll.lib on Windows-based systems

Designates C compiler library flags used by cobc.

COB_OPTIMIZE_FLAG<cc flags>

Default is -O2

Designates C compiler flags added when -O option is used.

Warning : On some platforms, using an optimization level greater than 2 can produce an unstable program.

COB_OPTSIZE_FLAG=[optimization flag]

Default is -O2

Designates flag to be used in the C compilation phase for the purpose of optimization.

Usage: To cause the -Os flag to be used by the C compiler


```
export COB_OPTSIZE_FLAG=-Os
```

COB_STDUNIX <1/0>

Default is 0

When set to 1, on Windows-based systems, indicates that end-of-line is marked by LF, not CR/LF as is the default on Windows-based systems.

COB_SUNSTUDIO12=[Y/N]

Default is N.

On sun SPARC Solaris, when using C compiler Sun Studio 12. you must set the runtime environment variable "COB_SUNSTUDIO12" to "Y" to avoid receiving the warning:

```
cc: Warning: -xarch=generic64 is deprecated, use -m64 to create 64-bit programs
```

This environment variable setting is required when creating 64-bit programs. Otherwise, 4-byte structure alignments could cause your program to abort at runtime with the error: "Bus Error (core dumped)".

COBCPY <directory list>

Designates colon ':'-separated list of pathes (Unix,Linux) or semi-colon ';' -separated list of pathes (Windows) where cobc looks for COPY files.

COBCTMP=<directory>

Default is /tmp on UNIX/Linux-based systems and the local user's AppData\Local\Temp directory on Windows-based systems.

Designates the directory where temporary files are stored.

Note- The default TMPDIR setting is returned in the command "cobc -V".

Note- When compiling with the -save-temps compiler flag, temporary files are stored in the current directory.

COBITOPT=[string of command-line compiler flags]

Stores command-line compiler flags, for use with cobc. COBITOPT is designed to be compatible with the MF environment variable COBOPT.

COBOPT=[string of command-line compiler flags]

Stores command-line compiler flags, for use with `cobmf`. COBOPT is designed to be compatible with the MF environment variable COBOPT.

COBOLITDIR=<directory>

Default is `/opt/COBOL-it` on UNIX/Linux-based systems and `C:\COBOL\COBOLIT` on Windows-based systems.

Names the directory in which COBOL-IT is installed.

TMPDIR or TMP=<directory>

Default is `/tmp` on UNIX/Linux-based systems and the local user's `AppData\Local\Temp` directory on Windows-based systems.

Designates the directory where temporary files are stored.

Note- The default TMPDIR setting is returned in the command “`cobc -V`”.

Note- When compiling with the `-save-temps` compiler flag, temporary files are stored in the current directory.

COBOL-IT Runtime Options

Usage: `cobcrun [param ...] PROGRAM`

COBOL-IT runtime parameters

--checkpoint <file>

Sets checkpoint filename

--console, -c

Creates a new console

--debug, -d

Suspends and waits for debugger

--debug, -d --remote -r

Same as `-debug` but uses a separate file for events

--help, -h

Prints this help

--reload

Reloads checkpoint

--version, -V

Display runtime version

COBOL-IT runtime environment variables

The following environment variables are used by cobcrun, or by the COBOL-IT native executable at run time.

COB_CALL_CASE=xul [*where x=exact match, u=uppercase, l=lowercase*]

COB_LOAD_CASE=xul [*where x= exact match, u=uppercase, l=lowercase*]

COB_CALL_CASE and COB_LOAD_CASE should be used together, and affect the behavior of the CALL statement, as regards algorithms applied when locating the target of the CALL statement.

When COBOL-IT executes a CALL “[Symbol]” statement, the default behavior for the runtime is: to look first in memory for:

- 1- an exact case match of “[Symbol]”,
- 2- “[Symbol]” in upper case,
- 3- “[Symbol]” in lower case.

If “[Symbol]” is not found in memory, the runtime will then look for a filename with:

- 1- an exact case match of “[Symbol]”,
- 2- “[Symbol]” in upper case,
- 3- “[Symbol]” in lower case.

The first match is used.

COB_CALL_CASE and COB_LOAD_CASE provide the user with control over this lookup process, as follows:

COB_CALL_CASE=xul controls the behavior of “[Symbol]” lookup in memory.

COB_LOAD_CASE=xul controls the behavior of “[Symbol]” lookup in a filename on disk.

Where the letters xul represent three letters, each of which may be set to Y or N, to set whether or not the x (exact match check), u (uppercase check) or l (lower-case check) is performed.

Default settings are :

COB_CALL_CASE=YYY

COB_LOAD_CASE=YYY

An example of a usage, in which only exact case matching would be applied for a symbol in memory would be, while upper-case and lower-case matching for the filename is preserved:

```
export COB_CALL_CASE=YNN
```

```
export COB_LOAD_CASE=YYY
```

COB_CONSOLE_CP=<code page>

Sets the Windows console code page to <code page>. COB_CONSOLE_CP is available in Windows only.

COB_CURRENT_DATE

The COB_CURRENT_DATE runtime environment variable can be used to control the results of the ACCEPT FROM DATE YYYYMMDD statement. COB_CURRENT_DATE holds a date in the format “yyyy-mm-dd”.

Usage:

In Windows:

```
>set COB_CURRENT_DATE=2015-01-31
```

....

In Linux/Unix:

```
>export COB_CURRENT_DATE=2015-01-31
```

```
77 the-date PIC 9(8).
```

....

```
ACCEPT the-date FROM DATE YYYYMMDD .
```

```
DISPLAY the-date line 10 col 10.
```

Alternatively, you can use the SET verb to set COB_CURRENT_DATE, as follows:

```
SET ENVIRONMENT “COB_CURRENT_DATE” TO “2011-01-31”.
```

In each of these cases the variable the-date would be returned as 20110131 instead of the current date, as would be expected if COB_CURRENT_DATE were not set.

Note: The ACCEPT FROM DATE statement now checks once to determine whether the COB_CURRENT_DATE environment variable is set. If it is set, then the value is stored in cache, and is retrieved by the ACCEPT FROM DATE statement. If it is not set, then COB_CURRENT_DATE is not re-checked, and the ACCEPT FROM DATE statement applies its default behavior of retrieving the date from the system.

COB_DEBUG_ALLUSER=1

The COB_DEBUG_ALLUSER environment variable, when set to 1, and when defined before

running a COBOL program, causes the pipes that are created by the debugger to communicate with cobcdb to have an attribute mask of 777, which provides Read/Write attributes for all users.

For the case where `cob_init(. . .)` has already been called, the same effect can be achieved by calling:

```
cob_debug_acl_alluser(rtd,1);
```

This will also ensure that the pipes that are created by the debugger to communicate with cobcdb have Read/Write attributes for all users.

Note- Usage of `COB_DEBUG_ALLUSER`, and/or `COB_DEBUG_TMP` may be indicated if you receive this error message opening a pipe created by the debugger:

```
Error opening /[path]/debug_xxx.cit for write (13: Permission denied)
```

COB_DEBUG_ID=<debug-id>

Defines a numeric ID that may be used to catch the program instead of the process id (PID). When defined, a debugger may attach to the program using the `COB_DEBUG_ID`.

Before running the program in debug, define the environment variable `COB_DEBUG_ID`, for example:

```
export COB_DEBUG_ID= <debug-id>
```

where `<debug-id>` is an integer.

Attach to the program using the `debug-id`:

For details on the debug attach functionality, see the documentation of the `C$DEBUG` library routine.

Then at run time you must define the runtime environment variable:

```
COB_DEBUGDB=<DebugDB-name>
```

When compiling with `-debugDB=<DebugDB-name>`, the compiler will modify the way debugging information is stored at compile-time. Instead of storing the metadata in the compiled object, the metadata will be stored in an SQLite3 database named by `<DebugDB-name>`.

The `COB_DEBUGDB=<DebugDB-name>` runtime environment variable allows the runtime to locate this file during a debugging session, and use the debugging information. Currently only 1 database may be use at a time. As a consequence, the user must use the same database for all of the programs in his run unit.

COB_DEBUG_MODULES=<program-id1>:<program-id2>....

`COB_DEBUG_MODULES` is a list of program-ids, in which the entries are separated by a colon character “:”. Adding the program-id of a program in your application to the list of `COB_DEBUG_MODULES` causes the debugger to break at the entry of that program.

This provides an alternative way to attach the debugger to a running process in cases where programs do not contain calls to “C\$DEBUG”, or where you do not have access to the remote attach interface in the Developer Studio.

COB_DEBUG_STARTUP_FILE=<filename>

The console debugger cobcdb can locate source files that have been re-located after compilation using the COB_DEBUG_STARTUP_FILE runtime environment variable and invoking the replace debugger command.

The COB_DEBUG_STARTUP_FILE runtime environment variable is set to the name and location of a file containing any number of commands that are executed when cobcdb is started.

Open <filename> and place the “replace” commands on separate lines:

```
replace /opt/cobol-it-64/samples/test:/opt/cobol-it-64/samples/src  
replace ?
```

Save <filename> and export the COB_DEBUG_STARTUP_FILE environment variable.

```
export COB_DEBUG_STARTUP_FILE=<filename>
```

To locate a source file that has been moved, and associate it with an object compiled for debug, use the 'replace' debugger command, which changes the path to the source file.

The syntax is as follows: replace <oldprefix> : <newprefix>

The replace debugger command allows you to replace the location where the source files associated with the program being debugged are stored.

The replace debugger command replaces any prefix of the full pathname, so the command replace /dirA : /dirB will allow any program that was originally compiled in /dirA/dev/sources to have its source stored in /dirB/dev/sources.

Subsequent commands are stacked, so when typing two more commands as follows :

```
replace /dirC : /dirD  
replace /dirE : /dirF
```

you will end up with a list of three possible replacements. Only the first matching replacement will be executed.

Further usages include:

replace <no arguments>	Resets the list, removing active replacements
replace ?	Produces a list of active replacements.

Note that replace only affects the output of the list command.

The list debugger command allows you to expand the source you can see inside the console

debugger as you execute your debugger commands.

COB_DEBUG_TMP=<directory>

Default is /tmp.

COB_DEBUG_TMP control where the files and pipes created by the debugger are stored.

The runtime debugger uses named pipes to communicate. These are pipes with a file name, and by default, they are located in /tmp. You may relocate them by defining the COB_DEBUG_TMP environment variable.

This variable can be set in the login script of the user used to connect the remote debugger, as defined in the remote connection tab of the Developer Studio. This variable can also be set in the local runtime environment. It does not need to be set in both locations. If it is set in both locations, the settings should be identical, or the settings will be ignored, and the default value of /tmp will be used.

The COB_DEBUG_TMP environment variable may be required when debugging remotely attaching to a running process using the Developer Studio Remote System Explorer. This could be the case if COBOL-IT user:group that the program is running under have different permissions on pipe files created by default in the /tmp directory than the user:group of the user running the debugger. This problem is resolved by use of the COB_DEBUG_TMP environment variable which can be used to relocate the named pipes used by the runtime debugger into a directory in which the permissions of the user:group running the program and the permissions of the user:group running the debugger are the same.

Note- Usage of COB_DEBUG_ALLUSER, and/or COB_DEBUG_TMP may be indicated if you receive this error message opening a pipe created by the debugger:

Error opening /[path]/debug_xxx.cit for write (13: Permission denied)

COB_DISPLAY_PRINTER=<filename>

The COB_DISPLAY_PRINTER runtime environment variable defines a file that is appended to for each “DISPLAY UPON PRINTER” statement. Each “DISPLAY UPON PRINTER” statement OPENS this file, WRITES to the file, and then CLOSES the file.

COB_DUMP=<filename>

Designates the filename used to dump memory information, when a program aborts that has been compiled with the -fmem-info compiler flag.

When set to N/NO, no dump is produced. If COB_DUMP is not set, then the memory information is dumped to the file named by the COB_ERROR_FILE environment variable.

If COB_ERROR_FILE is also not set, memory information is written to stderr.

The output of this dump has been enhanced by adding the memory address of each field.

As an example:

WORKING-STORAGE

RETURN-CODE [6AEF4438] = +000000000

TALLY [6AEF4440] = +000000000

SORT-RETURN [6AEF4448] = +000000000

NUMBER-OF-CALL-PARAMETERS [6AEF4458] = +000000000

COB_ERROR_FILE=<filename>

Designates the filename used to receive all runtime error messages that would otherwise be sent to stderr. When writing an error message, the runtime will create the specified filename if it does not exist, and will append to it if it does exist.

COB_EXTFH=<EXTFH Entry>

Defines the EXTFH handle name to be used for all COBOL files.

At run time, if the COB_EXTFH environment variable is defined (and no additional variables are included to specify what library to load), the runtime looks for lib\$(COB_EXTFH) on UNIX and \$(COB_EXTFH)_dll.dll on Windows in the COBOL-IT installation directory and all directories indicated in the COB_LIBRARY_PATH.

To use D-ISAM through EXTFH :

>export COB_EXTFH=disamextfh

Libdisamextfh.so will be found and loaded into the CIT distribution directory (COBOLITDIR).

The D-ISAM indexed file engine includes a check utility: dcheck. Dcheck is available on all Linux, UNIX and Windows systems.

To use BerkeleyDB through EXTFH:

>export COB_EXTFH=bdbextfh

To use VBISAM through EXTFH*:

>export COB_EXTFH=vbisamextfh

*In version 3.10, VBISAM is the default file system, and does not require setting the COB_EXTFH environment variable. However, in the future releases of Version 4.x, D-ISAM will replace VBISAM as the default file system, and use of VBISAM will require this setting.

To use C-Tree through EXTFH:

COB_EXTFH=CTEXTFH; export COB_EXTFH

COB_EXTFH_LIB=/opt/mytools/lib/liba.so:/opt/mytools/lib/libb.so; export COB_EXTFH_LIB

These file systems can be used with all COBOL-IT tools including CitSORT.

COB_EXTFH_FLAT=<EXTFH Entry>

Defines the EXTFH handle name to be used for all COBOL flat files (SEQUENTIAL non-indexed and RELATIVE).

Example:

```
COB_EXTFH_FLAT=MyEXTFH; export COB_EXTFH_FLAT
```

COB_EXTFH_INDEXED=<EXTFH Entry>

Defines the EXTFH handle name to be used for all COBOL indexed files

Example:

```
COB_EXTFH_INDEXED=BDBEXTFH; export COB_EXTFH_INDEXED  
COB_EXTFH_LIB=/opt/COBOL-it/lib/libbdbextfh.so; export COB_EXTFH_LIB
```

COB_EXTFH_LIB=[list of shared libraries]

Iterates a colon –separated ':' list of external shared libraries on (Unix, Linux) or a semi-colon-separated ';' list of external shared libraries on Windows. The list of shared libraries is preloaded at start-up, and provides the reference to the COBOL-IT runtime, for locating the EXTFH handler defined by COB_EXTFH, COB_EXTFH_INDEXED and COB_EXTFH_FLAT.

Example:

```
COB_EXTFH_LIB=libbdbextfh.so:libdb-4.7.so; export COB_EXTFH_LIB
```

COB_FILE_CASE=[UPPER/LOWER]

Forces all file names to be converted to upper/lower case.

COB_FILEMAP_CASE=[UPPER/LOWER]

Forces the runtime to convert all literals that are the target of ASSIGN EXTERNAL clauses that are associated with environment variables to upper/lower case before trying to resolve the name of the environment variable associated with the file.

Example:

```
SELECT myfile      ASSIGN EXTERNAL "FileA"....
```

When there is no setting of COB_FILEMAP_CASE, the runtime looks for an environment variable named FileA.

When COB_FILEMAP_CASE=UPPER, the runtime looks for an environment variable named FILEA.

When COB_FILEMAP_CASE=LOWER, the runtime looks for an environment variable named filea.

COB_FILE_PATH=[PATH]

Designates the path to data files used by the application. COB_FILE_PATH is prepended to datafile names by the runtime as it works to resolve filenames and locations.

Example:

```
COB_FILE_PATH=./data; export COB_FILE_PATH
```

COB_FILE_RELATIVE_MF=Y

Default is N

When set to Y, the COBOL-IT runtime assumes the Micro Focus format for relative files for both READ and WRITE operations.

COB_FILE_TRACE=[Y/N]

Default is N

When set to Y, file tracing information is output to the file named by COB_ERROR_FILE, which includes information on how the runtime resolves file names on OPEN, and also status codes returned from unsuccessful file i-o operations. The COB_FILE_TRACE runtime environment variable is evaluated when the OPEN statement is executed by the runtime. Changes to the COB_FILE_TRACE runtime environment variable can be made during the runtime session.

COB_FULL_CANCEL=[Y/N]

Default is N.

Affects the behavior of the CANCEL statement which, by default, causes a “Logical Cancel” to be implemented.

When set to Y,

All CANCEL statements performed in the running program cause a “Full Cancel” to be implemented.

This behavior can also be achieved by setting the compiler configuration file flag:

```
full-cancel:Y.
```

Clarifications on the difference between a “Logical Cancel” and a “Full Cancel”:

In a “Logical Cancel”, the Working-Storage Section is reset to its initial values. Working-Storage initial values are the values set the first time the module was loaded in memory.

In a “Full Cancel”, the Working-Storage Section is reset to its initial values, and the module binary is unloaded, if possible, from memory. Unloading the module binary from memory is only possible if the module is not being used in another region/thread, and it has been loaded by a CALL STATEMENT (not by a preload of a shared library).

COB_KEY_DUP_ALWAYS_22=[Y/N]

Default is N.

When set to Y, causes an error code of 22 to be returned if an attempt is made to add a record that results in a duplicate key error condition.

By default, COBOL-IT returns an error code of 21 if an attempt is made to add a record that results in a duplicate key error condition.

COB_LIBRARY_PATH=[PATH]

Designates the path to shared objects that will be executed with cobcrun. COB_LIBRARY_PATH is prepended to shared object names that are CALL'ed by the runtime as it works to resolve called object names and locations.

COB_LOAD_CASE=[UPPER/LOWER]

Controls the case transformation when looking for an external program in a CALL statement.

When set to LOWER, the CALL'ed name is converted to lower case.

When set to UPPER, the CALL'ed name is converted to upper case.

Otherwise, the name remains unchanged.

COB_LOAD_PRIORITY=[Y/N]

Default is N.

When set to no,

To resolve "myprog" in the phrase CALL "myprog", the default behavior is first to look for the symbol "myprog" in the linked library. Then if the symbol is not found, the runtime searches for a shared library (myprog.cit, myprog.dll (Windows) or myprog.so (Linux/UNIX)) and searches for the symbol myprog in the shared library.

When set to yes,

The search order is reversed. The runtime first looks for the shared library and if the symbol is not found, it then looks into the linked symbols.

Setting the COB_LOAD_PRIORITY environment variable provide the ability at runtime to produce the effect created by using the `module-load-priority=y` compiler configuration file flag at compilation time.

COB_LS_DOS=[Y/N]

Default is N.

When set to Y/YES, the record delimiter for LINE SEQUENTIAL files is set to CR/LF. The COB_LS_DOS runtime environment variable is evaluated when the OPEN statement is executed by the runtime. Changes to the COB_LS_DOS runtime environment variable can be made during the runtime session.

COB_LS_FIXED=[Y/N]

Default is N.

When set to Y/YES, LINE SEQUENTIAL files are processed as RECORD SEQUENTIAL FILES. The COB_LS_FIXED runtime environment variable is evaluated when the OPEN statement is executed by the runtime. Changes to the COB_LS_FIXED runtime environment variable can be made during the runtime session.

COB_LS_NULLS=[Y/N]

Default is Y

When set to Y, causes LINE SEQUENTIAL files to escape all values less than 0x20 with a null (0) value. The COB_LS_NULLS runtime environment variable is evaluated when the OPEN statement is executed by the runtime. Changes to the COB_LS_NULLS runtime environment variable can be made during the runtime session.

COB_NO_DOT_DAT

The COB_NO_DOT_DAT runtime environment variable, when set to Y, instructs the VBISAM indexed file driver to not append .dat to the data file. The index file still is created with a .idx extension.

COB_NO_SIGNAL=[Y/N]

Default is N.

When set to Y, causes the runtime to not catch the signal which lets the system build a core dump. Setting COB_NO_SIGNAL can improve performance, while reducing the diagnostic capabilities of the runtime.

COB_PAD_BUG=[0/1]

Default is 0.

The runtime environment variable COB_PAD_BUG compensates for a bug that was detected in the handling of variable length RECORD SEQUENTIAL files (REC MODE "V") created in a COBOL-IT version prior to version 3.5.8 that was greater than 2GB in size, and that was created using the compiler configuration flag variable-rec-pad-mf: yes .

When the user had set variable-rec-pad-mf: yes, then when a record sequential file exceeded 2GB in size, all records whose last character address was larger than 2GB(0x80000000) were written with an additional 4-byte low-value filler. This anomaly did not occur when record addresses were smaller than 2GB (0x80000000).

Prior to version 3.5.8, this extra padding was not detected by the COBOL-IT runtime, and did not affect the handling of the data in the file. However, the extra 4-byte low-value filler did cause errors when using external tools, such as Syncsort.

With the correction of this problem, the COBOL-IT runtime will detect the extra padding condition, and these files will not be READable if the environment variable COB_PAD_BUG=1 is not set.

Note that while setting the environment variable `COB_PAD_BUG=1`, the old behavior of adding extra padding when writing records whose last character address is larger than 2GB is preserved, and the ability to read these records is preserved.

To rebuild a file that has been corrupted in this manner, set the environment variable `COB_PAD_BUG=1`, use the COBOL configuration file setting `variable-rec-pad-mf=yes`, and run a program which READS through the file with records containing extra padding, and WRITES the records back out to a flat file with non-variable length (REC MODE "F"). Then, set the environment variable `COB_PAD_BUG=0`, use the configuration file setting `variable-rec-pad-mf=yes`, and run a second program which READS the flat file, computes the length of the variable-length record, and WRITES the records back out to a flat file with variable length (REC MODE "V"). Note that two separate programs are required, as COBOL-IT reads the `COB_PAD_BUG` environment variable just once when the runtime starts.

The `COB_PAD_BUG` runtime environment variable is evaluated when the OPEN statement is executed by the runtime. Changes to the `COB_PAD_BUG` runtime environment variable can be made during the runtime session.

COB_PRE_LOAD=[list of modules]

Iterates a colon-separated ':' list of external shared libraries on (Unix, Linux) or a semi-colon-separated ';' list of external shared libraries on Windows. The list of shared libraries is preloaded at start-up.

COB_PROFILING_DIR

When compiling with the `-fprofiling` compiler flag, the runtime will check for the `COB_PROFILING_DIR` environment variable, and generate the profiling data file in that directory if it is defined. Otherwise, the profiling data file is created in the same directory as the source file.

Note that `COB_PROFILING_DIR` environment variable requires a trailing slash.

As an example, in Linux/UNIX environments:

```
>export COB_PROFILING_DIR=mydir/
```

As a result, the output file is generated as `mydir/cob_profiling_<pid>_final.xls`

In Windows environments:

```
>set COB_PROFILING_DIR=mydir\
```

As a result, the output file is generated as `mydir\cob_profiling_<pid>_final.xls`

COB_PROFILING_EACH_MODULE

The `COB_PROFILING_EACH_MODULE` runtime environment variable, when set to Y, causes the profiler to revert to the old profiling behavior, in which the .xls file is output at the exit of each module in an application.

COB_RTL_CP=<codepage>

Names the code page for Right-to-Left languages such as Hebrew.

Define the environment variable COB_RTL_CP=<codepage> where <codepage> is the code page of the Right To Left Language. Then when displaying a string, the part of the string that must be written Right to left will be inverted.

Note that in Windows environments, if you define the environment variable

COB_CONSOLE_CP=<codepage>

the codepage of the console is changed at program startup. (This is equivalent to DOS command chcp<codepage>). Currently for Hebrew (1255, 8859-8, 862) are supported.

COB_RUNTIME_CHECK_TRACE=[Y/N/Module list separated by ; or : (Windows)]

This environment variable should be used with caution.

The COB_RUNTIME_CHECK_runtime environment variable affects the behavior of the “subscript out of bounds” runtime check that is made when compiling with -g -debug, and optionally with -fmem-info.

When set to “Y”, the “subscript out of bounds” will not force the runtime to abort, but to generate a message referencing the line on which the condition was detected. Messages generated will be written to the COB_ERROR_FILE if it is defined, otherwise, they will be written to stdout. No memory dump is produced, even if the COB_DUMP environment variable is set and the program is compiled with -fmem-info.

Consider a case where a program has a table with 5 elements, but a PERFORM loop increments the subscript 10 times. When COB_RUNTIME_CHECK_TRACE is set to N (the default), the “subscript out of bounds” condition forces the runtime to abort when the first out of bounds condition is detected. If the program is compiled with -fmem-info and the COB_DUMP environment variable is set, a memory dump is written to the COB_DUMP file.

When COB_RUNTIME_CHECK_TRACE is set to Y, the runtime will continue to run, producing output as the subscript continues to increment, as seen below:

```
C:/COBOL/COBOLIT/Samples/test.cbl:20: libcob: Subscript of 'sample-element' out of bounds: 6
C:/COBOL/COBOLIT/Samples/test.cbl:20: libcob: Subscript of 'sample-element' out of bounds: 7
C:/COBOL/COBOLIT/Samples/test.cbl:20: libcob: Subscript of 'sample-element' out of bounds: 8
C:/COBOL/COBOLIT/Samples/test.cbl:20: libcob: Subscript of 'sample-element' out of bounds: 9
C:/COBOL/COBOLIT/Samples/test.cbl:20: libcob: Subscript of 'sample-element' out of bounds: 10
```

When set to Module list separated by “;” or “:” (Windows)

COB_RUNTIME_CHECK_TRACE is set to Y only with the listed modules.

Note that the module list must be surrounded by single-quotes in Linux/UNIX, and not surrounded

by single-quotes in Windows.

In Linux/UNIX, the module list must be surrounded by single-quotes:

Example:

```
>export COB_RUNTIME_CHECK_TRACE='module1;module2'
```

In Windows, the module list must not be surrounded by single-quotes, and both the “;” and “:” characters are accepted as module delimiters.

Example:

```
>SET COB_RUNTIME_CHECK_TRACE=module1;module2
```

or

```
>SET COB_RUNTIME_CHECK_TRACE=module1:module2
```

COB_SCREEN_DISABLE_REFORMAT=[Y/N]

Default is N.

When set to Y, disables the reformatting associated by default with the COB_SCREEN_UPDATE_FIRST_KEY_ERASE behavior.

COB_SCREEN_ESC=[Y/N]

Default is N.

When set to “Y”, enables use of the escape key. When COB_SCREEN_ESC=Y, and COB_SCREEN_EXCEPTIONS=Y, the COBOL-IT runtime will return the CRT Status value as described in the table below:

Condition	Key Pressed	Cit Value returned to CRT-STATUS-VAR
Where COB_SCREEN_ESC=Y	Esc	2005

COB_SCREEN_EXCEPTIONS=[Y/N]

Default is N.

When set to “Y”, enables the use of the Page Up, Page Down, Up Arrow, and Down Arrow keys on Field-level ACCEPT statements. Also enables the use of the escape key when COB_SCREEN_ESC=Y.

When COB_SCREEN_EXCEPTIONS=Y, the COBOL-IT runtime will return the CRT Status values as described in the table below. Note that Page Up, Page Down, Up Arrow, and Down Arrow are enabled by default when ACCEPTing a Screen:

Condition	Key Pressed	Cit Value returned to CRT-STATUS-VAR
-----------	-------------	--------------------------------------

On ACCEPT <Screen> or On ACCEPT <Field> where COB_SCREEN_EXCEPTIONS=Y	Page Up	2001
	Page Down	2002
	Up Arrow	2003
	Down Arrow	2004
When COB_SCREEN_ESC=Y	Escape Key	2005
	Print	2006

COB_SCREEN_INPUT_BOLDDED=[Y/N]

Default is N.

When set to Y, causes all input fields to be displayed in bold.

COB_SCREEN_INPUT_FILLER=[char]

When set to [char], changes the PROMPT character to [char].

COB_SCREEN_INPUT_INSERT_TOGGLE=[Y/N]

Default is N.

When set to Y, causes the INS Key to toggle between Overwrite and Insert mode. The default behavior of the INS key is to insert a SPACE at the current cursor position.

COB_SCREEN_INPUT_REVERSED=[Y/N]

Default is N.

When set to Y, causes all input fields to be displayed in REVERSE.

COB_SCREEN_INPUT_UNDERLINED=[Y/N]

Default is N.

When set to Y, causes all input fields to be displayed with UNDERLINE.

COB_SCREEN_RAW_KEYS=[Y/N]

Default is N.

When set to Y, enables use of the “raw keys”.

The “raw keys” are the Home, End, Insert, Delete, and Erase EOL keys. When COB_SCREEN_RAW_KEYS=Y, the COBOL-IT runtime will return CRT Status values as described in the table below:

Condition	Key Pressed	Cit Value returned to CRT-STATUS-VAR
-----------	-------------	--------------------------------------

Where COB_SCREEN_RAW_KEYS=Y	Home	2007
	End	2008
	Ins	2009
	Del	2010
	Erase EOL	\2011

COB_SCREEN_UPDATE_FIRST_KEY_ERASE=[Y/N]

Default is N.

When set to Y, causes all field-level ACCEPT WITH UPDATE statements to behave as described below:

If the first key pressed when entering the field is an alpha/number key the field is erased, and the keystroke recorded. Reformatting rules are applied when exiting the field if the number of characters entered is fewer than can be held by the field.

Consider an example:

A field-level ACCEPT WITH UPDATE is being done on a numeric data element described as PIC 9(4). The user enters a single digit “1”, and exits the field. The environment variable COB_SCREEN_UPDATE_FIRST_KEY_ERASE is set to “Y”.

Prior to recording the keystroke, the runtime initializes the field to zeroes. Then, the runtime records the keystroke of “1” in the first character position, then exits the field.

In many (perhaps most) cases, the user will expect the data item to reformat from 1000 to 0001. This reformatting is the default behavior when COB_SCREEN_UPDATE_FIRST_KEY_ERASE is set to “Y”.

For the cases where this reformatting is not desired, the user can disable this default behavior with the COB_SCREEN_DISABLE_REFORMAT behavior, as described below.

The runtime environment variable COB_SCREEN_DISABLE_REFORMAT, when set to Y, disables the reformatting associated by default with the COB_SCREEN_UPDATE_FIRST_KEY_ERASE behavior.

COB_STDUNIX=[Y/N]

Default is N.

Windows-only. When set to Y/YES, stdin, stdout, and stderr are opened in binary mode. This means that the EOL (end of line) character used is LF (line-feed) only, as opposed to CR/LF, which is the default in Windows operating environments.

COB_SWITCH_0... COB_SWITCH_16

Setting any of the COB_SWITCH_1 thru COB_SWITCH_16 environment variables provides a way for the user to set the corresponding SWITCH 0 thru SWITCH 16 declared in SPECIAL-NAMES to either an ON or OFF state.

For example :

```
SET COB_SWITCH_1=ON  
>cobcrun switchtest
```

provides a way for the program switchtest, in which SWITCH 1 is declared in SPECIAL-NAMES, to run with SWITCH 1 set to an ON STATUS.

COB_SYNC=[Y/N]

Default is N.

When set to Y, causes all WRITE operations to be followed by a flush to disk. Warning- this option may degrade the performance of your program. The COB_SYNC runtime environment variable is evaluated when the OPEN statement is executed by the runtime. Changes to the COB_SYNC runtime environment variable can be made during the runtime session.

COB_VAR_REC_PAD=[Y/N]

Default is N.

The COB_VAR_REC_PAD runtime environment variable affects padding rules applied to variable length sequential records. When set to yes, variable size RECORD SEQUENTIAL records (REC MODE 'V') are stored with padding characters at the end to ensure that the next record starts on a 4 byte boundary. The COB_VAR_REC_PAD runtime environment variable is evaluated when the OPEN statement is executed by the runtime. Changes to the COB_VAR_REC_PAD runtime environment variable can be made during the runtime session.

COBLPFORM="n:n:n : : : : : : :n"

Provides a way to emulate printer channels C01 through C12 by line feeds and form feeds.

With the implementation of the environment variable COBLPFORM, COBOL-IT can emulate the printer channels C01 through C12 by line feeds and form feeds. The environment variable COBLPFORM provides a syntax that allows the user to define line numbers on the form, and to associate printer channels with those line numbers.

Syntax: >COBLPFORM="n:n:n : : : : : : :n"; export COBLPFORM

Parameters:

File Status Codes

COBOL-IT produces the same file status codes for VBISAM, CISAM, DISAM, and BerkeleyDB indexed files. For information on how to map COBOL-IT file status codes to custom file status codes, see the `fstatus-map` compiler flag. The `-fstatus-map` compiler flag may be repeated as many times as necessary for cases where more than one file status code needs to be re-mapped.

00	Success The file operation was successfully executed.
02	Success- Duplicate The file allows duplicate keys and a duplicate key has been detected on a READ NEXT operation; or has been created on a WRITE or REWRITE operation. (READ, WRITE, REWRITE)
04	Success- Incomplete In a record sequential file with fixed record size, the READ of the last record returns fewer bytes than requested. (READ)
05	Success- Optional The SELECT statement for the file contains the OPTIONAL phrase and the file does not exist, but the operation being executed does not require the existence of the file. (OPEN, DELETE)
07	Success- No Unit A CLOSE UNIT WITH NO REWIND or CLOSE UNIT REMOVAL statement detected the absence of a UNIT, but does not require it. (CLOSE)
10	End of file. A READ NEXT or READ PREVIOUS statement has detected the end (or beginning) of the file. (READ)
14	Out of KeyRange A READ operation on a relative file failed when the key number was larger than permitted by the definition of the relative key data item. (READ)
21	Invalid Key A WRITE operation on a relative file failed when the key number was inconsistent with the definition of the relative key data item. A DELETE, or READ on key operation failed because the record did not exist.

	<p>A REWRITE operation failed because the key was changed. (WRITE, DELETE, READ, REWRITE)</p>
22	<p>If the environment variable COB_KEY_DUP_ALWAYS_22 is set to "Y":</p> <p>Key Exists / Duplicate Key The File does not allow duplicate keys and a duplicate key has been detected as the result of a WRITE or REWRITE operation. (WRITE, REWRITE)</p>
23	<p>Record not found. A READ on key, REWRITE, DELETE, or READ NEXT after START failed because the record was not found. (READ, REWRITE, DELETE)</p>
30	<p>Permanent Error An error at the level of the operating system has occurred.</p>
35	<p>File Not Found (OPEN, SORT)</p>
37	<p>Permission Denied The OPEN statement being executed is not permitted. The user may not have permissions on the system to create a file in a given directory, or the OPEN operation requested may be not permitted. A PRINT file for example, may not be OPENed INPUT. (OPEN)</p>
38	<p>Closed with Lock An OPEN statement failed on a file that was CLOSED with LOCK by the current run unit. (OPEN)</p>
39	<p>Conflict Attribute An OPEN statement failed when a difference between the attributes of the file being OPENed and the file description was detected. (OPEN)</p>
41	<p>File Already Open An OPEN statement failed because the file is already open. (OPEN)</p>
42	<p>File Already Closed A CLOSE statement failed because the file is already closed. (CLOSE)</p>

43	<p>Read not Done</p> <p>When ACCESS MODE is SEQUENTIAL, A REWRITE or DELETE statement failed because it was not preceded by a READ statement.</p> <p>(REWRITE, DELETE)</p>
44	<p>Record overflow</p> <p>A READ operation on a variable size record or a LINE SEQUENTIAL file failed because the read record is greater than the declared Record</p> <p>A WRITE operation fail because the value of the field of the clause SIZE DEPENDING ON is smaller than the minimum record size or greater than the maximum size</p> <p>A REWRITE operation fail when ACCESS MODE is SEQUENTIAL because the new record size is different from the original record size</p>
46	<p>Read Error</p> <p>A READ NEXT/PREVIOUS statement failed because there is no file pointer of reference.</p> <p>(READ)</p>
47	<p>Input Denied</p> <p>An input file operation failed either because the file was not open, or not open in the mode required.</p> <p>(READ, START)</p>
48	<p>Output Denied</p> <p>An output file operation failed either because the file was not open, or not open in the mode required.</p> <p>(WRITE)</p>
49	<p>I_O Denied</p> <p>An IO file operation failed either because the file was not open, or not open in the mode required. (REWRITE, DELETE)</p>
51	<p>Record Locked</p> <p>A READ operation failed because the record is locked by another process.</p> <p>(READ)</p>
52	<p>EOP</p> <p>A WRITE operation on a PRINT file failed after End-of-Page.</p> <p>(WRITE)</p>
57	<p>I_O Linage</p> <p>An OPEN operation failed because the LINAGE description was incorrect.</p> <p>(OPEN)</p>
61	<p>File Sharing</p> <p>An OPEN operation failed because the file was locked</p>

	by another process. (OPEN)
91	Not Available The operation requested is not available for the file. Returned when operations that are specific to indexed files are executed on record/line sequential files. (START, READ on key, REWRITE, DELETE)

Runtime Error Codes

128	CALL Unresolved
129	NULL name parameter passed to 'cobcancel'"
130	'cobcall' - Runtime has not been initialized"
131	Invalid number of arguments to 'cobcall'"
132	NULL name parameter passed to 'cobcall'"
133	'cobfunc' - Runtime has not been initialized"
134	NULL name parameter passed to 'cobsavenv'"
135	NULL name parameter passed to 'coblongjmp'"
136	Cannot acquire %d bytes of memory - Aborting"
137	BASED/LINKAGE item '%s' has NULL address"
138	'%s' not numeric: '%s'"
139	OCCURS DEPENDING ON '%s' out of bounds: %d"
140	Subscript of '%s' out of bounds: %d"
141	Offset of '%s' out of bounds: %d"
142	Length of '%s' out of bounds: %d"
143	EXTERNAL item '%s' has size > %d"
144	'cobcommandline' - Runtime has not been initialized"
145	Parameter to SYSTEM call is larger than 8192 characters"
146	Invalid context file %s for reading"
147	Can't open context file %s for reading"
148	Can't write context file %s key "
149	Can't open context file %s for writing"
150	Unexpected context mode %d"
151	Context file not open for read/write"
152	Can't read context file %s data"

153	Runtime is unable to acquire temporary file"
154	extfh_indexed_open : invalid open mode %d for %s
155	extfh_indexed_start : invalid condition
156	extfh_read : invalid read option %d"
157	extsm/SORT : maximum USING/GIVING clauses exceeded"
158	extfh/extsm Internal error
159	Failed to initialize curses"
160	cob_init() has not been called"
161	Codegen error - Please report this to support@COBOL-it.com"
162	ERROR - Recursive call of chained program"
163	Stack overflow, possible PERFORM depth exceeded"
164	CBL_xxx CALL Insufficient parameters count
253	Function only available in enterprise version
254	see messages
255	Undefined error

Data Memory allocation

Find Here after the memory allocation for every USAGE clause

“Format Native” means that the data may be stored in either little or big-endian representation, depending on the platform on which the program is running.

BINARY, COMPUTATIONAL

Size	<p>Depends on number of “9”s in PICTURE and the “binary-size” setting of the configuration file used to compile the program. See “binary-size” description for more detail.</p> <p>Depending of the “bin-opt:[yes/no]”</p> <p>Default is bin-opt:no.</p> <p>When set to yes,</p> <p>Enables binary operation optimization. The –bin-opt optimizations are enabled by use of the –O compiler flag.</p>
Format	<p><i>bin-opt-strict:[yes/no]</i></p> <p>When set to yes,</p> <p>Causes -fbin-opt binary operation optimization to be strictly respected.</p> <p>“binary-byteorder” setting of the configuration file used to compile the program</p>
Negative value allowed	If PICTURE contains “S”
PICTURE allowed	Yes

BINARY-CHAR, BINARY-CHAR SIGNED

Size	1 Byte
Format	Native
Negative value allowed	Yes
PICTURE allowed	No

BINARY-CHAR UNSIGNED

Size	1 Byte
Format	Native
Negative value allowed	No
PICTURE allowed	No

BINARY-C-LONG,

BINARY-C-LONG SIGNED

Size	Allocates the same amount of storage as does the C language "long" data type on that computer; typically this is 32 bits but it could be 64 bits
Format	Native
Negative value allowed	Yes
PICTURE allowed	No

BINARY-C-LONG UNSIGNED

Size	Allocates the same amount of storage as does the C language "long" data type on that computer; typically this is 32 bits but it could be 64 bits
Format	Native
Negative value allowed	No
PICTURE allowed	No

**BINARY-DOUBLE,
BINARY-DOUBLE SIGNED**

Size	Allocates a "traditional" double-word of storage (64 bits)
Format	Native
Negative value allowed	Yes
PICTURE allowed	No

BINARY-DOUBLE UNSIGNED

Size	Allocates a "traditional" double-word of storage (64 bits)
Format	Native
Negative value allowed	No
PICTURE allowed	No

**BINARY-LONG,
BINARY-LONG SIGNED,
SIGNED-LONG,
SIGNED-INT**

Size	32 Bits
Format	Native
Negative value allowed	Yes
PICTURE allowed	No

BINARY-SHORT

BINARY-SHORT SIGNED
SIGNED-SHORT

Size	16 Bits
Format	Native
Negative value allowed	Yes
PICTURE allowed	No

**BINARY-SHORT UNSIGNED
UNSIGNED-SHORT**

Size	16 Bits
Format	Native
Negative value allowed	No
PICTURE allowed	No

COMPUTATIONAL-1

Size	Allocates a word of storage (32 bits)
Format	Single-precision floating-point
Negative value allowed	Yes
PICTURE allowed	No

COMPUTATIONAL-2

Size	Allocates a double-word of storage (64 bits)
Format	Double-precision floating-point
Negative value allowed	Yes
PICTURE allowed	No

**COMPUTATIONAL-3
PACKED-DECIMAL**

Size	Allocates 4 bits per "9" in the PICTURE plus a (trailing) 4-bits field for the sign, rounded up to the nearest byte
Format	Packed decimal
Negative value allowed	If PICTURE contains "S"
PICTURE allowed	Yes

COMPUTATIONAL-4

Size	<p>If "pack-comp-4" setting of the configuration is set to yes, the size is computed like a COMPUTATIONAL as if the "binary-size" setting of the configuration is set to "1-8".</p> <p>If "pack-comp-4" setting of the configuration is set to No, the size is identical to COMPUTATIONAL</p> <p>See "binary-size" for more detail.</p> <p>Depends on setting of bin-opt:[yes/no]</p> <p>Default is bin-opt:no.</p>
Format	<p>When set to yes,</p> <p>Enables binary operation optimization. The -bin-opt optimizations are enabled by use of the -O compiler flag.</p>

bin-opt-strict:[yes/no]

When set to yes,

Causes -fbin-opt binary operation optimization to be strictly respected.

"binary-byteorder" setting of the configuration file used to compile the program.

 Negative value
allowed
PICTURE
allowed

If PICTURE contains "S"

Yes

COMPUTATIONAL-5

Size

Depends on number of "9"s in PICTURE and the "binary-size" setting of the configuration file used to compile the program. See "binary-size" description for more detail

 Format
Negative value
allowed
PICTURE
allowed

Native

If PICTURE contains "S"

Yes

COMPUTATIONAL-6

Size

Allocates 4 bits per "9" in the PICTURE, as is done with COMPUTATIONAL-3 data items. However, the data may not be SIGNED, so there is no trailing 4-bit field for the sign. When the number of digits is odd, the high-order 4-bits contains a 0. To calculate the size of a comp-6 data item, divide the PICTURE size by 2 and round up.

 Format
Negative value
allowed
PICTURE
allowed

Packed Decimal

No

Yes

COMPUTATIONAL-X

Size

The size is computed like Signed COMPUTATIONAL as if the "binary-size" setting of the configuration is set to "1-8" and Signed rule are used event if the Picture do not include "S"

Depending of the "bin-opt:[yes/no]"

Default is bin-opt:no.

Format

When set to yes,

Enables binary operation optimization. The -bin-opt optimizations are enabled by use of the -O compiler flag.

bin-opt-strict:[yes/no]

When set to yes,

Causes -fbin-opt binary operation optimization to be strictly respected.

“binary-byteorder” setting of the configuration file used to compile the program

Negative value allowed	If PICTURE contains “S”
PICTURE allowed	Yes

DISPLAY

Size	Depends on PICTURE - One character per X, A, 9, period, \$, Z, 0, *, S (if SEPARATE CHARACTER specified), +, - or B symbol in PICTURE; Add 2 more bytes if DB or CR symbol used
Format	Byte
Negative value allowed	If PICTURE contains “S”
PICTURE allowed	Yes

INDEX

Size	32 Bits
Format	Native
Negative value allowed	No
PICTURE allowed	No

POINTER**PROGRAM-POINTER**

Size	32 Bits or 64 Bits depending memory model used
Format	Native
Negative value allowed	No
PICTURE allowed	No

Using EXTFH-Compliant Indexed File Systems

COBOL-IT includes EXTFH Libraries

The COBOL-IT distribution includes EXTFH drivers for the BerkeleyDB, D-ISAM and VBISAM file systems. The EXTFH drivers and libraries for the C-Tree ISAM ISAM File engine can be acquired through COBOL-IT, and C-Tree is also supported. Enabling the use of any of these file systems can be done either with the use of a compiler flag, or with a setting of the COB_EXTFH runtime environment variable.

File System	Compiler Flag	Compiler Configuration File	COB_EXTFH environment var
BerkeleyDB	-fbdb	bdb:yes	>export COB_EXTFH=bdbextfh (UNIX/Linux) >set COB_EXTFH=bdbextfh (Windows)
D-ISAM	-fdisam	disam:yes	>export COB_EXTFH=disamextfh (UNIX/Linux) >set COB_EXTFH=disamextfh (Windows)
C-Tree	-fctree	ctree:yes	>export COB_EXTFH=ctextfh (UNIX/Linux) >set COB_EXTFH=ctextfh (Windows)
VBISAM	-fvbisam	vbisam:yes	>export COB_EXTFH=vbisamextfh (UNIX/Linux) >set COB_EXTFH=vbisamextfh (Windows)

When the COB_EXTFH environment variable is defined (and no additional variables are included to specify what library to load), the runtime looks for lib\$(COB_EXTFH) on UNIX and \$(COB_EXTFH)_dll.dll on Windows in the COBOL-IT installation directory and all directories indicated in the COB_LIBRARY_PATH.

As an example, to use D-ISAM through EXTFH (UNIX/Linux):

```
>export COB_EXTFH=disamextfh
```

Libdisamextfh.so will be found and loaded into the CIT distribution directory (COBOLITDIR).

These file systems can be used with all COBOL-IT tools including CitSORT.

VBISAM

The VBISAM file system is the default indexed file system used by COBOL-IT.

-fvbisam

The -fvbisam compiler flag forces use of the VBISAM Extfh indexed file engine.

This is the default setting in version 3.x and prior versions of COBOL-IT. However, in the future release of COBOL-IT version 4.x, VBISAM will be deprecated and D-ISAM will become the default. At that time, continued use of VBISAM files will require that the VBISAM Extfh indexed file engine be activated either by using the -vbisam compiler flag, or with the use of the

COB_EXTFH=vbisamextfh runtime environment variable setting.

vbisam: [yes/no]

Default is `vbisam: yes`.

When set to yes, forces use of the BerkeleyDB Extfh indexed file engine.

BerkeleyDB

BerkeleyDB is licensed by Oracle. For information on how to install, and license your BerkeleyDB file system, visit www.oracle.com.

-fbdb

The `-fbdb` compiler flag forces use of the BerkeleyDB Extfh indexed file engine.

bdb: [yes/no]

Default is `bdb: no`.

When set to yes, forces use of the BerkeleyDB Extfh indexed file engine.

D-ISAM

The D-ISAM engine is more widely used than the VBISAM engine, and is fully compatible with IBM C-ISAM 7.2. Unfortunately, VBISAM files are not readable by D-ISAM and will require conversion. The VBISAM engine will be deprecated in the next major release (4.x) and replaced by the D-ISAM indexed file engine.

-fdisam

The `-fdisam` compiler flag forces use of the D-ISAM Extfh indexed file engine. In the next major release (4.x), the `-fdisam` compiler flag will be set by default.

disam: [yes/no]

Default is `disam: no`.

The `disam` compiler configuration flag when set to yes, forces use of the D-ISAM Extfh indexed file engine. In the next major release (4.x), the default will be set to yes.

dcheck

The D-ISAM indexed file engine includes a check utility: `dcheck`. `Dcheck` is available on all Linux, UNIX and Windows systems.

Usage: `dcheck [-hifbB] isamfile [...]`

h display isam header information only

i just check indexes, ignore data file

f fix corrupt indexes
b rebuild all indexes
B rebuild specific index

NOTES

The option string, preceded by a dash, can be placed anywhere on the command line. All options must be specified in one string, and all options apply to all files specified.

The *-B* option is a little different. It can be specified multiple times, and each occurrence must be followed by an index number. A value of 1 denotes the primary index, 2 the next, etc. You cannot specify *-B* on multiple files.

dcheck is not interactive, and will ask no questions, so it can be safely used in batch and script files without operator intervention.

dcheck has been designed to run co-operatively (unless *-f* or *-b* specified) on files in active use, but note that other processes will be blocked for the duration of some of the check cycles.

Example:

In the example below, the *dcheck* utility is used with no options, and returns information about the *holidaysIX* data file. Here, we can see that the record length is 70 character, and there are two indexes, a primary index that begins at offset zero, and is 25 bytes long, and an alternate index that allows duplicate keys, begins at offset 25 and is 24 bytes long.

```
C:\COBOL\COBOLIT\Samples>dcheck holidaysIX
```

```
holidaysIX structure  
data record length: 70  
index block size: 1024  
index dup width: 4  
index 1: uniq char@0/25  
index 2: dups char@25/24  
data file: 19 slots allocated, 0 free  
index file: 4 slots allocated, 0 free
```

```
checking data..ok  
checking index 1..ok
```

C-Tree ACE

Documentation

Documentation for the complete c-Tree ACE engine is available at
http://www.faircom.com/ace/support_doc_t.php

Installation

Download the c-Tree ACE engine from the www.COBOL-it-online.com site.
You will find 2 different database engines:

- ISAM : Provides all ISAM data services.
- SQL: Provides ISAM services and the ability to query the stored table with SQL.

Ensure you download the version corresponding to the license key you have received.

By default the c-Tree engine will store data files in a sub directory of the c-Tree root installation directory. It is helpful to take that into consideration when selecting the install directory for the C-Tree distribution. Ensure that you will have enough space for your future COBOL data files.

Windows:

Just double click on the .msi file and follow the instructions.

Unix/Linux :

Move the Package to the selected installation directory (example /opt)
Decompress the package:

```
gunzipctreeACE-xxxxx.tar.gz
```

Un-tar the package:

```
tar xftreeACE-xxxxx.tar
```

This will create a root installation directory for c-Tree ACE. (Example /opt/linux.x64.64bit)

Set COB_CTREE_PATH

The COBOL-IT Runtime requires the environment variable **COB_CTREE_PATH** to be set on the path where the c-Tree runtime library is located. In Windows, the c-Tree runtime library is called mtclient.dll, and in Unix the c-Tree runtime library is called libmtclient.so.

Windows

<Root c-Tree path> is C:\FairCom\V9.3.0 in Windows installations.

Mtclient.dll is located in:

- <Root c-Tree path>\win32\bin\ace\sql\ for the SQL database engine installation
- <Root c-Tree path>\win32\bin\ace\isam for the ISAM database engine installation.

Typical settings will be:

```
SET COB_CTREE_PATH=C:\FairCom\V9.3.0\win32\bin\ace\sql
```

```
SET COB_CTREE_PATH=C:\FairCom\V9.3.0\win32\bin\ace\isam
```

Unix/Linux

<Root c-Tree path> varies in Linux/Unix installations, according to the platform.

By default, libmtclient.so is located in:

- <Root c-Tree path>/bin/ace/sql for the SQL database engine installation
- <Root c-Tree path>/bin/ace/isam for the ISAM database engine installation.
-

An example for a 64-bit Linux installation would be:

```
export COB_CTREE_PATH=/opt/linux.x64.64bit/bin/ace/sql
export COB_CTREE_PATH=/opt/linux.x64.64bit/bin/ace/isam
```

Compiling

The c-Tree database engine is not required to compile your programs for use with c-Tree ACE.

-fctree

Use the compilation flag `-fctree` to activate the usage of c-tree or add “ctree:yes” to your compiler configuration file.

Running

A program compiled to use c-Tree (with the `-fctree` compiler flag) requires the environment variable `COB_CTREE_PATH` to be set.

Start/Stop Engine

Windows

On Windows, the SQL/ISAM engine is installed as a service. Please refer to the c-Tree ACE documentation for details.

Unix/Linux

In the c-Tree root directory you will find 2 scripts :

- `startace` : Start the engine
- `stopace` : Stop the engine
-

Data file location

By default, the c-Tree data directory is located in the data subdirectory of the database engine. Specifically:

- <Root c-tree directory>/bin./ace/sql/data for the SQL database engine

- <Root c-tree directory>/bin./ace/isam/data for the ISAM database engine

Files whose location is described with a fully qualified path name, for example, /usr/data/ on a Unix system, or C:\data on a Windows system, are stored in that location.

Files whose location is described with a relative path are stored in a location relative to the c-Tree data directory.

Note that for both fully qualified paths, and for relative paths, the directories named will not be created by c-Tree if they do not exist. Make sure that all directories named in c-Tree file paths exist.

File name transformation follows the same rule as with the normal COBOL-IT file handles. DD_[filename] and COB_FILE_PATH are used to transform file names before calling c-Tree.

Reserved Word List

ABEND	ARGUMENT-VALUE	BINARY-SHORT
ACCEPT	AS	BIT
ACCESS	ASCENDING	BLANK
ADD	ASSIGN	BLINK
ADDRESS	AT	BLOCK
ADVANCING	AUTO	BOTTOM
AFTER	AUTO-SKIP	BY
ALL	AUTOMATIC	BYTE-LENGTH
ALLOCATE	AUTOTERMINATE	CALL
ALPHABET	B-AND	CANCEL
ALPHABETIC	B-EXOR	CDECL
ALPHABETIC-LOWER	B-NOT	CENTURY-DATE
ALPHABETIC-UPPER	B-OR	CENTURY-DAY
ALPHANUMERIC	B-XOR	CHAIN
ALPHANUMERIC-EDITED	BACKGROUND-COLOR	CHAINING
ALSO	BACKGROUND-COLOUR	CHANGED
ALTER	BASED	CHARACTER
ALTERNATE	BEEP	CHARACTERS
AND	BEFORE	CHECKPOINT
ANY	BELL	CLASS
APPLY	BINARY	CLOSE
ARE	BINARY-C-LONG	CODE
AREA	BINARY-CHAR	CODE-SET
AREAS	BINARY-DOUBLE	COL
ARGUMENT-NUMBER	BINARY-LONG	COLLATING

COLS	CONTENT	DELIMITER
COLUMN	CONTINUE	DEPENDING
COLUMNS	CONTROL	DESCENDING
COMMA	CONTROLS	DESCRIPTOR
COMMAND-LINE	CONVERT	DETAIL
COMMIT	CONVERTING	DIR-SEPARATOR
COMMON	COPY	DISK
COMP	CORE-INDEX	DISPLAY
COMP-1	CORR	DISPLAY-1
COMP-2	CORRESPONDING	DIVIDE
COMP-3	COUNT	DIVISION
COMP-4	CRT	DOWN
COMP-5	CURRENCY	DUPLICATES
COMP-6	CURSOR	DYNAMIC
COMP-X	CYCLE	EBCDIC
COMPUTATIONAL	DATA	ECHO
COMPUTATIONAL-1	DATE	ELSE
COMPUTATIONAL-2	DAY	EMPTY-CHECK
COMPUTATIONAL-3	DAY-OF-WEEK	ENCODING
COMPUTATIONAL-4	DE	END
COMPUTATIONAL-5	DEBUGGING	END-ACCEPT
COMPUTATIONAL-X	DECIMAL-POINT	END-ADD
COMPUTE	DECLARATIVES	END-CALL
CONFIGURATION	DEFAULT	END-COMPUTE
CONSTANT	DELETE	END-DELETE
CONTAINS	DELIMITED	END-DISPLAY

END-DIVIDE	ERASE	FREE
END-EVALUATE	ERROR	FROM
END-EXHIBIT	ESCAPE	FULL
END-IF	EVALUATE	FUNCTION
END-MULTIPLY	EXCEPTION	FUNCTION-ID
END-OF-PAGE	EXCLUSIVE	GENERATE
END-PERFORM	EXHIBIT	GIVING
END-READ	EXIT	GLOBAL
END-RETURN	EXTEND	GO
END-REWRITE	EXTERNAL	GOBACK
END-SEARCH	FAILURE	GREATER
END-START	FALSE	GROUP
END-STRING	FD	GROUP-USAGE
END-SUBTRACT	FILE	HEADING
END-UNSTRING	FILE-CONTROL	HIGH
END-WRITE	FILE-ID	HIGH-VALUE
END-XML	FILLER	HIGH-VALUES
ENTRY	FINAL	HIGHLIGHT
ENVIRONMENT	FIRST	I-O
ENVIRONMENT-NAME	FLOAT-LONG	I-O-CONTROL
ENVIRONMENT-VALUE	FLOAT-SHORT	ID
EOL	FOOTING	IDENTIFICATION
EOP	FOR	IF
EOS	FOREGROUND-COLOR	IGNORE
EQUAL	FOREGROUND-COLOUR	IGNORING
EQUALS	FOREVER	IN

INDEX	LENGTH-CHECK	NATIONAL
INDEXED	LESS	NATIONAL-EDITED
INDICATE	LIKE	NATIVE
INITIAL	LIMIT	NEGATIVE
INITIALISE	LIMITS	NEXT
INITIALISED	LINAGE	NO
INITIALIZE	LINAGE-COUNTER	NO-ECHO
INITIALIZED	LINE	NOT
INITIATE	LINES	NULL
INPUT	LINKAGE	NULLS
INPUT-OUTPUT	LOCAL-STORAGE	NUMBER
INSPECT	LOCALE	NUMBERS
INTO	LOCK	NUMERIC
INTRINSIC	LOW	NUMERIC-EDITED
INVALID	LOW-VALUE	OBJECT-COMPUTER
IS	LOW-VALUES	OCCURS
JUST	LOWLIGHT	OF
JUSTIFIED	MANUAL	OFF
KEPT	MEMORY	OMITTED
KEY	MERGE	ON
LABEL	MINUS	ONLY
LAST	MODE	OPEN
LEADING	MOVE	OPTIONAL
LEFT	MULTIPLE	OR
LENGTH	MULTIPLY	ORDER
LENGTH-AN	NAMED	ORGANISATION

ORGANIZATION	PROGRAM	REPORTS
OTHER	PROGRAM-ID	REPOSITORY
OUTPUT	PROGRAM-POINTER	REQUIRED
OVERFLOW	PROMPT	RESERVE
OVERLINE	QUOTE	RETURN
PACKED-DECIMAL	QUOTES	RETURNING
PADDING	RANDOM	REVERSE
PAGE	RD	REVERSED
PARAGRAPH	READ	REVERSE-VIDEO
PARSE	RECORD	REWIND
PERFORM	RECORD-OVERFLOW	REWRITE
PIC	RECORDING	RIGHT
PICTURE	RECORDS	ROLLBACK
PLUS	RECURSIVE	ROUNDED
POINTER	REDEFINES	RUN
POSITION	REEL	SAME
POSITIVE	REFERENCE	SCREEN
PRESENT	RELATIVE	SCROLL
PREVIOUS	RELEASE	SD
PRINTER	REMAINDER	SEARCH
PRINTING	REMOVAL	SECTION
PROCEDURE	RENAMES	SECURE
PROCEDURE-POINTER	REORG-CRITERIA	SEGMENT-LIMIT
PROCEDURES	REPLACING	SELECT
PROCEED	REPORT	SENTENCE
PROCESSING	REPORTING	SEPARATE

SEQUENCE	SUBTRACT	UNDERLINE
SEQUENTIAL	SUCCESS	UNIT
SET	SUM	UNLOCK
SHARING	SUPPRESS	UNSIGNED
SIGN	SYMBOLIC	UNSIGNED-INT
SIGNED	SYNC	UNSIGNED-LONG
SIGNED-INT	SYNCHRONIZED	UNSIGNED-SHORT
SIGNED-LONG	TAB	UNSTRING
SIGNED-SHORT	TALLYING	UNTIL
SIZE	TAPE	UP
SORT	TERMINATE	UPDATE
SORT-MERGE	TEST	UPON
SOURCE	THAN	USAGE
SOURCE-COMPUTER	THEN	USE
SPACE	THROUGH	USING
SPACES	THRU	VALUE
SPECIAL-NAMES	TIME	VALUES
STANDARD	TIMEOUT	VARYING
STANDARD-1	TIMES	WAIT
STANDARD-2	TO	WHEN
START	TOP	WITH
STATIC	TRAILING	WORDS
STATUS	TRANSFORM	WORKING-STORAGE
STDCALL	TRUE	WRITE
STOP	TYPE	WRITE-ONLY
STRING	TYPEDDEF	XML

YYYYDDD	ZERO	ZEROS
YYYYMMDD	ZEROES	

Intrinsic Function List

Function name	Arguments	Type of fn	Returning
ABS	arg-1	Numeric	Absolute value of arg-1
ACOS	arg-1	Numeric	Arccosine of arg-1
ANNUITY	arg-1, arg-2	Numeric	Ration of annuity paid for arg-2 periods at interest of arg-1 to initial investment of one
ASIN	arg-1	Numeric	Arcsine of arg-1
ATAN	arg-1	Numeric	Arctangent of arg-1
BOOLEAN-OF-INTEGERS	Not supported	NA	NA
BYTE-LENGTH	arg-1	Numeric	Size in bytes of arg-1
BYTE-OF	arg-1	Numeric	Size in bytes of memory
CHAR	arg-1	Numeric	Character in position of arg-1 of the alphanumeric collating sequence
CHAR-NATIONAL	Not supported	NA	NA
COMBINED-DATETIME	arg-1, arg-2	Numeric	accepts two arguments - a date in integer date form, and a time in standard numeric time form - and returns a numeric value in which the date occupies the integer part of the value and the time represents the fractional part, according to the expression $\text{argument-1} + (\text{argument-2} / 100000)$. For example, given the integer date form value 143951 (representing the date February 15, 1995) and the standard numeric time form value 18867.812479168304 (representing the time 05:14:27.812479168304), the returned value would be exactly 143951.18867812479168304
CONCATENATE	arg-1 ...	Alphanumeric	Concatenation of alphanumeric representation of arg-x
COS	arg-1	Numeric	Cosine of arg-1

CURRENT-DATE	None	Alphanumeric	Current date and time and difference from Greenwich Mean Time
DATE-OF-INTEGER	arg-1	Numeric	Standard date equivalent (YYYYMMDD) of integer date
DATE-TO-YYYYMMDD	arg-1 ...	Numeric	arg-1 is converted from YYMMDD to YYYYMMDD based on the value of arg-2
DAY-OF-INTEGER	arg-1	Numeric	Julian date equivalent (YYYYDDD) of integer date.
DAY-TO-YYYYDDD	arg-1 ...	Numeric	arg-1 converted from YYDDD to YYYYDDD based on the value of arg-2
DISPLAY-OF	arg-1, arg-2 (optional)	Alphanumeric	An alphanumeric character string consisting of the content of argument-1 converted to a specific code page representation defined by arg-2 arg-1 must be class NATIONAL If arg-2 is omitted, the code page specified at compilation time is used
E	None	Numeric	Returns the value of e, the natural base
EXCEPTION-FILE	None	Alphanumeric	pos. 1-2: I-O-status pos. 3-134: file-name only for I-O conditions
EXCEPTION-FILE-N	Not supported	NA	NA
EXCEPTION-LOCATION	None	Alphanumeric	Three parts, separated by '; ' *1. program, function, or method name *2. paragraph name, including qualification if applicable *3. an implementor-defined identifier of the source line containing causing the exception
EXCEPTION-LOCATION-N	Not supported	NA	NA
EXCEPTION-STATEMENT	None	Alphanumeric	name of the verb causing the exception condition
EXCEPTION-STATUS	None	Alphanumeric	name of the exception condition
EXP	arg-1	Numeric	E raised to the power of arg-1
EXP10	arg-1	Numeric	10 raised to the power of arg-&

FACTORIAL	arg-1	Numeric	Factorial of arg-1
FRACTION-PART	arg-1	Numeric	Fraction part of arg-1
HIGHEST-ALGEBRAIC	Not supported	NA	NA
INTEGER	arg-1	Numeric	The greatest integer not greater than arg-1
INTEGER-OF-BOOLEAN	Not supported	NA	NA
INTEGER-OF-DATE	arg-1	Numeric	The integer equivalent of the standard date (YYYYMMDD)
INTEGER-OF-DAY	arg-1	Numeric	The integer date equivalent of the Julian Date (YYYYDDD)
INTEGER-PART	arg-1	Numeric	Integer part of arg-1
LENGTH	arg-1	Numeric	Length of arg-1 in character positions
LENGTH-AN	arg-1	Numeric	Length of arg-1 in alphanumeric character positions. It may be applied to an alphanumeric, national, or numeric data item, or literal.
LOCALE-COMPARE	Not supported	NA	NA
LOCALE-DATE	arg-1, arg-2	Alphanumeric	A character string containing a date in a culturally-appropriate format specified by a locale. arg-1 must be alphanumeric 8 character positions in length. arg-1 must be a date in the same format as the year, month, and day returned in character positions 1 through 8 by the CURRENT-DATE function. arg-2 must be associated with a locale in the SPECIAL-NAMES paragraph.
LOCALE-TIME	arg-1, arg-2	Alphanumeric	A character string containing a time in a culturally-appropriate format specified by a locale. arg-1 must be alphanumeric 13 character positions in length. arg-1 must be in the same format as the hours, minutes, and seconds

			returned in character positions 9 through 21 by the CURRENT-DATE function. arg-2 must be associated with a locale in the SPECIAL-NAMES paragraph.
LOCALE-TIME-FROM-SECONDS	arg-1, arg-2	Alphanumeric	Same as LOCAL-TIME arg-1 is numeric and represent number of second from Midnight
LOG	arg-1	Numeric	Natural log of arg-1
LOG10	arg-1	Numeric	Log to base 10 of arg-1
LOWER-CASE	arg-1	Alphanumeric	All letters in arg-1 are set to lower-case
LOWEST-ALGEBRAIC	Not supported	NA	NA
MAX	arg-1 ...	Numeric	Value of maximum argument
MEAN	arg-1 ...	Numeric	Arithmetic mean of arguments
MEDIAN	arg-1 ...	Numeric	Arithmetic median of arguments
MIDRANGE	arg-1 ...	Numeric	Mean of minimum and maximum arguments
MIN	arg-1 ...	Numeric	Value of minimum argument
MOD	arg-1, arg-2	Numeric	arg-1 modulo arg-2
NATIONAL-OF	arg-1, arg-2	National	National character string
NUMVAL	arg-1	Numeric	Numeric value of simple numeric string
NUMVAL-C	arg-1, arg-2	Numeric	Numeric value of numeric string with optional commas and currency string
NUMVAL-F	Not supported	NA	NA
ORD	arg-1	Numeric	Ordinal position of the argument in the collating sequence
ORD-MAX	arg-1 ...	Numeric	Ordinal position of maximum argument
ORD-MIN	arg-1 ...	Numeric	Ordinal position of minimum argument
PI	None	Numeric	Value of Pi
PRESENT-VALUE	arg-1 ...	Numeric	Preset value of a series of future period-end amounts, arg-2, at a discount rate of arg-1
RANDOM	Optional arg-1	Numeric	Random number. If arg-1 is provided, it will be used to reinitialize the 'pseudo'

				<p>random sequence. If no argument is provided, the next random number is returned. The returned number is between 0 and 1 with 10 significant decimal. It should be assigned to a PIC 9V9(10).</p>
RANGE	arg-1 ...	Numeric		Value of maximum argument minus value of minimum argument
REM	arg-1, arg-2	Numeric		Remainder of arg-1 /arg-2
REVERSE	arg-1	Alphanumeric		Reverse order of the characters of the argument
SECONDS-FROM-FORMATTED-TIME	arg-1, arg-2	Numeric		accepts two parameters - a literal that is either a time format or a combined date and time format, and a data item whose content is in the specified format - and returns a value in standard numeric time form.
SECONDS-PAST-MIDNIGHT	None	Numeric		number of second from Midnight
SIGN	arg-1	Numeric		The sign of arg-1
SIN	arg-1	Numeric		Sine of arg-1
SQRT	arg-1	Numeric		Square root of arg-1
STANDARD-COMPARE	Not Supported	NA		NA
STANDARD-DEVIATION	arg-1...	Numeric		Standard deviation of arguments
STORED-CHAR-LENGTH	arg-1	Numeric		The length of a string arg-1 ignoring trailing spaces
SUBSTITUTE	arg-1 ...	Alphanumeric		Copy of arg-1 where partial string arg-2 is replaced by value of arg-3, arg-4 by value of arg-5, ...
SUBSTITUTE-CASE	arg-1 ...	Alphanumeric		Idem SUBSTITUTE but comparison are done case independent
SUM	arg-1 ...	Numeric		Sum of arguments
TAN	arg-1	Numeric		Tangent of arg-1
TEST-DATE-YYYYMMDD	arg-1	Numeric		It returns a zero if the argument is a valid date; the value 1 if the year subfield content is out of range; the value 2 if the

				month subfield content is out of range; or the value 3 if the day subfield content is out of range for the given year and month. arg-1 must be standard date form (YYYYMMDD)
TEST-DAY-YYYYDDD	arg-1	Numeric		It returns a zero if the argument is a valid date; the value 1 if the year subfield content is out of range; or the value 2 if the day subfield content is out of range for the given year. Arg-1 must be in Julian date form (YYYYDDD)
TEST-NUMVAL	Not supported	NA		NA
TEST-NUMVAL-C	Not supported	NA		NA
TEST-NUMVAL-F	Not supported	NA		NA
TRIM	arg-1, arg-2	Alphanumeric		The given string arg-1 with any leading and trailing blanks removed, or if arg-2 if given: 0 trim spaces both before and after 1 trim spaces before only 2 trim spaces after only
TRIML	arg-1	Alphanumeric		Equivalent to: TRIM(arg-1, 1)
TRIMR	arg-1	Alphanumeric		Equivalent to: TRIM(arg-1, 2)
UPPER-CASE	arg-1	Alphanumeric		All letters in the argument are set to upper case
VARIANCE	arg-1...	Numeric		Variance of argument
WHEN-COMPILED	None	Alphanumeric		Date and time program was compiled
YEAR-TO-YYYY	arg-1, arg-2	Numeric		arg-1 converted from YY to YYYY based on the value of arg-2

COBOL-IT® Library Routines

COBOL-IT supports a number of library routines, which are built into the compiled objects, and can be CALL'ed directly.

These library routines cover a range of functionalities, including:

- Bytestream routines
 - Bytestream routines allow for file handling without having FD and SELECT statements for a file.
- File/Directory routines
 - File/Directory routines allow for the creation and deletion of files and directories, and various other functionalities such as COPY functionality, and RENAME functionality, and routines which can be used to retrieve information about given files.
- Logical Operation routines,
 - Logical Operation routines allow for the application of logical operators such as AND, OR, XOR, NOT, to parameters provided by the user,
- Text String routines
 - Text String routines allow for the case transformation and justification of given strings of text,
- Linkage-oriented routines
 - Linkage-oriented allow the user to check the number of parameters passed from a CALLing program, as wells as parameter size,
- System-level routines,
 - System-level routines include calls to “SYSTEM”, as well as to “Sleep” routines, which can be used to pause the system.
- Debugging-oriented routines,
 - Debugging-oriented routines allow the user to pause the runtime so that it may be restarted in the debugger,
- Error and Exit routines
 - Error and Exit routines allow the user to direct control to routines set up to handle error- and exit- procedure code.
- Call-by-number routines
 - Call-by-number routines provide compatibility with the commonly used X”91”, X”F4” and X”F5” function.

Notes applying to Library Routines generally:

The General Format for callable Library Routines is:

```
CALL library-routine [ USING { parameter-n,... } ]  
                    [ GIVING call-status      ].
```

- *library-routine* is represented as an alphanumeric literal, or, in some cases, as a variable. Note that the “SYSTEM” routine, and library routines that are prefixed with “C\$” (example: C\$COPY) can be dynamically called. In a dynamic CALL, the library name is replaced by a variable name, which can be populated anywhere prior to the execution of the CALL statement.

Format 1

```
CALL “library routine” USING parameter-1, GIVING call-status.
```

Format 2

```
77 function-call PIC X(20).
```

```
MOVE “library routine” to function-call.
```

```
CALL function-call.
```

- The GIVING phrase is always optional. If the GIVING phrase is omitted, then the return value from the CALL to the library routine will be returned to the return-code register.
- In this document, supported Library Routines are listed alphabetically.
- Runtime Abort Codes
 - CALLs to Library Routines return a 0 when successful. For the behavior of a Library Routine when unsuccessful, check the documentation for the specific routine. Most commonly, you will find that file-oriented routines return file-status codes that can be instructive as to why an operation has failed, and other routines will return a 128 (Call Unresolved).
 - If your CALL returns an error code that is not mentioned in this documentation, check the full list of runtime abort codes in the COBOL-IT Reference Manual for further clarification.

C\$CALLERNAME

C\$CALLERNAME returns the program-id name of the CALLing program in a CALL'ed program.

Usage

```
CALL "C$CALLERNAME" USING calling-program-name.
```

Parameters

calling-program-name PIC X(n)

Syntax

calling-program-name is the program-id name of the program that CALL'ed the currently running program.

General Rules

1. The function does not update return-code.
2. When executed from within a CALL'ed program, the function returns the program-id name of the CALLing program. When executed from within a MAIN program that has not been called, the function returns the string "UNDEFINED".

Code Sample

```
* C$CALLERNAME
77 CALLINGPGM-NAME                    PIC X(30) .
. . .
*
. . .
  INITIALIZE CALLINGPGM-NAME .
  CALL "C$CALLERNAME" USING CALLINGPGM-NAME .
*
```

C\$CHDIR

C\$CHDIR changes the current working directory to the directory named in new-dir.

Usage

```
CALL "C$CHDIR" USING new-dir,  
                  GIVING call-status.
```

Parameters

new-dir PIC X(n)
call-status PIC S9(9).

Syntax

new-dir is the name of the new current working directory.
call-status is updated with the success or failure status.

General Rules

3. When the function is successful, call-status is set to 0.
4. When the function fails, call-status is set to 128.

Code Sample

```
* C$CHDIR  
77 NEW-DIR            PIC X(6) .  
77 CALL-STATUS        PIC S9(9) .  
. . .  
*  
. . .  
    INITIALIZE NEW-DIR, CALL-STATUS .  
    MOVE "SUBDIR" TO NEW-DIR .  
  

```

C\$COPY

C\$COPY copies a source file to a target filename.

Usage

```
CALL "C$COPY"  
    USING src-file,  
         target-file,  
         file-type,  
    GIVING call-status.
```

Parameters

source-file PIC X(n)
target-file PIC X(n)
file-type PIC X.
call-status PIC S9(9).

Syntax

source-file is the original file to be copied.
target-file is the new file which is a copy of source-file.
file-type Permissible values are:
"S" Sequential
"R" Relative
"I" Indexed
"T" Text
call-status is updated with the success or failure status.

General Rules

1. When the function is successful, call-status is set to 0.
2. When the function fails, call-status is set to 128.

Code Sample

```
* C$COPY VARIABLES  
77 SRC-FILE PIC X(11).  
77 TARGET-FILE PIC X(10).  
77 FILE-TYPE PIC X VALUE "T".  
77 CALL-STATUS PIC S9(9).  
...  
    INITIALIZE CALL-STATUS.  
    MOVE "LIBTEST.CBL" TO SRC-FILE.  
    MOVE "NEWFIL.CBL" TO TARGET-FILE.  
  
    CALL "C$COPY" USING SRC-FILE,  
                     TARGET-FILE,  
                     FILE-TYPE,  
    GIVING CALL-STATUS.
```

C\$DEBUG

C\$DEBUG is a library routine which can be called using either the PID of the runtime session, or the value of the environment variable COB_DEBUG_ID. Prior to calling C\$DEBUG, the program should acquire the value of the PID / COB_DEBUG_ID.

You may acquire the value of the PID of the runtime session by calling the C\$PID library routine, using a PIC 9(n) parameter. The parameter must be numeric, and large enough to hold the value of the Process ID.

For example :

```
77 ws-pid    PIC 9(5).
```

```
.....
```

```
CALL « C$PID » USING ws-pid.
```

```
CALL « C$DEBUG » USING ws-pid.
```

You may also call C\$DEBUG USING the value of the runtime environment variable COB_DEBUG_ID. Using the runtime environment variable COB_DEBUG_ID to hold the value of this parameter has an advantage if you prefer to set the value of the parameter yourself. Acquire the value of COB_DEBUG_ID programmatically before calling the C\$DEBUG library routine. The parameter must be numeric, and large enough to hold the value of the value of the runtime environment variable COB_DEBUG_ID.

For example :

```
77 ws-did    PIC 9(5).
```

```
.....
```

```
ACCEPT ws-did FROM ENVIRONMENT « COB_DEBUG_ID ».
```

```
CALL «C$DEBUG » USING ws-did.
```

After a call to C\$DEBUG is made, the executing program, or subprogram is paused. In this state, the COBOL-IT Debugger may be attached to this runtime process from the COBOL-IT Developer Studio.

Key concepts

- In order to attach to the COBOL-IT Debugger, the program containing the call to C\$DEBUG library routine must be compiled with `-g`.
- The COBOL-IT Developer Studio will request the location of the source file associated with the program/subprogram that has been paused by the C\$DEBUG command, for purposes of debugging.
- The COBOL-IT Developer Studio attaching to the paused runtime session requires a COBOL Project, and requires that some configuration. Recommended settings are :
 - Window>Preferences>Run/Debug>Perspectives>Open the associated perspective when launching (Always)

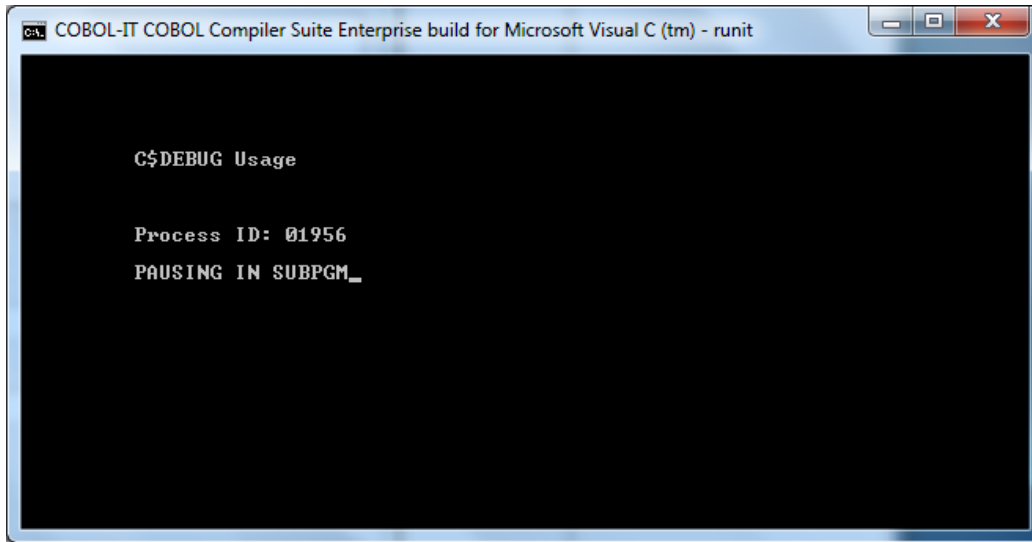
Demonstration

For our test, we have a program, `debugid.cbl`, which calls a subprogram, `subpgm.cbl`, which retrieves the PID of the runtime session, and then calls C\$DEBUG to pause the runtime session. We will run these programs from a batch file, as follows :

**Launch and pause the runtime using “C\$DEBUG”
runit.bat**

```
SET COB_LIBRARY_PATH=.\object  
cobcrun debugid
```

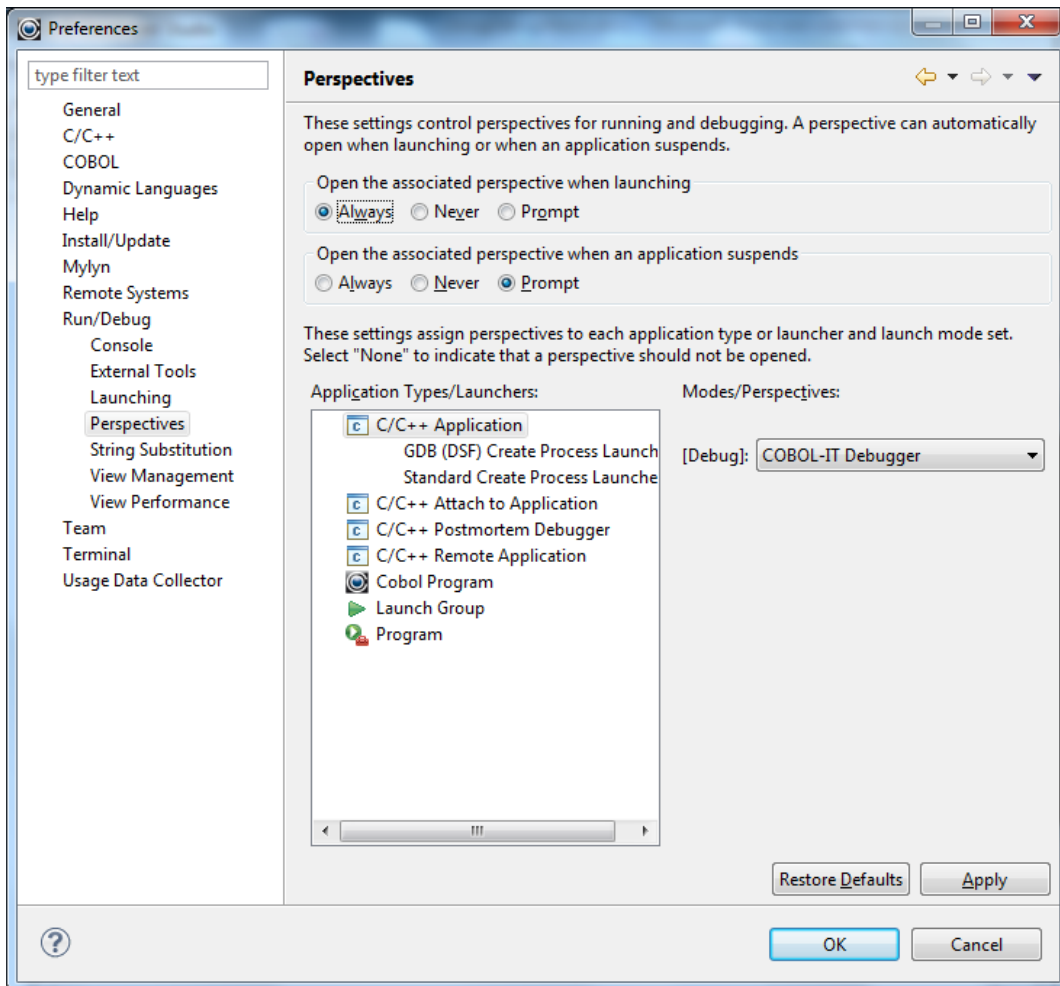
This will return the screen below. Note that in your case, the Process ID will likely be different.



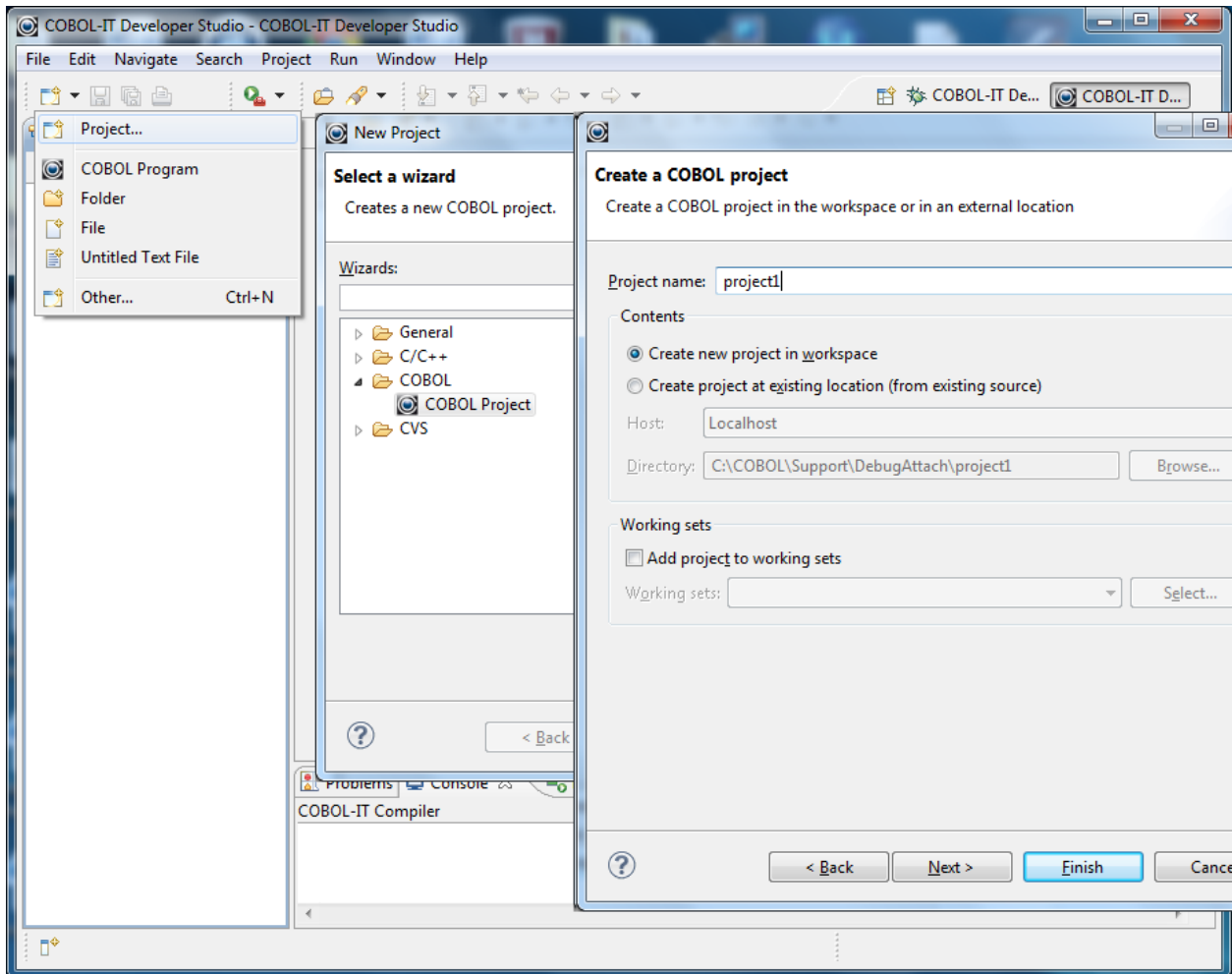
Attach the Debugger from the Developer Studio

When this program is paused, we will open the Developer Studio, configure :

Window>Preferences>Run/Debug>Perspectives>Open the associated perspective when launching (Select Always) .

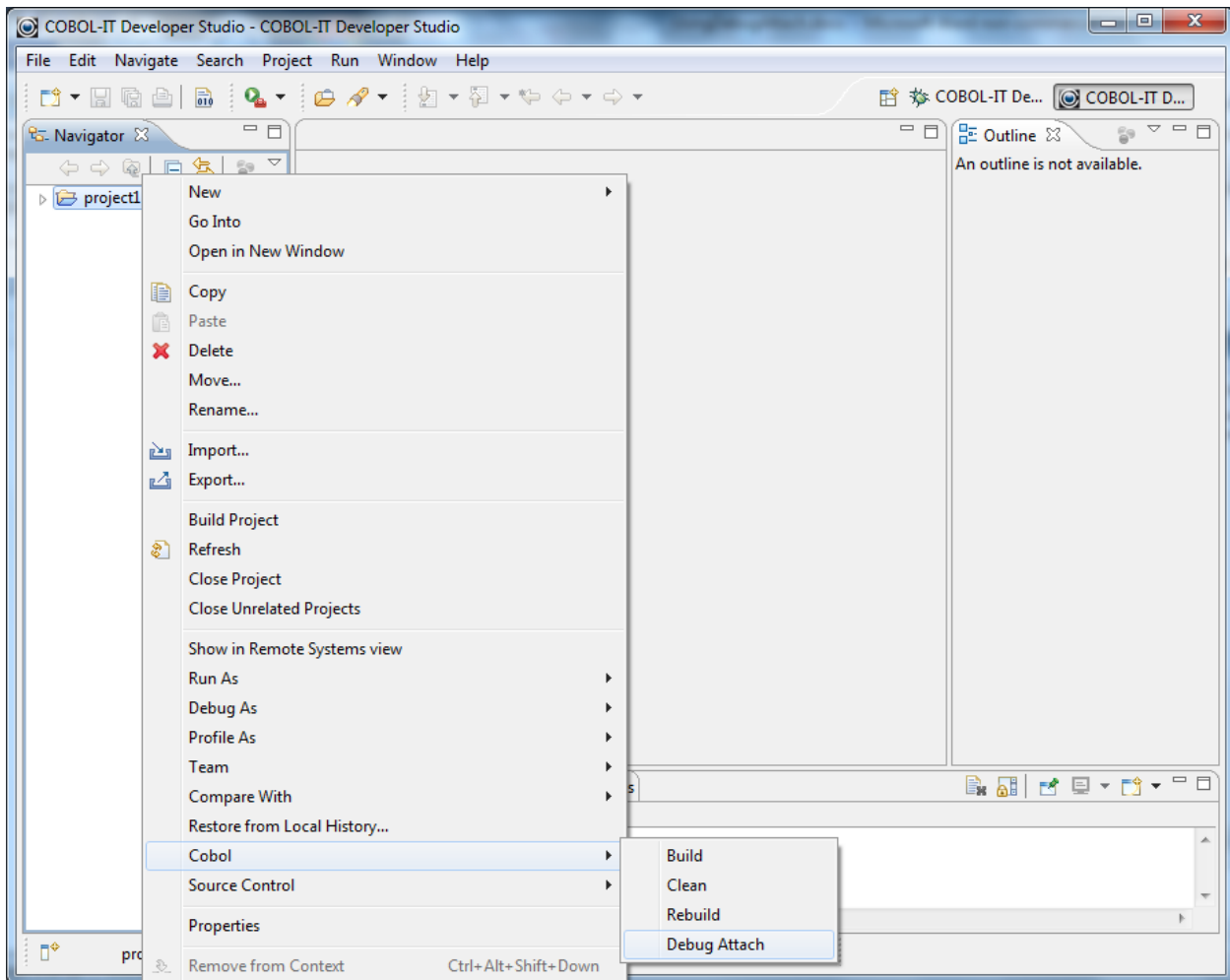


Create a new project



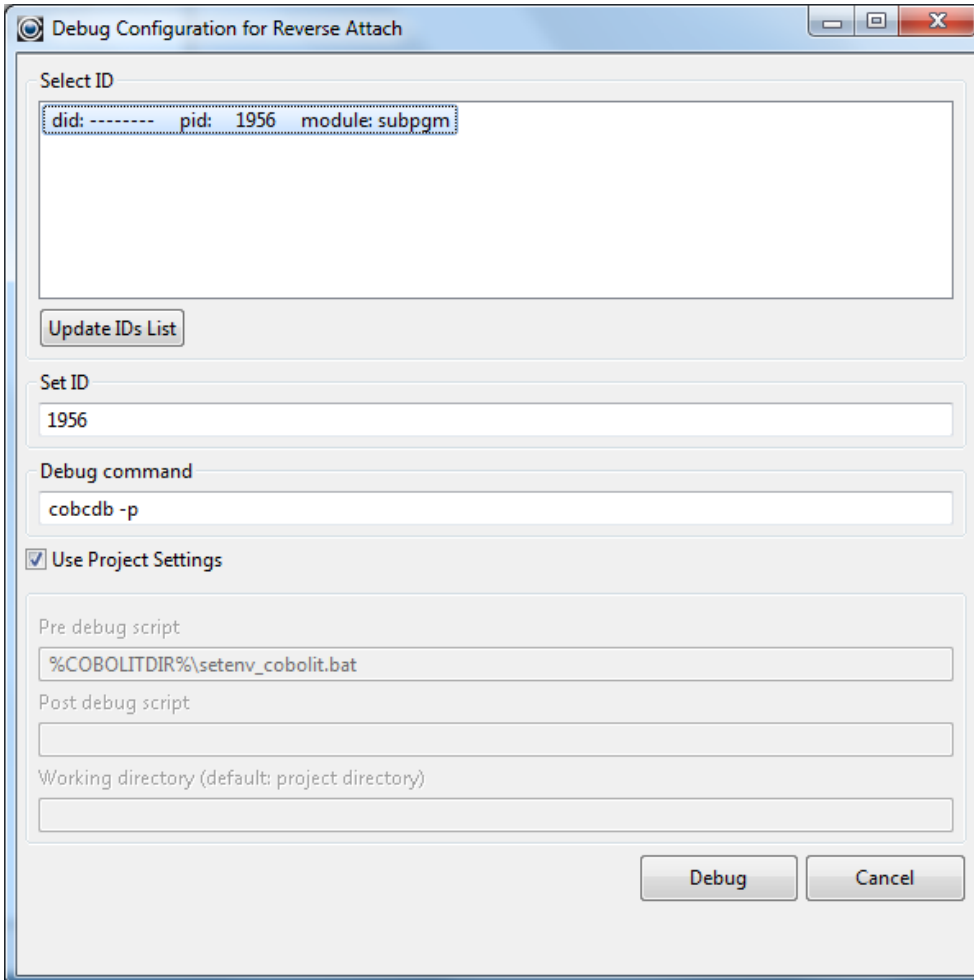
Select Debug Attach Function

In the Navigator Window, right-click on the Project, and select COBOL>Debug Attach from the dropdown list :



Select ID

In the Debug Configuration for Reverse Attach Window, select the entry with the PID that matches the PID of the paused runtime session . Click Debug.



Debug Configuration for Reverse Attach

Select ID

did: ----- pid: 1956 module: subpgm

Update IDs List

Set ID

1956

Debug command

cobcdb -p

Use Project Settings

Pre debug script

%COBOLITDIR%\setenv_cobolit.bat

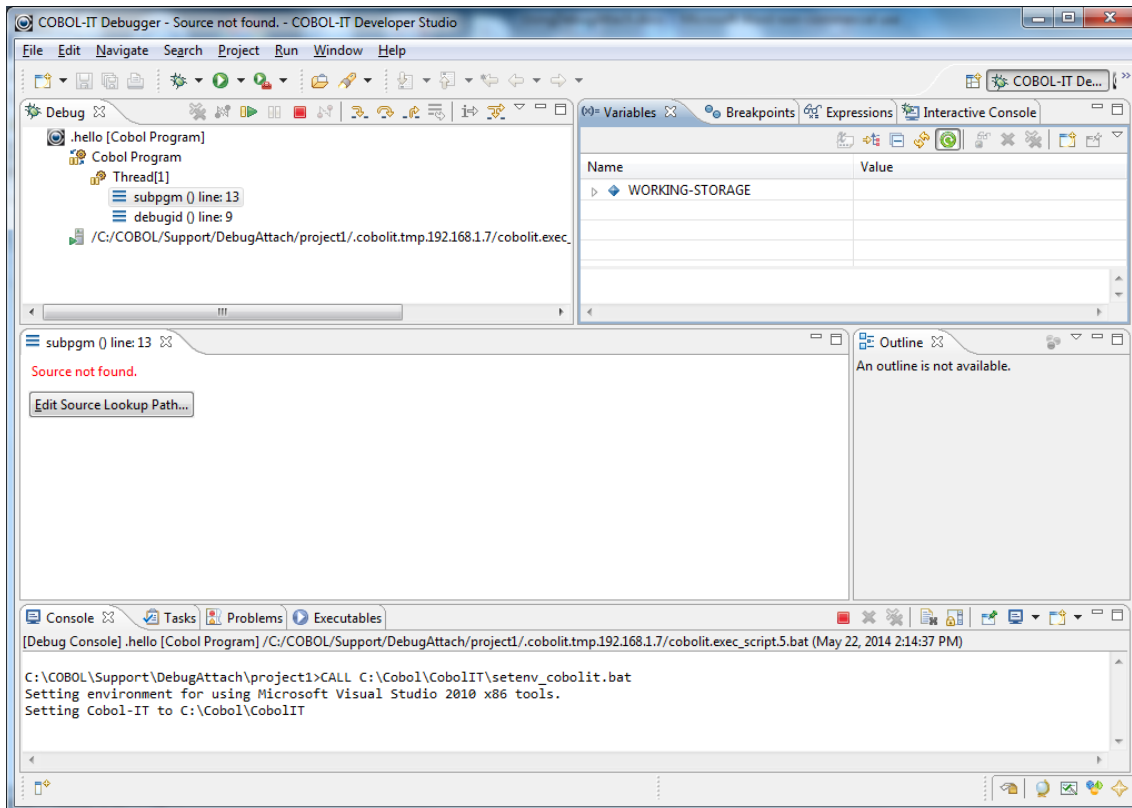
Post debug script

Working directory (default: project directory)

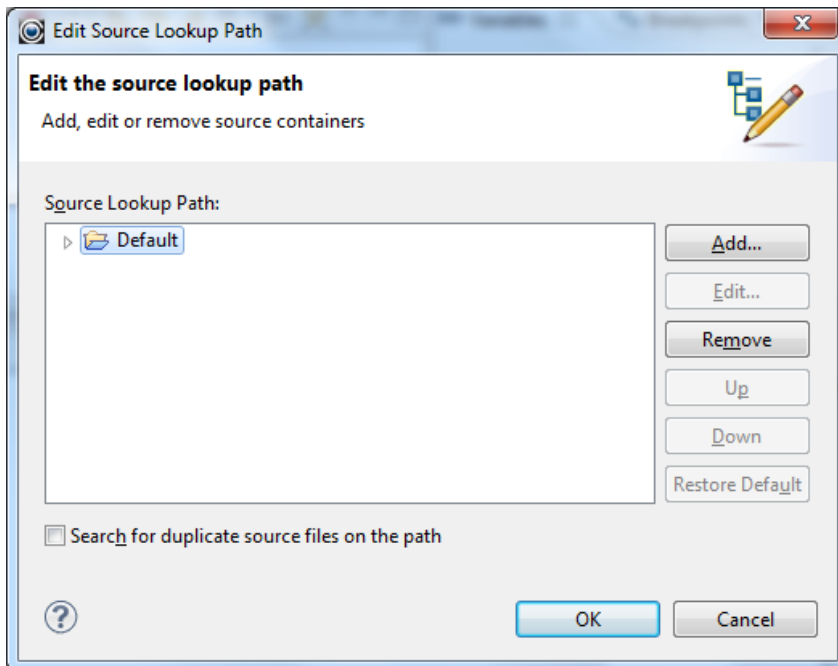
Debug Cancel

Edit Source Lookup Path

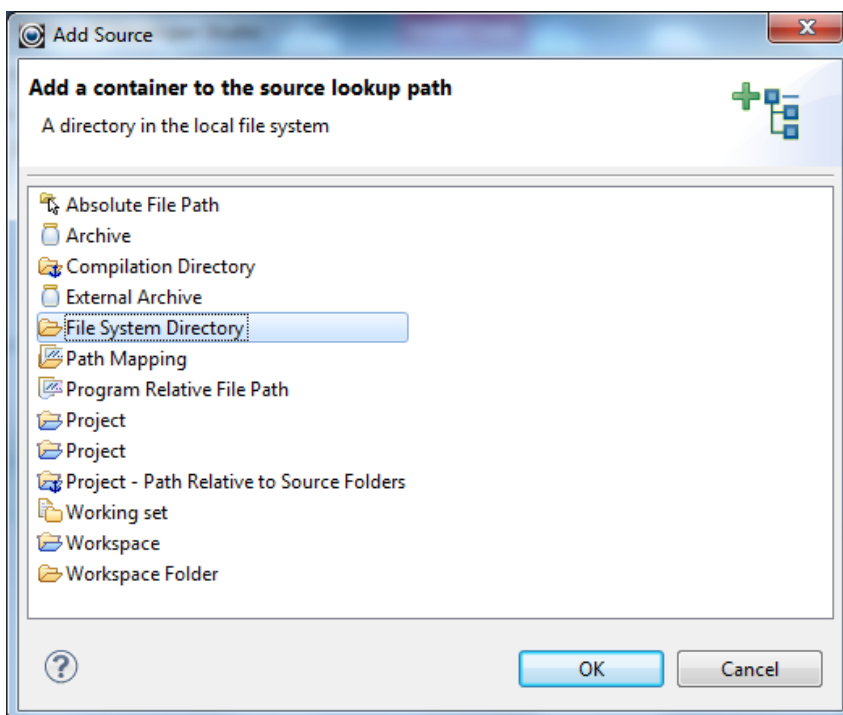
The Developer Studio opens in the Debugger Perspective. Note that there is a message, in red, that **Source Not Found**. To associate the the source code of subpgm.cbl with the project, click on the Edit Source Lookup Path... button.



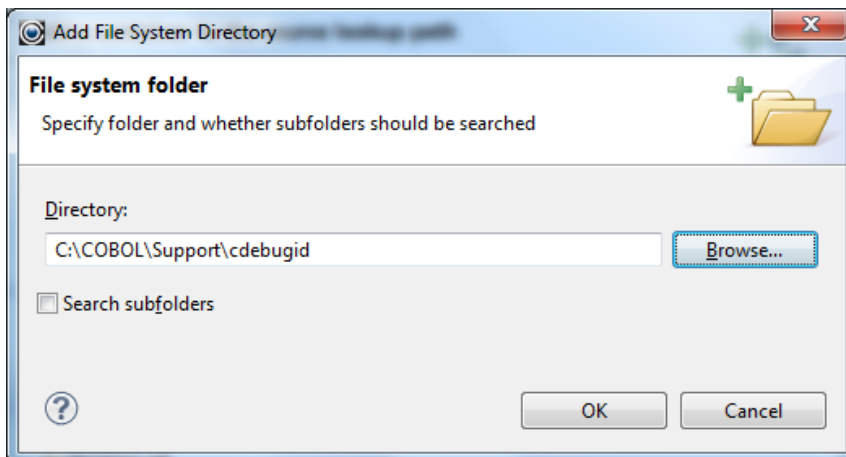
On the Edit the Source Lookup Path Screen, the Default setting is your current Project Path. If the source files are not in your Project Path, click the Add button.



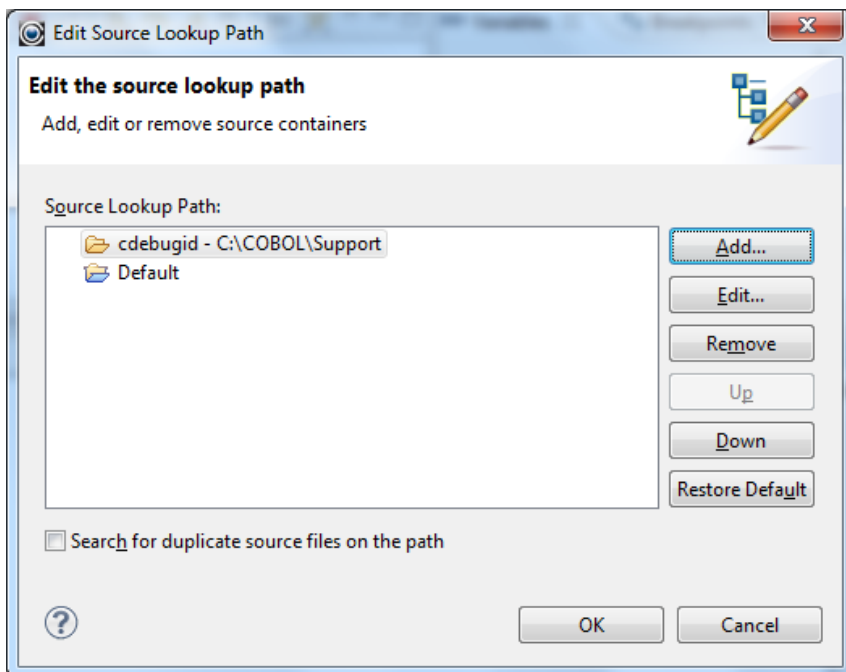
Select File System Directory, Click Ok.



Use the Browse button to locate the Source Location

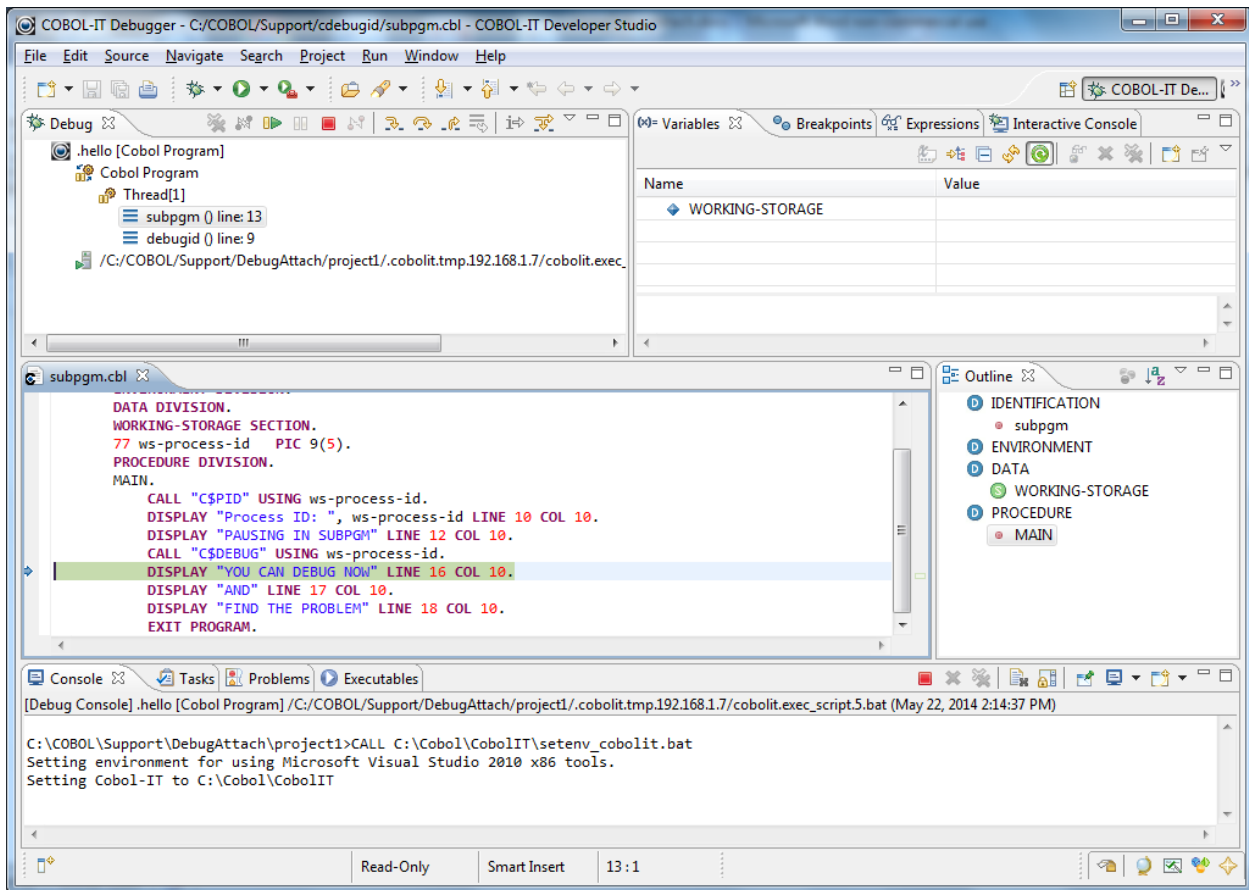


Your selection will appear in the Source Lookup Path window . Click OK.

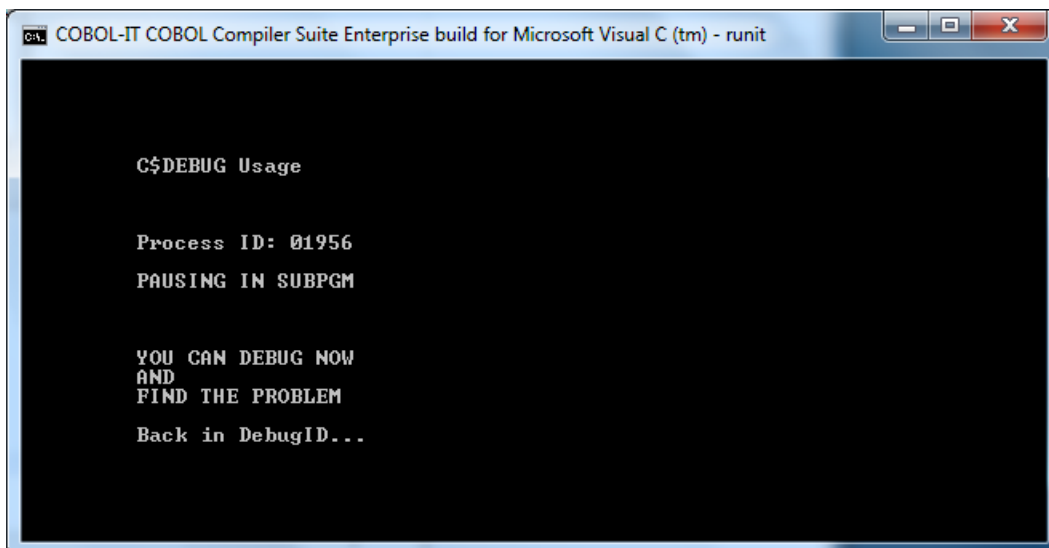


Debug in the Developer Studio

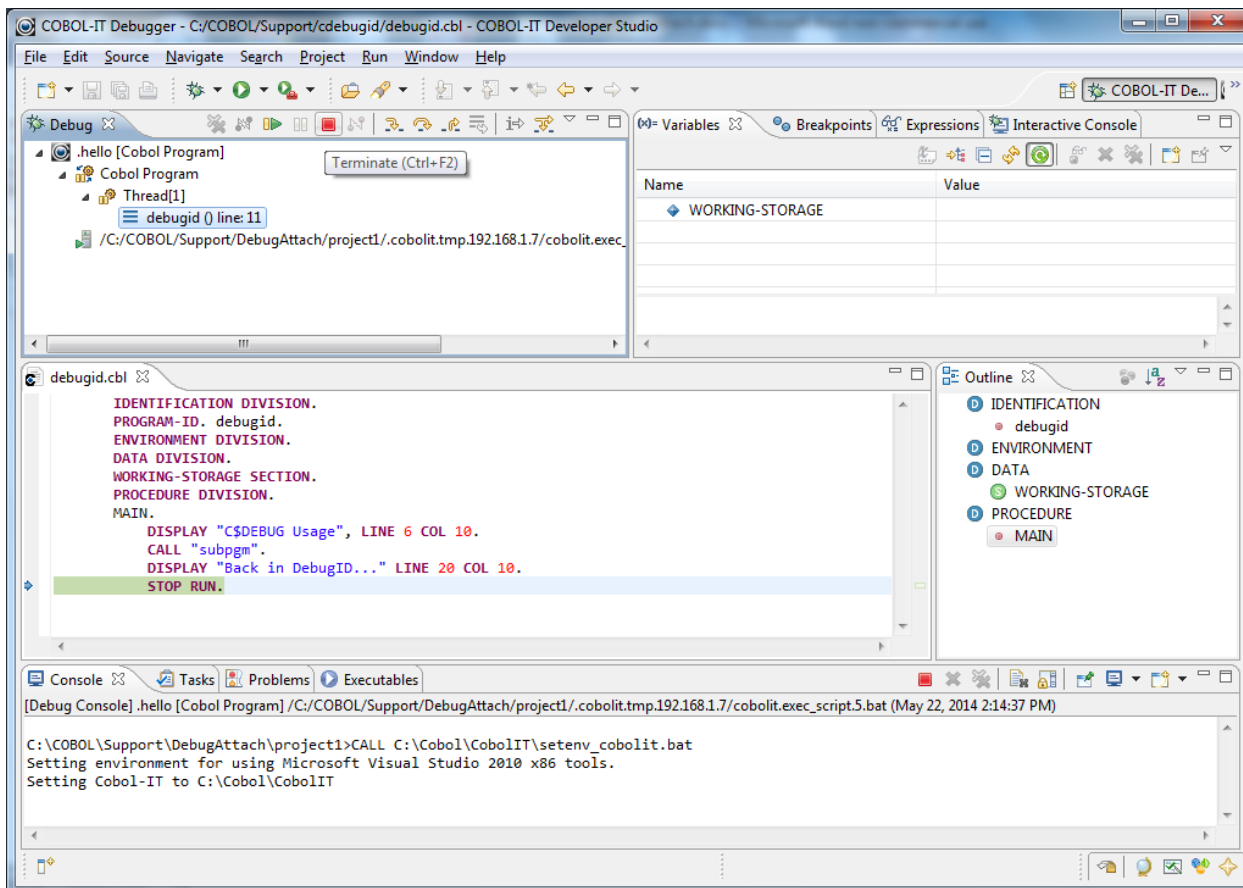
You are now ready to debug in the Developer Studio :



Use the Debugger toolbar buttons to debug your program.



Terminate the Debugger by clicking on the Terminate button.



Programs used in this sample

debugid.cbl

```

IDENTIFICATION DIVISION.
    PROGRAM-ID. debugid.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
MAIN.
    DISPLAY "C$DEBUG Usage", LINE 6 COL 10.
    CALL "subpgm".
    DISPLAY "Back in DebugID..." LINE 20 COL 10.
    STOP RUN.
    
```


subpgm.cbl

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. subpgm.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
77 ws-process-id PIC 9(5).  
PROCEDURE DIVISION.  
MAIN.  
  CALL "C$PID" USING ws-process-id.  
  DISPLAY "Process ID: ", ws-process-id LINE 10 COL 10.  
  DISPLAY "PAUSING IN SUBPGM" LINE 12 COL 10.  
  CALL "C$DEBUG" USING ws-process-id.  
  DISPLAY "YOU CAN DEBUG NOW" LINE 16 COL 10.  
  DISPLAY "AND" LINE 17 COL 10.  
  DISPLAY "FIND THE PROBLEM" LINE 18 COL 10.  
  EXIT PROGRAM.
```

runit.bat

```
SET COB_LIBRARY_PATH=.\object  
cobc -o .\object -g debugid.cbl  
cobc -o .\object -g subpgm.cbl  
cobcrun debugid
```

C\$DELETE

C\$DELETE deletes a source file.

Usage

```
CALL "C$DELETE" USING DEL-FILE-NAME,  
                     DEL-FILE-TYPE,  
                     GIVING call-status.
```

Parameters

```
del-file-name PIC X(n)  
del-file-type PIC X.  
call-status   PIC S9(9).
```

Syntax

del-file-name is the file to be deleted.
del-file-type is the type of file. Permissible values are:

“S” Sequential

“R” Relative

“I” Indexed

“T” Text

call-status is updated with the success or failure status.

General Rules

1. When the function is successful, call-status is set to 0.
2. When the function fails, call-status is set to 128.

Code Sample

```
* C$DELETE
77 DEL-FILE-NAME    PIC X(20) .
77 DEL-FILE-TYPE    PIC X VALUE "T" .
77 CALL-STATUS      PIC S9(9) .
...
      INITIALIZE CALL-STATUS .
      MOVE "NEWFIL.CBL" TO DEL-FILE-NAME .

      CALL "C$DELETE" USING DEL-FILE-NAME,
                          DEL-FILE-TYPE,
                          GIVING CALL-STATUS .

*
```

C\$FILEINFO

C\$FILEINFO retrieves the size and date/time stamp of a file.

Usage

```
CALL "C$FILEINFO" USING file-name,
                       file-info,
                       GIVING call-status.
```

Parameters

info-file-name	PIC X(n)
info-file-info	A group item with the following elements:
file-size	PIC X(8) COMP-X.
file-date	PIC 9(8) COMP-X.
file-time	PIC 9(8) COMP-X.
call-status	PIC S9(9).

Syntax

info-file-name	must be terminated with a space.
info-file-info	receives data if file-name exists.

call-status is updated with the success or failure status.

General Rules

1. When the function is successful, call-status is set to 0.
2. When the function fails, call-status is set to 35.

Code Sample

```
* C$FILEINFO
77 INFO-FILENAME          PIC X(16) .
01 INFO-FILE-INFO.
   05 FILE-SIZE    PIC X(8) COMP-X.
   05 FILE-DATE    PIC 9(8) COMP-X.
   05 FILE-TIME    PIC 9(8) COMP-X.
77 CALL-STATUS      PIC S9(9) .

...
   INITIALIZE CALL-STATUS.
   MOVE "NEWFIL.CBL" TO INFO-FILENAME.

   CALL "C$FILEINFO" USING INFO-FILENAME,
                           INFO-FILE-INFO
                           GIVING CALL-STATUS.

*
```

C\$JUSTIFY

C\$JUSTIFY performs left/right/center justification of data by removing leading and/or trailing spaces.

Usage

```
CALL "C$JUSTIFY" USING SOURCE-DATA-1,  
JUSTIFY-TYPE.
```

Parameters

source-data-1 Any data type
justify-type PIC X

Syntax

source-data-1 contains the data to be justified.
justify-type indicates whether justification is L (left), R (right), or C (center).

General Rules

The source-data item is transformed by the routine, as leading/trailing spaces are manipulated.

Code Sample

```
* C$JUSTIFY  
77 SOURCE-DATA-1 PIC X(20).  
77 JUSTIFY-TYPE PIC X.  
...  
MOVE " ABCDEFGHIJ " TO SOURCE-DATA-1.  
MOVE "L" TO JUSTIFY-TYPE.  
  
CALL "C$JUSTIFY" USING SOURCE-DATA-1, JUSTIFY-TYPE.  
  
MOVE "R" TO JUSTIFY-TYPE.  
  
CALL "C$JUSTIFY" USING SOURCE-DATA-1, JUSTIFY-TYPE.  
  
MOVE "C" TO JUSTIFY-TYPE.  
  
CALL "C$JUSTIFY" USING SOURCE-DATA-1, JUSTIFY-TYPE.  
  
...  
*
```

C\$MAKEDIR

C\$MAKEDIR creates a directory.

Usage

```
CALL "C$MAKEDIR" USING dir-name  
GIVING call-status.
```

Parameters

make-dir-name	PIC X(n)
call-status	PIC S9(9).

Syntax

make-dir-name	is the name of the directory to be created.
call-status	is updated with the success or failure status.

General Rules

1. When the function is successful, call-status is set to 0.
2. When the function fails, call-status is set to 128.
3. make-dir-name can contain full-path or relative-path notations.
4. C\$MAKEDIR cannot be used to create a series of sub-directories.

Code Sample

```
*  
77 MAKE-DIR-NAME      PIC X(20).  
77 CALL-STATUS        PIC S9(9).  
...  
    INITIALIZE CALL-STATUS.  
    MOVE "SUBDIR" TO MAKE-DIR-NAME.  
  
    CALL "C$MAKEDIR" USING MAKE-DIR-NAME  
                        GIVING CALL-STATUS.  
    ...  
*
```

C\$NARG

C\$NARG returns the number of parameters that have been passed through linkage to the executing program.

Usage

```
CALL "C$NARG" USING number-of-parameters.
```

Parameters

number-of-parameters Numeric data item.

Syntax

number-of-paramters is returned as a result of the call.

General Rules

1. Number-of-parameters is returned the number of USING items in the CALL statement that are in the CALLing program.

Code Sample

```
*
77 NUM-PARAMS      PIC S9.
LINKAGE SECTION.
01 LK-NAME      PIC X(25) .
01 LK-ADDR      PIC X(25) .
01 LK-CUSTOMERID      PIC X(5) .
...
PROCEDURE DIVISION USING LK-NAME, LK-ADDR, LK-CUSTOMERID.
MAIN.
    CALL "C$NARG" USING NUM-PARAMS.
    ...
*
```

C\$PARAMSIZE

C\$PARAMSIZE takes the ordinal number of a parameter as input in a USING phrase, and returns its size in bytes in a parameter named in the GIVING phrase.

Usage

```
CALL "C$PARAMSIZE" USING param-num,  
                      GIVING param-size.
```

Parameters

param-num	PIC 9(n), or any numeric data item.
param-size	PIC 9(n).

Syntax

param-num	represents the ordinal position of parameter
param-size	receives from the function the number of bytes in the named data item

General Rules

There are no general rules.

Code Sample

```
*  
77 FIRST-PARAM-SIZE PIC 99.  
LINKAGE SECTION.  
01 LK-NAME          PIC X(25).  
01 LK-ADDR          PIC X(25).  
01 LK-CUSTOMERID   PIC X(5).  
PROCEDURE DIVISION USING LK-NAME, LK-ADDR, LK-CUSTOMERID.  
MAIN.  
    CALL "C$PARAMSIZE" USING 1, GIVING FIRST-PARAM-SIZE.  
    ...  
*
```

C\$PID

C\$PID retrieves the Process ID of the current process.

Note that C\$PID is not currently available on Windows platforms.

Usage

```
CALL "C$PID" USING process-id.
```

Parameters

process-id PIC 9(n).

Syntax

process-id is a numeric data item which must be large enough to hold the process-id.

Code Sample

```
*  
77 PROCESS-ID    PIC 9(7) .  
...  
    CALL "C$PID" USING PROCESS-ID .  
    ...  
*
```


C\$\$SLEEP

C\$\$SLEEP causes the program to “sleep” in a defined interval that is represented in seconds, or fractions of seconds.

Usage

```
CALL "C$$SLEEP" USING number-seconds.
```

Parameters

number-seconds Numeric literal or data item

Syntax

number-seconds is the elapsed time in seconds to sleep

Code Sample

```
*  
...  
CALL "C$$SLEEP" USING 1.5.  
...  
*
```

C\$TOLOWER

C\$TOLOWER translates a text string into lower-case.

Usage

```
CALL "C$TOLOWER" USING ctl-source-data,  
                       VALUE ctl-source-length.
```

Parameters

ctl-source-data PIC X(10).
ctl-source-length USAGE UNSIGNED-INT.

Syntax

ctl-source-data is the data to translate to upper-case.
ctl-source-length is the number of characters to translate.

General Rules

1. The string in ctl-source-data is transformed by the operation, with all characters being translated to lower-case.
2. Return-code is not updated following the operation.

Code Sample

```
*  
77 CTL-SOURCE-DATA     PIC X(10).  
77 CTL-SOURCE-LENGTH  USAGE UNSIGNED-INT.  
...  
   MOVE "ABCDEFGHIJ" TO CTL-SOURCE-DATA.  
   MOVE 10 TO CTL-SOURCE-LENGTH.  
   CALL "C$TOLOWER"  
       USING CTL-SOURCE-DATA,  
           VALUE CTL-SOURCE-LENGTH.  
   ...  
*
```

C\$TOUPPER

C\$TOUPPER translates a text string into upper-case.

Usage

```
CALL "C$TOUPPER" USING ctu-source-data,  
                       VALUE ctu-source-length.
```

Parameters

ctu-source-data	PIC X(n)
ctu-source-length	USAGE UNSIGNED-INT, or a numeric literal

Syntax

ctu-source-data	is the data to translate to upper-case.
ctu-source-length	is the number of characters to translate.

General Rules

1. The string in ctu-source-data is transformed by the operation, with all characters being translated to upper-case.
2. Return-code is not updated following the operation.

Code Sample

```
*  
77 CTU-SOURCE-DATA    PIC X(10).  
77 CTU-SOURCE-LENGTH  USAGE UNSIGNED-INT.  
...  
    MOVE "ABCDEFGHIJ" TO CTU-SOURCE-DATA.  
    MOVE 10 TO CTU-SOURCE-LENGTH.  
    CALL "C$TOUPPER"  
        USING CTU-SOURCE-DATA,  
              VALUE CTU-SOURCE-LENGTH.  
    ...  
*
```

CBL_ALLOC_DYN_MEM

CBL_ALLOC_DYN_MEM dynamically allocates memory, returning an address (MEMORY-POINTER) and a size..

Usage:

```
CALL "CBL_ALLOC_DYN_MEM" USING memory-pointer,
    BY VALUE memory-size,
    BY VALUE memory-flags,
    RETURNING call-status.
```

Syntax:

memory-pointer is a data element described as USAGE POINTER. Memory-pointer must be described as an 01-level data item. The memory allocated is not initialized.

memory-size represents the size of the memory being allocated, in bytes. memory size is described as PIC x(4) comp-5.

memory-flags describe the type of memory being allocated. This is done by setting of bits on a 4-byte field. See the table below for guidelines on setting bits to describe a type of memory. memory-flags is described as PIC x(4) comp-5.

bits 0-1	Reserved. Must be set to 0
bit 2	Allocate this memory independently from any calling program
bits 4-31	Reserved. Must be set to 0.

If bit 2 is not set, the memory allocated is freed when the currently running program is cancelled from memory. If bit 2 is set, the memory allocated is freed when the runtime session terminates. Memory allocated can also be freed by the CBL_FREE_DYN_MEM library routine.

call-status is a return code. Call-status settings are:

0	successful allocation of memory
157	unsuccessful allocation of memory
181	contradictory setting of flags

General Rules

The General Rules are described above.

CBL_ALLOC_MEM

CBL_ALLOC_MEM dynamically allocates memory, returning an address (MEMORY-POINTER) and a size..

Usage:

```
CALL "CBL_ALLOC_MEM" USING memory-pointer,
    BY VALUE memory-size,
    BY VALUE memory-flags,
    RETURNING call-status.
```

Syntax:

memory-pointer	is a data element described as USAGE POINTER. Memory-pointer must be described as an 01-level data item.
memory-size	represents the size of the memory being allocated, in bytes. memory size is described as PIC x(4) comp-5.
memory-Flags	describe the type of memory being allocated. This is done by setting of bits on a 4-byte field. See the table below for guidelines on setting bits to describe a type of memory. Memory-flags is described as PIC x(4) comp-5.
bit 0	Allocate the memory as shared memory
bit 1	Reserved. Must be set to 0.
bit 2	Allocate this memory independently from any calling program. If bit 3 is set it will be freed automatically when the calling thread ends. If bit 3 is unset it will be freed when the run-unit ends.
bit 3	Allocate this memory as thread local. If bit 2 is unset and there is a direct or indirect (in a mixed language environment) calling COBOL program, it will be freed when the calling program is canceled or the thread ends - whichever comes first.
bits 4-31	Reserved. Must be set to 0.

call-status	is a return code. Call-status settings are:
0	successful allocation of memory
157	unsuccessful allocation of memory
181	contradictory setting of flags

General Rules

- Use the memory allocation functions with caution. Updates to shared memory allocated to this function are not serialized or protected by the run-time system. It is advised that you use semaphores to maintain the integrity of the data.
- If the memory is allocated by a thread, it is freed when the thread terminates.
- Bit 1 and bit 2 or bit 3 are mutually exclusive. The contradictory setting of flags (error 181) is returned otherwise.
- If there is no calling program (directly or indirectly in a mixed language environment) bit 2 is ignored.
- If a COBOL program is directly or indirectly the caller of CBL_ALLOC_MEM, then all standard memory allocated by CBL_ALLOC_MEM is freed when the program that allocated it is canceled (logically or physically) if bit 2 is not set.



CBL_CHANGE_DIR

CBL_CHANGE_DIR changes the current working directory to the directory named in new-dir.

Usage

```
CALL "CBL_CHANGE_DIR" USING new-dir  
GIVING call-status.
```

Parameters

new-dir PIC X(n)
call-status PIC S9(9).

Syntax

new-dir is the name of the new current working directory.
call-status is updated with the success or failure status.

General Rules

1. When the function is successful, call-status is set to 0.
2. When the function fails, call-status is set to 128.

Code Sample

```
*  
77 NEW-DIR      PIC XX.  
77 CALL-STATUS PIC S9(9).  
...  
    MOVE ".." TO NEW-DIR.  
    CALL "CBL_CHANGE_DIR" USING NEW-DIR  
          GIVING CALL-STATUS.  
*
```

CBL_CHECK_FILE_EXIST

CBL_CHECK_FILE_EXIST checks to see if a file exists.

Usage

```
CALL "CBL_CHECK_FILE_EXIST" USING ccfе-file-name,  
                                ccfе-file-details.
```

Parameters

ccfe-file-name	PIC X(n)
ccfe-file-details	A group item with the following elements:
file-size	PIC X(8) COMP-X.
file-date	A group item with the following elements:
f-day	PIC X COMP-X.
f-month	PIC X COMP-X.
f-year	PIC X(2) COMP-X.
file-time	A group item with the following elements:
f-hours	PIC X COMP-X.
f-minutes	PIC X COMP-X.
f-seconds	PIC X COMP-X.
f-hundredths	PIC X COMP-X.

Syntax

ccfe-file-name	must be terminated with a space.
ccfe-file-details	is populated with data if file-name exists.

General Rules

When the function is successful, return-code is set to 0.

When the function fails, return-code is set to 35.

Code Sample

```
77 CCFE-FILENAME PIC X(15) .
01 CCFE-FILE-DETAILS .
   05 FILE-SIZE PIC X(8) COMP-X.
   05 FILE-DATE .
      10 F-DAY PIC X COMP-X.
      10 F-MONTH PIC X COMP-X.
      10 F-YEAR PIC X(2) COMP-X.
   05 FILE-TIME .
      10 F-HOURS PIC X COMP-X.
      10 F-MINUTES PIC X COMP-X.
      10 F-SECONDS PIC X COMP-X.
      10 F-HUNDREDTHS PIC X COMP-X.
   ...
   INITIALIZE RETURN-CODE.
   MOVE "NEWFIL.CBL " TO CCFE-FILENAME.

   CALL "CBL_CHECK_FILE_EXIST"
```

```
USING CCFE-FILENAME,  
      CCFE-FILE-DETAILS.
```

CBL_CLOSE_FILE

CBL_CLOSE_FILE closes a file that was created with either the CBL_OPEN_FILE or CBL_CREATE_FILE function.

Usage

```
CALL "CBL_CLOSE_FILE" using my-file-handle.
```

Parameters

my-file-handle PIC X(4) COMP-5.

Syntax

my-file-handle is returned after a successful Create, or Open.

General Rules

1. When the function is successful, return-code is set to 0.
2. When the function fails, return-code is set to an ANSI-74 file-status code according to the nature of the error.

Code Sample

```
*  
77 MY-FILE-HANDLE PIC X(4) COMP-5.  
...  
CALL "CBL_CLOSE_FILE" USING MY-FILE-HANDLE.  
...  
*
```

CBL_COPY_FILE

CBL_COPY_FILE copies a source file to a target filename.

Usage

```
CALL "CBL_COPY_FILE" USING CCF-SOURCE-FILE,  
                           CCF-TARGET-FILE.
```

Parameters

ccf-source-file	PIC X(n)
ccf-target-file	PIC X(n)

Syntax

ccf-source-file	is the original file to be copied.
ccf-target-file	is the new file which is a copy of source-file.

General Rules

1. When the function is successful, return-code is set to 0.
2. When the function fails, return-code is set to -1.

Code Sample

```
*  
77 CCF-SOURCE-FILE      PIC X(20) .  
77 CCF-TARGET-FILE     PIC X(20) .  
* * *  
    MOVE "COPYFIL.CBL" TO CCF-SOURCE-FILE .  
    MOVE "NEWFIL.CBL" TO CCF-TARGET-FILE .  
    CALL "CBL_COPY_FILE" USING CCF-SOURCE-FILE ,  
                               CCF-TARGET-FILE .  
*
```

CBL_CREATE_DIR

CBL_CREATE_DIR creates a directory.

Usage

```
CALL "CBL_CREATE_DIR" USING CCD-DIR-NAME.
```

Parameters

ccd-dir-name PIC X(n)

Syntax

ccd-dir-name is the name of the directory to be created.

General Rules

1. When the function is successful, return-code is set to 0.
2. When the function fails, return-code is set to 128.
3. ccd-dir-name can contain full-path or relative-path notations.
4. If path notations are part of dir-name, then the parent directory to the last directory must exist.
5. CBL_CREATE_DIR cannot be used to create a series of sub-directories.

Code Sample

```
*  
77 CCD-DIR-NAME            PIC X(20) .  
...  
    INITIALIZE RETURN-CODE .  
    MOVE "SUBDIR" TO CCD-DIR-NAME .  
  
    CALL "CBL_CREATE_DIR" USING CCD-DIR-NAME .  
*
```

CBL_CREATE_FILE

CBL_CREATE_FILE creates a sequential file with READ/WRITE permissions described by the parameters passed.

Usage

```
CALL "CBL_CREATE_FILE" USING my-file-name,
                             my-file-permissions,
                             my-file-restrictions,
                             my-device,
                             my-file-handle.
```

Parameters

my-file-name	PIC X(n).
my-file-permissions	PIC X COMP-X.
my-file-restrictions	PIC X COMP-X.
my-device	PIC X COMP-X.
my-file-handle	PIC X(4) COMP-5.

Syntax

my-file-name is a null-terminated character string.
file-permissions describes Read/Write permissions. It must be one of the following:

- | | | | |
|---|-----------|----|--------------------------------------|
| 1 | Read-only | 2 | Write-only |
| | | 3 | Read-Write |
| | | 64 | Read-Write for large files (> 4GB) |

my-file-restrictions describes Read/Write restrictions.

It must be one of the following:

- | | |
|---|----------------------------|
| 0 | Write-only |
| 1 | No write |
| 2 | No read |
| 3 | No read/write restrictions |

my-device must be set to 0.

my-file-handle is stored after a successful Create, or Open.

General Rules

When the function is successful, return-code is set to 0.

When the function fails, return-code is set to an ANSI-74 file-status code according to the nature of the error.

Code Sample

```
*
* CBL_CREATE_FILE
77 MY-FILE-NAME          PIC X(11).
```

```
77 MY-ACCESS-MODE          PIC X COMP-X VALUE 3.
77 MY-DENY-MODE           PIC X COMP-X VALUE 3.
77 MY-DEVICE              PIC X COMP-X VALUE 0.
77 MY-FILE-HANDLE        PIC X(4) COMP-5.
...
    INITIALIZE RETURN-CODE.

    STRING "AAAAAA.TXT" DELIMITED BY SIZE,
           X"00", DELIMITED BY SIZE,
           INTO MY-FILE-NAME.

    CALL "CBL CREATE FILE"
        USING MY-FILE-NAME,
              MY-ACCESS-MODE,
              MY-DENY-MODE,
              MY-DEVICE,
              MY-FILE-HANDLE.
```

*

CBL_DEBUGBREAK

CBL_DEBUGBREAK is a synonym for C\$DEBUG. CBL_DEBUGBREAK is a library routine which can be called using either the PID of the runtime session, or the value of the environment variable COB_DEBUG_ID.

For example :

```
77 ws-pid    PIC 9(5).
```

```
.....
```

```
CALL « C$PID » USING ws-pid.
```

```
CALL « CBL_DEBUGBREAK » USING ws-pid.
```

For more details, see the documentation of the C\$DEBUG library routine.

CBL_DELETE_DIR

CBL_DELETE_DIR deletes the named directory.

Usage

```
CALL "CBL_DELETE_DIR" USING del-dir-name.
```

Parameters

del-dir-name PIC X(n).

Syntax

del-dir-name is the name of the directory to be deleted.

General Rules

When the function is successful, return-code is set to 0.

When the function fails, return-code is set to 128.

Code Sample

```
*  
77 DEL-DIR-NAME      PIC X(20) .  
...  
    INITIALIZE RETURN-CODE .  
    MOVE "SUBDIR" TO DEL-DIR-NAME .  
  
    CALL "CBL_DELETE_DIR" USING DEL-DIR-NAME .  
*
```

CBL_DELETE_FILE

CBL_DELETE_FILE deletes the named file.

Usage

```
CALL "CBL_DELETE_FILE" USING CDF-FILE-NAME.
```

Parameters

cdf-file-name PIC X(n).

Syntax

cdf-file-name is the name of the file to be deleted.

General Rules

1. When the function is successful, return-code is set to 0.
2. When the function fails, return-code is set to 128.

Code Sample

```
*  
77 CDF-FILE-NAME          PIC X(20) .  
...  
    INITIALIZE RETURN-CODE .  
    MOVE "README.TXT" TO CDF-FILE-NAME .  
  
    CALL "CBL_DELETE_FILE" USING CDF-FILE-NAME .  
...  
*
```

CBL_ERROR_PROC

CBL_ERROR_PROC installs or uninstalls an error procedure, which is run when a program-ending error occurs. The Error Routine allows the user to register procedures that will automatically be executed either when a program-ending error occurs.

Usage

```
call "CBL_ERROR_PROC" using error-proc-flag,  
error-proc-addr.
```

Parameters

- error-proc-flag
 - set to 0 to install error proc
 - set to 1 to uninstall error proc

```
01 ERROR-PROC-FLAG PIC X COMP-X VALUE 0.
```
- error-proc-addr address of error proc

```
01 ERROR-PROC-ADDR USAGE PROCEDURE-POINTER.
```
- error-proc-msg message from error
LINKAGE SECTION.

```
01 ERROR-PROC-MSG PIC X(ERROR-PROC-MSG-LEN).
```

Syntax

error-proc-flag	set to 0 to install error proc set to 1 to uninstall error proc
error-proc-addr	address of error proc
error-proc-msg	message returned through linkage

Code Sample

```
...  
78 ERROR-PROC-MSG-LEN VALUE 325.  
01 ERROR-PROC-FLAG PIC X COMP-X VALUE 0.  
01 ERROR-PROC-ADDR USAGE PROCEDURE-POINTER.  
01 STATUS-CODE PIC 9(4) COMP VALUE ZEROS.  
...  
LINKAGE SECTION.  
01 ERROR-PROC-MSG PIC X(ERROR-PROC-MSG-LEN).  
PROCEDURE DIVISION.  
MAIN.  
SET ERROR-PROC-ADDR TO ENTRY "ERROR-PROC".  
CALL "CBL_ERROR_PROC" USING ERROR-PROC-FLAG,
```

ERROR-PROC-ADDR
RETURNING STATUS-CODE.

* ...

```
ENTRY "ERROR-PROC" USING ERROR-PROC-MSG.  
  DISPLAY "IN ERROR PROCEDURE".  
  DISPLAY FUNCTION TRIM(ERROR-PROC-MSG).  
  DISPLAY FUNCTION EXCEPTION-LOCATION.  
  EXIT PROGRAM.  
  STOP RUN.
```

CBL_EQ

CBL_EQ performs a logical EQUAL operation on bits of param-1 and param-2, over the course of a byte-length which is given in param-3.

Usage

```
CALL "CBL_EQ" USING eq-param-1,
                    eq-param-2,
                    BY VALUE eq-length-in-bytes.
```

Parameters

eq-param-1	PIC X(n)
eq-param-2	PIC X(n)
eq-length-in-bytes	PIC 9(n)

Syntax

eq-param-1 may be an alphanumeric literal or data item.
 must be at least 1 byte in length.

eq-param-2 must be a data item.
 must be at least 1 byte in length.

is transformed by the operation, as it will hold the result of the logical EQ operation.

eq-length-in-bytes must be passed "by value".

General Rules

A logical EQ operation is performed on corresponding bits in eq-param-1 and eq-param-2, with the result of the logical EQ operation written to eq-param-2.

The EQ operation uses the following "truth table":

EQ	0	1
0	1	0
1	0	1

Code Sample

```
*
77 EQ-PARAM-1    PIC X VALUE "A".
77 EQ-PARAM-2    PIC X VALUE "B".
77 EQ-LENGTH-IN-BYTES    PIC 9 VALUE 1.
...
      CALL "CBL_EQ" USING EQ-PARAM-1,
                          EQ-PARAM-2,
                          BY VALUE EQ-LENGTH-IN-BYTES.
...

```

CBL_EXIT_PROC

CBL_EXIT_PROC installs or uninstalls an exit procedure, which is run when the application terminates either normally or abnormally. The Exit Routine allows the user to register procedures that will automatically be executed when the program does a normal exit.

Usage

```
call "CBL_EXIT_PROC" using exit-proc-flag,  
                           exit-proc-addr.
```

Parameters

exit-proc-flag	PIC X COMP-X value 0.
exit-proc-addr	USAGE PROCEDURE-POINTER.

Syntax

exit-proc-flag	set to 0 to install exit proc set to 1 to uninstall exit proc
exit-proc-addr	is the address of exit proc

Code Sample

```
*  
...  
01 ERROR-PROC-FLAG      PIC X COMP-X VALUE 0.  
01 ERROR-PROC-ADDR     USAGE PROCEDURE-POINTER.  
*  
...  
SET EXIT-PROC-ADDR TO ENTRY "EXIT-PROC".  
...  
CALL "CBL_EXIT_PROC" USING EXIT-PROC-FLAG,  
                           EXIT-PROC-ADDR.  
STOP RUN.  
...  
ENTRY "EXIT-PROC".  
  DISPLAY "IN EXIT PROCEDURE".  
  EXIT PROGRAM.  
  STOP RUN.  
*
```

CBL_FLUSH_FILE

CBL_FLUSH_FILE causes any file buffers that have not been flushed to disk to be flushed.

Usage

```
CALL "CBL_FLUSH_FILE" USING my-file-handle.
```

Parameters

my-file-handle PIC X(4) COMP-5.

Syntax

my-file-handle is returned after a successful Create, or Open.

General Rules

1. When the function is successful, return-code is set to 0.
2. When the function fails, return-code is set to an ANSI-74 file-status code according to the nature of the error.

Code Sample

```
*  
77 MY-FILE-HANDLE            PIC X(4) COMP-5.  
...  
    INITIALIZE RETURN-CODE.  
  
    CALL "CBL_FLUSH_FILE" USING MY-FILE-HANDLE.  
*
```

CBL_FREE_MEM

CBL_FREE_MEM frees memory allocated by the CBL_ALLOC_MEM routine.

Usage

```
CALL "CBL_FREE_MEM" USING BY VALUE memory-pointer  
                          RETURNING call-status.
```

Parameters

memory-pointer USAGE POINTER.
call-status PIC S9(9).

Syntax

memory-pointer is a data element described as USAGE POINTER, which has been populated with a value by the CBL_ALLOC_MEM routine.

call-status is a return code.

CBL_FREE_DYN_MEM

CBL_FREE_DYN_MEM frees memory allocated by the CBL_ALLOC_DYN_MEM routine.

Usage

```
CALL "CBL_FREE_DYN_MEM" USING BY VALUE memory-pointer  
RETURNING call-status.
```

Parameters

memory-pointer USAGE POINTER.

call-status PIC S9(9).

Syntax

memory-pointer is a data element described as USAGE POINTER, which has been populated with a value by the CBL_ALLOC_DYN_MEM routine.

call-status is a return code.

CBL_GET_CURRENT_DIR

CBL_GET_CURRENT_DIR returns the full path of the current directory.

Usage

```
CALL "CBL_GET_CURRENT_DIR"
      USING BY VALUE      FLAGS
           BY VALUE      LENGTH-OF-DIRNAME
           BY REFERENCE  DIRECTORY-NAME.
```

Parameters

flags	PIC X COMP-X VALUE 0.
length-of-dirname	PIC 9(n)
directory-name	PIC X(n)

Syntax

flags must be set to 0.
 length-of-dirname corresponds to the length of directory-name.
 directory-name must contain enough characters to store the full path name of the current directory. The function call returns the full path into directory-name.

General Rules

1. When the function is successful, return-code is set to 0.
2. When the function fails, return-code is set to an 128.

Code Sample

```
*
77 FLAGS          PIC X COMP-X VALUE 0.
77 LENGTH-OF-DIRNAME PIC 99 VALUE 50.
77 DIRECTORY-NAME  PIC X(50) VALUE SPACES.
...
    CALL "CBL_GET_CURRENT_DIR"
        USING BY VALUE      FLAGS
             BY VALUE      LENGTH-OF-DIRNAME
             BY REFERENCE  DIRECTORY-NAME.
    ...
*
```

CBL_IMP

CBL_IMP performs a logical IMPLIES operation on bits of param-1 and param-2, over the course of a byte-length which is given in param-3.

Usage

```
CALL "CBL_IMP" USING IMP-PARAM-1,  
                    IMP-PARAM-2,  
                    BY VALUE IMP-LENGTH-IN-BYTES.
```

Parameters

imp-param-1 PIC X(n)
imp-param-2 PIC X(n)
imp-length-in-bytes PIC 9(n)

Syntax

imp-param-1 may be an alphanumeric literal or data item.
 must be at least 1 byte in length.

imp-param-2 must be a data item.
 must be at least 1 byte in length.

is transformed by the operation, as it will hold the result of the logical IMP operation.
Imp-length-in-bytes must be passed "by value".

General Rules

A logical IMP operation is performed on corresponding bits in param-1 and param-2, with the result of the logical IMP operation written to param-2.

The IMP operation uses the following "truth table":

IMP	0	1
0	1	1
1	0	1

Code Sample

```
*  
77 IMP-PARAM-1    PIC X VALUE "A".  
77 IMP-PARAM-2    PIC X VALUE "B".  
77 IMP-LENGTH-IN-BYTES    PIC 9 VALUE 1.  
...  
    MOVE "A" TO IMP-PARAM-1.  
    MOVE "B" TO IMP-PARAM-2.  
  
    CALL "CBL_IMP" USING IMP-PARAM-1,  
                          IMP-PARAM-2,  
                          BY VALUE IMP-LENGTH-IN-BYTES.
```

...

CBL_NIMP

CBL_NIMP performs a logical NOT IMPLIES operation on bits of param-1 and param-2, over the course of a byte-length which is given in param-3.

Usage

```
CALL "CBL_NIMP" USING NIMP-PARAM-1,  
                     NIMP-PARAM-2,  
                     BY VALUE NIMP-LENGTH-IN-BYTES.
```

Parameters

nimp-param-1	PIC X(n)
nimp-param-2	PIC X(n)
nimp-length-in-bytes	PIC 9(n)

Syntax

nimp-param-1 may be an alphanumeric literal or data item.
 must be at least 1 byte in length.

nimp-param-2 must be a data item.
 must be at least 1 byte in length.

is transformed by the operation, as it will hold the result of the logical NIMP operation.

nimp-length-in-bytes must be passed "by value".

General Rules

A logical NIMP operation is performed on corresponding bits in param-1 and param-2, with the result of the logical NIMP operation written to param-2.

The NIMP operation uses the following "truth table":

IMP	0	1
0	0	0
1	1	0

Code Sample

```
*  
77 NIMP-PARAM-1    PIC X VALUE "A".  
77 NIMP-PARAM-2    PIC X VALUE "B".  
77 NIMP-LENGTH-IN-BYTES    PIC 9 VALUE 1.  
...  
    CALL "CBL_NIMP" USING NIMP-PARAM-1,  
                          NIMP-PARAM-2,  
                          BY VALUE NIMP-LENGTH-IN-BYTES.  
    ...  
*
```

CBL_NOR

CBL_NOR performs a logical NOR operation on bits of param-1 and param-2, over the course of a byte-length which is given in param-3.

Usage

```
CALL "CBL_NOR" USING nor-param-1,  
                    nor-param-2,  
                    BY VALUE nor-length-in-bytes.
```

Parameters

nor-param-1	PIC X(n)
nor-param-2	PIC X(n)
nor-length-in-bytes	PIC 9(n)

Syntax

nor-param-1 may be an alphanumeric literal or data item.
 must be at least 1 byte in length.

nor-param-2 must be a data item.
 must be at least 1 byte in length.

is transformed by the operation, as it will hold the result of the logical NOR operation.

nor-length-in-bytes must be passed "by value".

General Rules

A logical NOR operation is performed on corresponding bits in param-1 and param-2, with the result of the logical NOR operation written to param-2.

The NOR operation uses the following "truth table":

NOR	0	1
0	1	0
1	0	0

Code Sample

```
*  
77 NOR-PARAM-1    PIC X VALUE "A".  
77 NOR-PARAM-2    PIC X VALUE "B".  
77 NOR-LENGTH-IN-BYTES    PIC 9 VALUE 1.  
...  
    CALL "CBL_NOR" USING NOR-PARAM-1,  
                          NOR-PARAM-2,  
                          BY VALUE NOR-LENGTH-IN-BYTES.  
...  
...
```

CBL_NOT

CBL_NOT performs a logical NOT operation on bits of param-1 and over the course of a byte-length which is given in param-1.

Usage

```
CALL "CBL_NOT" USING not-param-1,  
BY VALUE not-length-in-bytes.
```

Parameters

not-param-1 PIC X(n)
not-length-in-bytes PIC 9(n)

Syntax

not-param-1 may be an alphanumeric literal or data item.
 must be at least 1 byte in length.
is transformed by the operation, as it will hold the result of the logical NOT operation.
not-length-in-bytes must be passed "by value".

General Rules

A logical NOT operation is performed on bits in param-1, with the result of the logical NOT operation written to param-1.

The NOT operation uses the following "truth table":

NOT	
0	1
1	0

Code Sample

```
*  
77 NOT-PARAM-1    PIC X VALUE "A".  
77 NOT-LENGTH-IN-BYTES    PIC 9 VALUE 1.  
...  
    CALL "CBL_NOT" USING NOT-PARAM-1,  
          BY VALUE NOT-LENGTH-IN-BYTES.  
    ...  
*
```

CBL_OC_NANOSLEEP

CBL_OC_NANOSLEEP causes the program to “sleep” in a defined interval that is represented in nanoseconds.

Usage

```
CALL "CBL_OC_NANOSLEEP" USING number-nanoseconds.
```

Parameters

number-nanoseconds Numeric literal or data item

Syntax

number-nanoseconds is the elapsed time in nanoseconds to sleep

Code Sample

```
*  
...  
CALL "CBL_OC_NANOSLEEP" USING 250000000.  
...  
*
```


CBL_OPEN_FILE

CBL_OPEN_FILE opens a sequential file with READ/WRITE permissions described by the parameters passed.

Usage

```
CALL "CBL_OPEN_FILE" USING my-file-name,  
                           my-file-permissions,  
                           my-file-restrictions,  
                           my-device,  
                           my-file-handle.
```

Parameters

my-file-name	PIC X(n).
my-file-permissions	PIC X COMP-X.
my-file-restrictions	PIC X COMP-X.
my-device	PIC X COMP-X.
my-file-handle	PIC X(4) COMP-5.

Syntax

my-file-name is a null-terminated character string.

my-file-permissions describes Read/Write permissions. It must be one of the following:

1	Read-only	2	Write-only
		3	Read-Write
		64	Read-Write for large files (> 4GB)

my-file-restrictions describes Read/Write restrictions. It must be one of the following:

0	Write-only
1	No write
2	No read
3	No read/write restrictions

my-device must be set to 0.

my-file-handle is stored after a successful Create, or Open.

General Rules

When the function is successful, return-code is set to 0.

When the function fails, return-code is set to an ANSI-74 file-status code according to the nature of the error.

My-file-name may include an environment variable notation, for purposes of locating the file, as follows: (See the code sample below, for more details on the usage of CBL_OPEN_FILE).

```
01 my-file-name pic x(40) value "$MYPATH/workfile.txt".  
....
```

In this case, the \$MYPATH notation would cause the value of the “MYPATH” environment variable to be prepended to workfile.txt, for purposes of locating the file.

Code Sample

```
*
77 MY-FILE-NAME           PIC X(11) .
77 MY-FILE-PERMISSIONS   PIC X COMP-X VALUE 3 .
77 MY-FILE-RESTRICTIONS  PIC X COMP-X VALUE 3 .
77 MY-DEVICE             PIC X COMP-X VALUE 0 .
77 MY-FILE-HANDLE        PIC X(4) COMP-5 .
...
    STRING "MYTEXTFILE" DELIMITED BY SIZE,
           X"00", DELIMITED BY SIZE,
           INTO MY-FILE-NAME .

    CALL "CBL_OPEN_FILE"
        USING MY-FILE-NAME,
             MY-FILE-PERMISSIONS,
             MY-FILE-RESTRICTIONS,
             MY-DEVICE,
             MY-FILE-HANDLE .
*
```

CBL_OR

CBL_OR performs a logical OR operation on bits of param-1 and param-2, over the course of a byte-length which is given in param-3.

Usage

```
CALL "CBL_OR" USING or-param-1,  
                  or-param-2,  
                  BY VALUE or-length-in-bytes.
```

Parameters

or-param-1	PIC X(n)
or-param-2	PIC X(n)
or-length-in-bytes	PIC 9(n)

Syntax

or-param-1 may be an alphanumeric literal or data item.
 must be at least 1 byte in length.

or-param-2 must be a data item.
 must be at least 1 byte in length.

is transformed by the operation, as it will hold the result of the logical OR operation.
or-length-in-bytes must be passed "by value".

General Rules

A logical OR operation is performed on corresponding bits in param-1 and param-2, with the result of the logical OR operation written to param-2.

The OR operation uses the following "truth table":

OR	0	1
0	0	1
1	1	1

Code Sample

```
*  
77 OR-PARAM-1    PIC X VALUE "A".  
77 OR-PARAM-2    PIC X VALUE "B".  
77 OR-LENGTH-IN-BYTES    PIC 9 VALUE 1.  
...  
    CALL "CBL_OR" USING OR-PARAM-1,  
                          OR-PARAM-2,  
                          BY VALUE OR-LENGTH-IN-BYTES.  
...           *
```

CBL_READ_FILE

CBL_READ_FILE READs a number of bytes from an offset of a file into a buffer.

Usage

```
CALL "CBL_READ_FILE" USING my-file-handle,  
                           my-file-offset,  
                           my-byte-count,  
                           my-read-flag,  
                           my-read-buffer.
```

Parameters

my-file-handle	PIC X(4) COMP-5.
my-file-offset	PIC X(8) COMP-X.
my-byte-count	PIC X(n) COMP-X.
my-read-flag	PIC X COMP-X.
my-read-buffer	PIC X(n).

Syntax

my-file-handle is required, and obtained by performing a successful CREATE or OPEN.
my-file-offset is the offset to begin the READ operation, beginning the file at offset 0.
my-byte-count is the number of bytes to read from the file.
my-read-flag must be one of the following:
0 Standard read
128 Return the current file size in file-offset
my-read-buffer is the data that is being read from the file.

General Rules

1. When the function is successful, return-code is set to 0.
2. When the function fails, return-code is set to an ANSI-74 file-status code according to the nature of the error.

Code Sample

```
*  
77 MY-FILE-HANDLE          PIC X(4) COMP-5.  
77 MY-FILE-OFFSET         PIC X(8) COMP-X VALUE 0.  
77 MY-BYTE-COUNT          PIC X(4) COMP-X VALUE 11.  
77 MY-READ-FLAG           PIC X COMP-X VALUE 0.  
77 MY-READ-BUFFER         PIC X(11) VALUE SPACES.  
...  
    INITIALIZE MY-READ-BUFFER.  
    CALL "CBL_READ_FILE"  
        USING MY-FILE-HANDLE,  
              MY-FILE-OFFSET,  
              MY-BYTE-COUNT,  
              MY-READ-FLAG,  
              MY-READ-BUFFER.
```


CBL_RENAME_FILE

CBL_RENAME_FILE renames the named source file.

Usage

```
CALL "CBL_RENAME_FILE" USING source-filename,  
                             renamed-filename.
```

Parameters

source-filename PIC X(n).
renamed-filename PIC X(n).

Syntax

source-filename is the name of the original source file.
renamed-filename is the name to which it is renamed.

General Rules

1. When the function is successful, return-code is set to 0.
2. When the function fails, return-code is set to 128.

Code Sample

```
*  
77 SOURCE-FILENAME        PIC X(20).  
77 RENAMED-FILENAME      PIC X(20).  
...  
    MOVE "HELLO.CBL" TO SOURCE-FILENAME.  
    MOVE "WORLD.CBL" TO RENAMED-FILENAME.  
    CALL "CBL_RENAME_FILE"  
        USING SOURCE-FILENAME,  
              RENAMED-FILENAME.  
    ...  
*
```

CBL_TOLOWER

CBL_TOLOWER translates a text string into lower-case.

Usage

```
CALL "CBL_TOLOWER" USING source-data,  
                        VALUE source-length.
```

Parameters

source-data PIC X(n)

source-length USAGE UNSIGNED-INT, or a numeric literal

Syntax

source-data is the data to translate to upper-case.

is transformed by the CBL_TOLOWER operation, with all characters being translated to lower-case.

source-length is the number of characters to translate.

General Rules

1. Return-code is not updated following the operation.

Code Sample

```
*  
77 SOURCE-DATA PIC X(10).  
77 SOURCE-LENGTH USAGE UNSIGNED-INT.  
...  
    MOVE "ABCDEFGHIJ" TO SOURCE-DATA.  
    MOVE 10 TO SOURCE-LENGTH.  
    CALL "CBL_TOLOWER"  
        USING SOURCE-DATA, VALUE SOURCE-LENGTH.  
    ...  
*
```

CBL_TOUPPER

CBL_TOUPPER translates a text string into upper-case.

Usage

```
CALL "CBL_TOUPPER" USING source-data, VALUE source-length
```

Parameters

source-data PIC X(n)

source-length USAGE UNSIGNED-INT, or a numeric literal

Syntax

source-data is the data to translate to upper-case.

is transformed by the CBL_TOUPPER operation, with all characters being translated to upper-case.

source-length is the number of characters to translate.

General Rules

1. Return-code is not updated following the operation.

Code Sample

```
*
 77 SOURCE-DATA PIC X(10).
 77 SOURCE-LENGTH USAGE UNSIGNED-INT.
 ...
 MOVE "ABCDEFGHIJ" TO SOURCE-DATA.
 MOVE 10 TO SOURCE-LENGTH.
 CALL "CBL_TOUPPER"
     USING SOURCE-DATA,
          VALUE SOURCE-LENGTH.
 ...
*
```


CBL_WRITE_FILE

CBL_WRITE_FILE is WRITES a number of bytes to an offset of a file from a buffer.

Usage:

```
CALL "CBL_WRITE_FILE" USING my-file-handle,  
                             my-file-offset,  
                             my-byte-count,  
                             my-write-flag,  
                             my-write-buffer.
```

Parameters

my-file-handle	PIC X(4) COMP-5.
my-file-offset	PIC X(8) COMP-X.
my-byte-count	PIC X(n) COMP-X.
my-write-flag	PIC X COMP-X.
my-write-buffer	PIC X(n).

Syntax

my-file-handle	Required, and can only be obtained by performing a successful CREATE or OPEN.
my-file-offset	The offset to begin the WRITE operation on;- the beginning of the file is offset 0.
my-byte-count	The number of bytes to write to the file.
my-write-flag	Must be one of the following:
0	Standard write
128	Return the current file size in file-offset
my-write-buffer	The data that is being written to the file.

General Rules

1. When the function is successful, return-code is set to 0.
2. When the function fails, return-code is set to an ANSI-74 file-status code according to the nature of the error.

Code Sample

```
*  
77 MY-FILE-HANDLE          PIC X(4) COMP-5.  
77 MY-FILE-OFFSET          PIC X(8) COMP-X VALUE 0.  
77 MY-BYTE-COUNT           PIC X(4) COMP-X VALUE 11.  
77 MY-WRITE-FLAG           PIC X COMP-X VALUE 0.  
77 MY-WRITE-BUFFER         PIC X(11) VALUE SPACES.  
...  
    MOVE "HELLO WORLD" TO MY-WRITE-BUFFER.  
    CALL "CBL_WRITE_FILE"  
        USING MY-FILE-HANDLE,  
              MY-FILE-OFFSET,  
              MY-BYTE-COUNT,
```

```
MY-WRITE-FLAG,  
MY-WRITE-BUFFER.
```

CBL_XOR

CBL_XOR performs a logical XOR operation on bits of param-1 and param-2, over the course of a byte-length which is given in param-3.

Usage

```
CALL "CBL_XOR" USING xor-param-1,
                    xor-param-2,
                    BY VALUE xor-length-in-bytes.
```

Parameters

xor-param-1	PIC X(n)
xor-param-2	PIC X(n)
xor-length-in-bytes	PIC 9(n)

Syntax

xor-param-1 may be an alphanumeric literal or data item.
 must be at least 1 byte in length.

xor-param-2 must be a data item.
 must be at least 1 byte in length.

is transformed by the operation, as it will hold the result of the logical XOR operation.
xor-length-in-bytes must be passed "by value".

General Rules

A logical XOR operation is performed on corresponding bits in param-1 and param-2, with the result of the logical XOR operation written to param-2.

The XOR operation uses the following "truth table":

XOR	0	1
0	0	1
1	1	0

Code Sample

```
*
77 XOR-PARAM-1    PIC X VALUE "A".
77 XOR-PARAM-2    PIC X VALUE "B".
77 XOR-LENGTH-IN-BYTES    PIC 9 VALUE 1.
...
      CALL "CBL_XOR" USING XOR-PARAM-1,
                          XOR-PARAM-2,
                          BY VALUE XOR-LENGTH-IN-BYTES.
...

```

SYSTEM

SYSTEM provides a means of executing command-line commands from within the COBOL program. The program is paused until the command is complete.

Usage

```
CALL "SYSTEM" USING command,  
                GIVING exit-status.
```

Parameters

command	PIC X(n)
exit-status	Any numeric data item

Syntax

command	is the command-line command that is executed.
exit-status	returns the called program's exit status, or -1 if the command failed.

General Rules

1. When the function is successful, return-code is set to 0.
2. When the function fails, return-code is set to 128.

Code Sample

```
*  
...  
CALL "SYSTEM" USING "NOTEPAD.EXE".  
...  
*
```

X"91" function 11

X"91" function 11 sets the Special-Names programmable switches.

Usage

```
call x"91" using fn-status,  
                x91-function,  
                group-item.
```

Parameters

fn-status	PIC X COMP-X
x91-function	PIC X COMP-X
group-item	group-item with
switch-flags	PIC X COMP-X occurs 8
debug-flag	PIC X COMP-X

Syntax

fn-status	set to 0 if the function is successful, else 1.
X91-function	set to 11.
group-item	group-item with
switch-flags	values (0 or 1) for each of the 8 switches
debug-flag	set to 0.

Code Sample

```
*  
77 FN-STATUS PIC X COMP-X.  
77 X91-FUNCTION PIC X COMP-X VALUE 11.  
01 GROUP-ITEM.  
05 SWITCH-FLAGS PIC X COMP-X OCCURS 8.  
05 DEBUG-FLAG PIC X COMP-X.  
...  
MOVE 1 TO SWITCH-FLAGS(1).  
CALL X"91" USING FN-STATUS, X91-FUNCTION, GROUP-ITEM.  
...  
*
```

X"91" function 12

X"91" function 12 reads the Special-Names programmable switches.

Usage

```
call x"91" using fn-status,  
                x91-function,  
                group-item.
```

Parameters

fn-status	PIC X COMP-X
x91-function	PIC X COMP-X
group-item	group-item with
switch-flags	PIC X COMP-X occurs 8
debug-flag	PIC X COMP-X

Syntax

fn-status	set to 0 if the function is successful, else 1.
X91-function	set to 12.
group-item	group-item to receive:
switch-flags	value of each of the switches set
debug-flag	set to 0.

Code Sample

```
*  
...  
SPECIAL-NAMES.  
    SWITCH 1 IS SWITCH-1.  
...  
77 FN-STATUS    PIC X COMP-X.  
77 X91-FUNCTION PIC X COMP-X VALUE 12.  
01 GROUP-ITEM.  
    05 SWITCH-FLAGS PIC X COMP-X OCCURS 8.  
    05 DEBUG-FLAG PIC X COMP-X.  
...  
    SET SWITCH-1 TO ON.  
    CALL X"91" USING FN-STATUS, X91-FUNCTION, GROUP-ITEM.  
    ...  
*
```

X"91" function 15

X"91" function 15 checks to see if a program exists.

Usage

*call x"91" using fn-status,
x91-function,
param.*

Parameters

fn-status	PIC X COMP-X
x91-function	PIC X COMP-X
param	Group Item containing
length-of-progname	PIC X COMP-X
program-name	PIC X(N).

Syntax

fn-status set to program-name if successful, 0 if program not found.
 X91-function set to 15.
 param group item containing length of program-name, and program-name.

Code Sample

```
*
...
77 FN-STATUS          PIC X COMP-X.
77 X91-FUNCTION       PIC X COMP-X VALUE 15.
01 PARAM.
   05 LWNGTH-OF-PROGNAME  PIC X COMP-X.
   05 PROGRAM           PIC X(6) .
...
PROCEDURE DIVISION.
...
MOVE 6 TO LENGTH-OF-PROGNAME.
MOVE "TESTIT" TO PROGRAM-NAME.
CALL X"91" USING FN-STATUS, X91-FUNCTION, PARAM.
...
*
```

X"91" function 16

X"91" function 16 returns the number of parameters passed through linkage to a sub-program.

Usage

```
call x"91" using fn-status,  
                x91-function,  
                num-params.
```

Parameters

fn-status	PIC X COMP-X
x91-function	PIC X COMP-X
num-params	PIC X COMP-X

Syntax

fn-status set to 0 if the function is successful, else 1.
X91-function set to 16.
num-params receives the number of parameters on the Procedure Division USING statement.

Code Sample

```
*  
...  
77 FN-STATUS            PIC X COMP-X.  
77 X91-FUNCTION PIC X COMP-X VALUE 16.  
77 NUM-PARAMS        PIC X COMP-X.  
...  
LINKAGE SECTION.  
01 LK-NAME            PIC X(25) .  
01 LK-ADDR            PIC X(25) .  
01 LK-CUSTOMERID       PIC X(5) .  
PROCEDURE DIVISION USING LK-NAME, LK-ADDR, LK-CUSTOMERID.  
...  
    CALL X"91" USING FN-STATUS, X91-FUNCTION, NUM-PARAMS.  
...  
*
```


X"F4"

X"F4" builds a byte (or bytes), bit-by-bit, using the least significant bits of bytes that have been supplied in a table.

Usage

```
CALL X"F4" using  new-byte,  
                 byte-array.
```

Parameters

new-byte PIC X COMP-X
byte-array A group-item table with:
tbl-byte USAGE BINARY-CHAR OCCURS 8

Syntax

new-byte contains the new byte.
byte-array array of eight bytes.

General Rules

The least significant bit of each byte in byte-array will form a bit in new-byte. To manufacture the letter "A", which consists of the bits 0100 0001 for example, the least significant bit of tblbyte(1) must be 0, the least significant bit of tbyte(2) must be 1, and in so forth. In this fashion, the byte is constructed from left to right, adding 0, then 1, then 0, then 0, then 0, then 0, then 0, then 1 to build 0100 0001, or the character "A".

Code Sample

```
*  
01 NEW-BYTE PIC X COMP-X.  
01 BIT-TABLE.  
   05 TBL-BYTE OCCURS 8 TIMES USAGE BINARY-CHAR.  
   ...  
       MOVE 0 TO TBL-BYTE (1) .  
       MOVE 1 TO TBL-BYTE (2) .  
       MOVE 0 TO TBL-BYTE (3) .  
       MOVE 0 TO TBL-BYTE (4) .  
       MOVE 0 TO TBL-BYTE (5) .  
       MOVE 0 TO TBL-BYTE (6) .  
       MOVE 0 TO TBL-BYTE (7) .  
       MOVE 1 TO TBL-BYTE (8) .  
       CALL X"F4" USING NEW-BYTE, BIT-TABLE.  
       ...
```

X"F5"

X"F5" unpacks a byte (or bytes), bit-by-bit, into an array of bytes, mapping the first bit of the byte into the least significant bit of the first item in the array, the second bit into the least significant bit of the second byte in the array, and so forth...

Usage

```
CALL X"F5" using  source-byte,  
                 byte-array.
```

Parameters

source-byte PIC X COMP-X
byte-array A group-item table with:
tbl-byte USAGE BINARY-CHAR OCCURS 8

Syntax

source-byte contains the new byte.
byte-array array of eight bytes.

General Rules

1. To unpack the letter "A", for example, which consists of the bits 0100 0001 the most significant bit (0) is written to the least significant bit of tblbyte(1), then moving from left to right, the next most significant bit (1) is written to the least significant bit of tblbyte(2) and so forth. In this fashion, the byte is unpacked, from left to right, adding 0, then 1, then 0, then 0, then 0, then 0, then 0, then 1 to transform the table such that the least significant bits of each of its entries hold the bits 0, 1, 0, 0, 0, 0, 0, 1.

Code Sample

```
*  
...  
01 SRC-BYTE      PIC X VALUE "A".  
01 SOURCE-BYTE  REDEFINES SRXC-BYTE PIC X COMP-X.  
01 BYTE-ARRAY.  
   05 TBL-BYTE  OCCURS 8 TIMES USAGE BINARY-CHAR.  
...  
   CALL X"F5" USING SOURCE-BYTE, BYTE-ARRAY.  
...  
*
```

The Runtime Data Structure (rtd)

From C use Runtime Data Structure (rtd)

```
#include <libcob.h>
```

```
...
```

```
COB_RTD = cob_get_rtd();
```

```
Define cit_runtime_t * const rtd
```

Representing Runtime Flags in the Runtime Data Structure

COB_NO_SIGNAL=1	rtd->cob_disable_signal_handler = 1;
COB_ERROR_FILE=<Filename>	rtd->cob_err_file = stderr;
COB_WARNING=N	rtd->warning_disable = 1;
COB_LOAD_CASE ->doc	UPPER : rtd->name_convert = 1; LOWER: rtd->name_convert = 2; rtd->load_match_exact_case = 1/0; rtd->load_match_upper_case = 1/0; rtd->load_match_lower_case = 1/0;
COB_CALL_CASE ->doc	rtd->load_match_exact_case = 1/0; rtd->load_match_upper_case = 1/0; rtd->load_match_lower_case = 1/0;
COB_LOAD_PRIORITY	rtd->call_flag = COB_LOAD_PRIORITY;
COB_FULL_CANCEL	rtd->call_flag = COB_FULL_CANCEL;
COB_PRE_LOAD=<lib list>	cob_load_shared_lib (rtd, s);
COB_EXTFH=<Name>	rtd->default_extfh_entry =<Name> extfh:<name>
COB_EXTFH_INDEXED=<Name> isam-extfh:name	rtd->default_extfh_indexed_entry =...
COB_EXTFH_FLAT=<Name> flat-extfh:name	rtd->default_extfh_flat_entry = ...
COB_EXTFH_LIB=<Library Name> isam-extfh-lib:lib_name flat-extfh-lib:lib_name	cob_load_shared_lib (rtd,s);
COB_SYNC=Y	rtd->cob_do_sync = 1;

COB_SORT_MEMORY=<Bytes>	rtd->cob_sort_memory = n;
COB_LS_NULLS=Y/N	YES : rtd->cob_ls_nulls = 1; NO: rtd->cob_ls_nulls = -1; !!! THIS overwrite the line-seq-mf:yes/no
COB_LS_DOS=Y/N	YES : rtd->cob_ls_dos = 1; NO: rtd->cob_ls_dos = -1; !!! THIS overwrite the line-seq-dos:yes/no

The COBOL-IT Region Interface

Overview

For the purposes of this documentation, there are 4 sample programs, and a compile script which are designed to highlight key information about the COBOL Region Interface.

The upper limit on the number of regions that can be processed concurrently by CICS is 100 concurrent regions.

When implementing the COBOL-IT Region Interface, programs must be compiled with the `-fthread-safe` compiler flag. Compiling with `-fthread-safe` provides region isolation, which is implemented in the same way as thread isolation.

The sample programs are:

<code>comp.sh</code>	The compile script
<code>testregion.c</code>	The main program
<code>proga.cob</code>	COBOL program CALL'ed by <code>testregion.c</code>
<code>progb.cob</code>	COBOL program CALL'ed by <code>proga</code> .
<code>progfail.cob</code>	

The compile script used, and the sample programs are all included at the end of this chapter.

The compile script-

comp.sh

```
cobc -fthread-safe -m proga.cob
cobc -fthread-safe -m progb.cob
cobc -fthread-safe -m progfail.cob
cobc -x testregion.c
```

Please note that all of the programs used in the demo are compiled with the `-fthread-safe` compiler flag. It is a requirement that all programs that are involved with regions **MUST** be compiled the `-fthread-safe` compiler option. For more information on the `-fthread-safe` compiler flag, please

reference “Guidelines for thread-safe programs” in “Getting Started with COBOL-IT”.

To enable debugging of the “C” code, the compiler flags `-G -save-temps` should be added to each of the compile commands in `comp.sh`.

After compiling the programs, run the `testregion` executable:

```
>testregion
```

`Testregion` calls a COBOL program called “`proga`”, which calls a subprogram “`progb`”.

The demo shows :

- How to initialize a region, and call a program in that region
- How to cancel a program
- How to use `cob_set_exit_rtd_proc` with `setjmp/longjmp`

The REGION API

All API functions get as a parameter the runtime data from region 0 (the main program region). The runtime data from region 0 is retrieved with a call to `cob_get_rtd()` at program startup. Note that “`rtd`” is an abbreviation of “runtime data”.

The API functions:

```
unsigned int cob_enterprise_get_current_region (cit_runtime_t * r0rtd);
```

Returns the current region number.

Input :

```
cit_runtime_t * r0rtd           Region 0 (main region ) rtd
```

```
cit_runtime_t * cob_enterprise_set_current_region (cit_runtime_t * r0rtd,  
unsigned int region);
```

Returns an `rtd` of the region number 'region'. The region is initialized if needed.

Input :

```
cit_runtime_t * r0rtd           Region 0 (main region ) rtd  
unsigned int region             Requested region, number between 0 and  
                                15
```

```
void cob_enterprise_cancel_region(COB_RTD, unsigned int region, int full_cancel)  
;
```

Cancels all programs started in a region

Input :

```
cit_runtime_t * r0rtd          Region 0 (main region ) rtd
unsigned int region            Region number to cancel between 0 and
                               15
int full_cancel                If not 0, the programs are unloaded
                               from memory and memory allocated is
                               freed.
```

Examples for performing Logical/Full Cancels in Region 1:

<code>cob_enterprise_cancel_region(rtd, 1,0)</code>	The last parameter must be set to 0 to perform a "Logical Cancel".
<code>cob_enterprise_cancel_region(rtd, 1,1)</code>	The last parameter must be set to 1 to perform a "Full Cancel".

How to call a program (and all sub programs) in a region

When a region is set to be the current active region, it returns an rtd that must be used as the first parameter for all calls to the runtime library.

To call a program in a region :

- Set the current active region
- Use the returned rtd to resolve and call the program

For example:

```
union {
int (*func) ();
void *func_void;
} unifunc;
cit_runtime_t * rtd_region;

rtd_region = cob_enterprise_set_current_region(R0_rtd, region);
unifunc.func_void = cob_resolve (rtd_region, "proga");
if ( unifunc.func_void == NULL ) {
cob_call_error (rtd_region);
}
unifunc.func (NULL,NULL,NULL);
```

WARNING: As a general rule, you should never call a runtime function giving as a first parameter an rtd that does not reference runtime data of the current active region.

The sole exception to this rule would be a call to `cob_enterprise_cancel_region`.

SetJump/LongJump

In the Region 0 the C program may call 'cob_set_exit_rtd_proc' to set up a C function that will be called if for any reason the runtime aborts. Note that this may also be called if the COBOL program executes a STOP RUN.

Used with setjmp/longjmp this provides a way to recover from an error.

The sample program below shows how to recover from an error which is caused by CALL'ing a subprogram that does not exist:

```
void cob_set_exit_rtd_proc (cit_runtime_t * r0rtd, cob_rtd_exit_proc cob_exit_proc);
```

Input :

cit_runtime_t * r0rtd	Region 0 (main region) rtd
cob_rtd_exit_proc	Followed by the pointer to the exit proc
cob_exit_proc	A pointer to a C function getting as its first parameter the rtd of the current region and as its second parameter an int representing the exit status.

The Sample Programs

comp.sh

```
cobc -fthread-safe -m proga.cob  
cobc -fthread-safe -m progb.cob  
cobc -fthread-safe -m progfail.cob  
cobc -fthread-safe -x testregion.c
```

testregion.c

```
#include <stdio.h>  
#include <assert.h>  
#include "libcob.h"  
  
jmp_buf jump_buffer;  
  
void cob_exit_proc (COB_RTD, int status)  
{  
    /* terminate the current region */  
    cob_terminate_exec(rtd );  
  
    /* then go back to call*/  
    longjmp(jump_buffer, 1);  
}
```

```
void call_prog (COB_RTD, int region, char *prog_name, char *
mess) {
    union {
        int (*func)();
        void *func_void;
    } unifunc;
    cit_runtime_t * rtd_region;
    rtd_region = cob_enterprise_set_current_region(rtd, region);
    unifunc.func_void = cob_resolve (rtd_region, prog_name);
    if ( unifunc.func_void == NULL ) {
        cob_call_error (rtd_region);
    }
    printf("\nregion %s\n", mess);
    unifunc.func (NULL,NULL,NULL);
}
```

```
int main (int argc, char **argv)
{
    COB_RTD = cob_get_rtd();
    cit_runtime_t * rtd_region1;
    cit_runtime_t * rtd_region2;

    cob_init (rtd, 0, NULL);

    /* set up terminate Proc in Region 0*/
    cob_set_exit_rtd_proc(rtd,cob_exit_proc);

    /* test Long jump*/
    if (setjmp(jump_buffer) == 0) {

        call_prog(rtd, 0, "proga", "main");
        call_prog(rtd, 0, "proga", "main");

        call_prog(rtd, 1, "proga", "r1");
        call_prog(rtd, 1, "proga", "r1");
        call_prog(rtd, 2, "proga", "r2");
        printf("\n cancel region 1\n");
        cob_enterprise_cancel_region(rtd, 1, 0);
        call_prog(rtd, 0, "proga", "main");
        call_prog(rtd, 1, "proga", "r1");
        call_prog(rtd, 2, "proga", "r2");
        cob_enterprise_cancel_region(rtd, 2, 0);
        call_prog(rtd, 2, "proga", "r2");

        call_prog(rtd, 2, "progfail", "r1");
        printf("We should never display this message");
    }
}
```



```
} else {  
    printf("After long jump");  
    call_prog(rtd, 1, "proga", "r1");  
}  
  
}
```

proga.cob

```
IDENTIFICATION    DIVISION.  
PROGRAM-ID.      proga.  
ENVIRONMENT      DIVISION.  
CONFIGURATION SECTION.  
WORKING-STORAGE SECTION.  
01  fa PIC X(30) VALUE "INITITAL VALUE".  
  
PROCEDURE        DIVISION.  
DISPLAY "PROGRAM A " fa .  
MOVE "New data" TO fa.  
CALL "progb" .  
CALL "progb" .  
DISPLAY "PROGRAM A CANCEL ProgB" .  
CANCEL "progb".  
  
CALL "progb" .
```

progb.cob

```
IDENTIFICATION    DIVISION.  
PROGRAM-ID.      progb.  
ENVIRONMENT      DIVISION.  
CONFIGURATION SECTION.  
WORKING-STORAGE SECTION.  
01  fa PIC X(30) VALUE "INITITAL VALUE".  
  
PROCEDURE        DIVISION.  
DISPLAY "PROGRAM B " fa .  
MOVE "New data" TO fa.
```

progfail.cob

```
IDENTIFICATION    DIVISION.  
PROGRAM-ID.      progfail.
```

```

ENVIRONMENT          DIVISION.
CONFIGURATION SECTION.
WORKING-STORAGE SECTION.
01    fa PIC X(30) VALUE "INITIAL VALUE".
    
```

```

PROCEDURE           DIVISION.
DISPLAY "PROGRAM Fail " fa .
MOVE "New data" TO fa.
CALL "FAIL" .
    
```

What cobc does

To get a better feel for exactly what cobc does, add the `-v` and `-save-temps` compiler flags to your command line. Follow the output, and step through the separate processes of :

- Precompiling the COBOL program
- Translating the COBOL program to “C”
- Compiling the “C” program
- Linking the executable object (shared object/DLL or native executable)
- Embedding the manifest file in the executable object

After examining the two cases of created a DLL, and a native executable in Windows, we will examine a third case, in which cobc is used to compile a “C” program, and explain what is taking place “under the hood”.

For the purpose of this exercise, we have created a table, with the output in the column on the left, and commentary in the column on the right.

Creating a shared object/dll (Windows)

1	<code>C:\lab4>cobc -v -save-temps hello.cbl</code>	Compiling hello.cbl with <code>-v</code> and <code>-save-temps</code> . <code>-save-temps</code> causes a subfolder called “c” to be created, in which intermediate files are stored, in the step of translating COBOL to “C”. The absence of any other compiler options will assume the default of <code>-m</code> , and create a shared object/DLL as the executable object.
2	<code>cobc:0: Temps files to 'C:\lab4\c'</code>	When translating COBOL to “C”, intermediate files will be stored in a “c” subfolder.
3	<code>cobc:0: preprocessing C:\lab4\hello.cbl into C:\lab4\c\hello.i</code>	<code>cobc</code> begins by preprocessing the COBOL program, expanding copy files, setting constant values that have been defined in level 78s, applying conditional compile directives, and so forth. The preprocessed COBOL program is stored in a temporary

		COBOL program file with a .i extension.
4	cobc:0: Processing hello	Informational message marking the transition from preprocessing to processing.
5	cobc:0: translating C:\lab4\c\hello.i into C:\lab4\c\hello.c	cobc translates the preprocessed COBOL program into “C”. Note that cobc can be directed to stop here, using the command >cobc -C hello.cbl
6	cobc:0: cl /c /I C:\COBOL\CobolIT64\include /DCOB_HAS_THREAD /W0 /nologo /GF /MD /Fo"hello.obj" "C:\lab4\c\hello.c"	cobc compiles the “C” program to an .obj file in the current directory. These are the default flags passed to the “C” compiler, and can be changed by changing the compiler environment variable COB_CFLAGS. Note- On Windows platforms, the default setting for COB_CFLAGS is: /I C:\Cobol\CobolIT\include /DCOB_HAS_THREAD /W0 /nologo /GF /MD. These are the flags added after “cl /c” and before /Fo “hello.obj” [c file].
7	hello.c	Informational message.
8	cobc:0: Exit code = 0	Informational message. The operation was successful.
9	cobc:0: link /DLL /MANIFEST /out:"hello.dll" /nologo "hello.obj" C:\COBOL\CobolIT64\lib\libcobit_dll.lib /DEFAULTLIB:MSVCRT.LIB	Links hello.obj with libcobit_dll.lib. Creates output files hello.lib, hello.exp, hello.dll.manifest, hello.dll.
10	Creating library hello.lib and object hello.exp	Informational message
11	cobc:0: Exit code = 0	Informational message. The operation was successful.
12	cobc:0: mt /nologo /manifest "hello.dll.manifest" "/outputresource:hello.dll;#2"	Embeds the manifest file in the compiled object.
13	cobc:0: Exit code = 0	Informational message. The operation was successful.

Creating an executable (.exe) (Windows)

1	C:\lab4>cobc -v -save-temps -x hello.cbl	Compiling hello.cbl with -v -save-temps -x. -save-temps causes a subfolder called “c” to be created, in which intermediate files are stored, in the step of translating COBOL to “C”. It also ensures that intermediate files in the working directory will be saved. A native executable will be created by the compiler.
2	cobc:0: Temps files to 'C:\lab4\c'	When translating COBOL to “C”, intermediate files will be stored in a “c”

		subfolder.
3	cobc:0: preprocessing C:\lab4\hello.cbl into C:\lab4\c\hello.i	cobc begins by preprocessing the COBOL program, expanding copy files, setting constant values that have been defined in level 78s, applying conditional compile directives, and so forth. The preprocessed COBOL program is stored in a temporary COBOL program file with a .i extension.
4	cobc:0: Processing hello	Informational message marking the transition from preprocessing to processing.
5	cobc:0: translating C:\lab4\c\hello.i into C:\lab4\c\hello.c	cobc translates the preprocessed COBOL program into “C”. Note that cobc can be directed to stop here, using the command >cobc -C hello.cbl
6	cobc:0: cl /c /I C:\COBOL\CobolIT64\include /DCOB_HAS_THREAD /W0 /nologo /GF /MD /Fo"hello.obj" "C:\lab4\c\hello.c"	cobc compiles the “C” program to an .obj file in the current directory. These are the default flags passed to the “C” compiler, and can be changed by changing the compiler environment variable COB_CFLAGS. Note- On Windows platforms, the default setting for COB_CFLAGS is: /I C:\Cobol\CobolIT\include /DCOB_HAS_THREAD /W0 /nologo /GF /MD. These are the flags added after “cl /c” and before /Fo “hello.obj” [c file].
7	hello.c	Informational message
8	cobc:0: Exit code = 0	Informational message. The operation was successful.
9	cobc:0: Building main entry point	Generates hello_main.c in the current directory.
10	cobc:0: cl /c /I C:\COBOL\CobolIT64\include /DCOB_HAS_THREAD /W0 /nologo /GF /MD /Fo"hello_main.obj" "hello_main.c"	Compiles hello_main.c to hello_main.obj using COB_CFLAGS (or default settings, as in this case).
11	hello_main.c	Informational message.
12	cobc:0: Exit code = 0	Informational message. The operation was successful.
13	cobc:0: link /SUBSYSTEM:CONSOLE /MANIFEST "/out:hello.exe" /nologo "hello.obj" C:\COBOL\CobolIT64\lib\libcobit_dll.lib "hello_main.obj" /DEFAULTLIB:MSVCRT.LIB	Creates hello.lib, hello.exp, hello.exe, hello.exe.manifest.
14	Creating library hello.lib and object hello.exp	Informational message.
15	cobc:0: Exit code = 0	Informational message. The operation was successful.
16	cobc:0: mt /nologo /manifest hello.exe.manifest /outputresource:hello.exe;#2	Embeds the manifest file in the executable hello.exe.
17	cobc:0: Exit code = 0	Informational message. The operation was successful.

Compiling a “C” program with cobc (Windows)

In the Interoperability topics, we see examples of using `cobc` to compile “C” programs. This may seem odd at first, but it is important to remember that `cobc` translates COBOL into “C”, and then invokes the host “C” compiler. When compiling a “C” program, `cobc` skips the COBOL preprocessing, and translation steps, and proceeds directly to the “C” compilation step. Again, let’s examine what that looks like in an example, using the `-v -save-temps` compiler flags.

1	<code>C:\lab4>cobc -v -save-temps say.c</code>	Compiling <code>say.c</code> with <code>-v</code> and <code>-save-temps</code> . <code>-save-temps</code> causes a subfolder called “c” to be created. Note that in this case, no intermediate files will be created in the “c” subdirectory. However, the <code>-save-temps</code> compiler flag will prevent the <code>.obj</code> file, which is referenced in the <code>-v</code> output, from being deleted. The absence of any other compiler options will assume the default of <code>-m</code> , and create a shared object/DLL as the executable object.
2	<code>cobc:0: Temps files to 'C:\lab4\c'</code>	The “c” subfolder is created, but no files are created in it.
4	<code>cobc:0: Processing say</code>	Informational message. Note that the preprocessing and translation steps were skipped, as there was no COBOL to preprocess or translate into “C”.
6	<code>cobc:0: cl /c /I C:\COBOL\CobolIT64\include /DCOB_HAS_THREAD /W0 /nologo /GF /MD /Fo"say.obj" "C:\lab4\say.c"</code>	<code>cobc</code> compiles the “C” program to an <code>.obj</code> file in the current directory. These are the default flags passed to the “C” compiler, and can be changed by changing the compiler environment variable <code>COB_CFLAGS</code> . Note- On Windows platforms, the default setting for <code>COB_CFLAGS</code> is: <code>/I C:\Cobol\CobolIT\include /DCOB_HAS_THREAD /W0 /nologo /GF /MD</code> . These are the flags added after “ <code>cl /c</code> ” and before <code>/Fo “hello.obj” [c file]</code> .
7	<code>say.c</code>	Informational message.
8	<code>cobc:0: Exit code = 0</code>	Informational message. The operation was successful.
9	<code>cobc:0: link /DLL /MANIFEST /out:"say.dll" /nologo "say.obj" C:\COBOL\CobolIT64\lib\libcobit_dll.lib /DEFAULTLIB:MSVCRT.LIB</code>	Links <code>say.obj</code> with <code>libcobit_dll.lib</code> . Creates output files <code>say.lib</code> , <code>say.exp</code> , <code>say.dll.manifest</code> , <code>say.dll</code> .
10	<code>Creating library say.lib and object say.exp</code>	Informational message
11	<code>cobc:0: Exit code = 0</code>	Informational message. The operation was successful.

12	<code>cobc:0: mt /nologo /manifest "say.dll.manifest" "/outputresource:say.dll;#2"</code>	Embeds the manifest file in the compiled object.
13	<code>cobc:0: Exit code = 0</code>	Informational message. The operation was successful.

Compatibility Topics

cobmf

Overview of cobmf

cobmf provides the user with an emulator that recognizes an important subset of the compiler flags supported by the Micro Focus COBOL compiler cob. When using cobmf, it is important to understand the following:

- Cobmf is not a perfect emulator. It has limitations. Some cob flags are supported and some are ignored. See the Table of Compiler Flags below for a guide to supported options.
- In cases where a cob flag is ignored, there may be a work-around using the -CIT "COBOLIT compiler flag" device. We have included some suggestions in the Table of Options below, and also included remarks on using -CIT with cobmf. See "Using the -CIT option with cobmf" for more details.
- There are minor differences in the implementation of the -C directive with cobmf. As an example:
 - In Micro Focus COBOL: `cob -C sign"ebcdic" program1.cbl`
 - Using cobmf:
 - `cobmf -C sign\"ebcdic\" program1.cbl , or`
 - `cobmf -C sign="ebcdic" program1.cbl`See "How cobmf handles the -C "[directive]" compiler flag below for more details.
- In the end, it is very useful to understand what cobmf does. As an emulator, cobmf translates your Micro Focus COBOL command line into a COBOL-IT command line. Some compiler flags will be translated into command-line compiler flags. Some will be translated into settings in a compiler configuration file that is generated along with the command. For details, see "What cobmf does" below for more details.

>cobmf [return]

Note- for a full list of the [options] supported by cobmf, type `cobmf [return]` at the command line.

Usage: cobmf [options] files

Cobmf options

Cobol-IT MF compatible cob
Cobol-IT MF compatible caller usage :
cob [options] files
options are :

-a	ignored
-C directive	Pass syntax-check phase directive to the Compiler
-CC option	Pass option to the C Compiler
-CIT option	Pass option to the CIT cobc Compiler
-Q option	Pass option to the Linker
-c	Compile no further than a linkable object module (.o)
-g	Create debugging information
-I dir	Where are stored copy book
-i	Compile to .int code for unlinked environment
-k cobol-file	Recognize COBOL source file with non-standard filename extension
-L dir	Pass option to system linker, changing search algorithm and maintaining relative ordering
-l lib	Pass option to system linker, maintaining relative ordering
-m symb=newsym	Map text symb onto newsym
-O	Enable optimization
-o filename	Specify output filename
-P	Produce COBOL compilation listing file
-u	Compile to .gnt code for unlinked environment
-V	Report version number
-v	Set verbose mode
-W err-level	Control error level for cob termination
-x	Process to system executable file
-Z	Process to shared library

Using the -CIT option with cobmf

The -CIT command line option is used to pass additional commands to the COBOL-IT compiler when using cobmf command line emulator.

Usage : cobmf -CIT "-v -Os -O" myprog.cob

Note that after including the -CIT option, you just list the COBOL-IT compiler flags, inside quotes, that you would like the COBOL-IT compiler to use in addition to those implied by the usage of the listed Micro Focus compiler flags.

What cobmf does

To examine what cobmf does, an example:

First, compile a simple hello.cbl program using cobmf:

In Windows:

```
> cobmf -L%COBOLITDIR%\lib hello.cbl
```

Cobol-IT MF compatible cob

```
Running : cobc -l citextfh_dll.lib -LC:\Cobol\CobolIT\lib -conf=hello.mfconf -fmf-int hello.cbl  
cob1D5F8608_1.c
```

Creating library hello.lib and object hello.exp

Breaking down the command line, we see:

cobc	The emulator has substituted the COBOL-IT compiler for cobmf
-l citextfh_dll.lib	In Micro Focus emulation mode, the extfh interface is enabled
-LC:\Cobol\CobolIT\lib	In Windows, you must point to the library where the extfh interface library is located. Since COBOL-IT supports the <code>-L <lib></code> compiler flag, no translation was necessary.
-conf=hello.mfconf	hello.mfconf is a compiler configuration file that is generated by the emulator, and which, at a minimum, references the Micro Focus compatibility mode, and may, depending on the compiler options selected, contain other entries. This is a minimal case, so the contents of hello.mfconf are simply: include "mf.conf"
-fmf-int	In Micro Focus emulation mode, an output file with a .int extension is created. The output file with the .int extension is an abbreviated list file.
hello.cbl	The name of the program.

How cobmf handles the `-C "[directive]"` compiler flag

As you use the `-C "[directive]"` compiler flag with cobmf, follow these rules:

- Place a forward slash `/` before quotes
- Directives are translated into equivalent compiler configuration file entries. After running the command, check the contents of the `"conf"` file generated by the emulator, to view how your directive has been translated.

Consider the common case where the `defaultbyte"0"` directive is passed to the compiler on the command-line.

In Micro Focus:

```
cob -C defaultbyte"0" hello.cbl
```

In COBOL-IT, construct the command as follows:

```
C:\COBOL\CobolIT>cobmf -C defaultbyte\"0\" -L%COBOLITDIR%\lib hello.cbl  
Cobol-IT MF compatible cob  
Running : cobc -l citextfh_dll.lib -L C:\Cobol\CobolIT\lib  
-conf=hello.mfconf -  
fmf-int hello.cbl
```



```
cob4AA78944_1.c
```

Creating library hello.lib and object hello.exp

Check the contents of hello.mfconf

Note that the key adjustment that has been made by the emulator is in the compiler configuration file, which now has the following contents:

```
include "mf.conf"
use-defaultbyte:yes
defaultbyte:0
```

COBOL-IT requires that two compiler configuration file entries be set for the DEFAULTBYTE directive, the use-defaultbyte entry, and the defaultbyte entry, which captures the value of the defaultbyte.

COBOPT

The COBOPT environment variable is supported by cobmf. The COBOPT environment variable stores command-line compiler flags, for use with cobmf.

Table of equivalents to cob compiler flags

Cob flag	Function	COBOL-IT Equivalent
-A [as option]	Pass as_option to Assembler	Not supported.
-a	Compile for animation	-g
-C [directive]	Pass syntax-check phase [directive] to the Compiler	Difference. Cobmf provides a limited ability to use. However, best practice is to determine the analogous compiler configuration setting, and update your compiler configuration file. See "Table of Directives"
-CC [cc option]	Pass [cc option] to the "C" Compiler	-Wc [cc option]

-c	Compile no further than linkable object module (.o)	-c
-d [symb]	Dynamically load [symb]	Difference. Use COB_LIBRARY_PATH to locate dynamically loaded module.
-e [symbol]	Set initial entry point to [symbol]	Not required.
-g	Produce debugging information	-g
-I [symbol]	Include [symbol] in executable module	Difference. Statically link [symb]: >cobc -x -o hello [symb1][symb2] -I [dir] directs the compiler to folders containing copy files or include files.
-i	Compile to .int code	Creates a shared object with a .int extension. The shared object is only renamed.
-k [COBOL src file]	Compile [COBOL src file] with non-standard extension	Not required.
-L [dir]	Pass [dir] to system linker, changing search algorithm and maintaining relative ordering	-L [dir]
+L [dir]	Pass [dir] to system linker after all other options, changing search algorithm	Not supported
-l [lib]	Pass [lib] to system linker, maintaining relative ordering	-l [lib]
+l [lib]	Pass option to system linker after all other options	Not supported
-m symb=newsymb	Map symb to newsymb	Not supported

-N [directive]	Pass generate-phase [directive] to compiler	Difference. Cobmf provides a limited ability to use. However, best practice is to determine the analogous compiler configuration setting, and update your compiler configuration file. See "Table of Directives"
-O	Enables optimization	-O
-o [file]	Specifies output [file] name	-o [file]
-P	Produce COBOL listing file with compilation	-t
-p	Instructs "C" compiler to use profiling routines.	-fprofiling
-Q [ld option] or	Pass [ld option] to linker	-Wl [ld option]
-Q, 1 [ld option] or		
-Q, 2 [ld option]		
-t	Creates multi-threading programs	-fthread-safe
-U	Dynamically load unresolved symbols	Not supported
-u	Compile to .gnt code	-fmf-gnt
-V	Returns version number	-V
-v	Sets verbose mode	-v
-W [error level]	Sets [error level] at which compiler aborts	-w compiler flags
-X [symb]	Excludes [symb] from executable output file	Not supported
-x	Produce executable file	x
-x,CC	Produce executable file with C++ support	Not required
-y	Produce self-contained callable shared object.	-m
-z	Produce callable shared object	-m

-z,U	Produce callable shared object and issue error if there are undefined symbols	-m
-z,CC	Produce callable shared object with C++ support.	-m
-Z	Produce shared library	-b
-Z,CC	Produce shared library with C++ support.	-b

Table of equivalents to compiler directives

Micro Focus Directive	What it does	COBOL-IT Equivalent
ACCEPTREFRESH	Causes data areas associated with Screen Section items to be updated from their corresponding Working-Storage Section items before an ACCEPT statement.	-faccept-with-update compiler flag or accept-with-update: yes
ALIGN	Sets memory boundaries on which data items of level-01 or level-77 are aligned.	-falign-8 compiler flag or align-8:yes
ALTER	Permits ALTER statements in your program.	Default behavior
ANIM	Adds extra information to the compiled object, for use by the Debugger.	-g compiler flag
APOST	Causes the Compiler to interpret the figurative constant QUOTE as the single-quote character (').	quote:[any single character] quote: '
ARITHMETIC	Affects how arithmetic expressions are evaluated.	-fcompute-ibm, -fno-compute-ibm compute-ibm:[yes/no]
ASSIGN	Affects how to assign a filename when neither EXTERNAL nor DYNAMIC appear in the SELECT statement.	assign-clause: [cobol2002 / mf / ibm / external]
AUTOLOCK	Sets the default locking to AUTOMATIC rather than EXCLUSIVE for files opened I-O or EXTEND in a multi-user environment.	autolock:yes
BOUND	Enforces runtime-checking of subscript and index values, to make sure they are within the bounds described by the OCCURS clause.	-debug compiler flag
CALLFH	Enables the EXTFH interface by causing all calls for I/O operations to be handled by the Callable File Handler.	-use-extfh compiler flag

CALLSORT	Enables the EXTSM interface, allowing a named program to be called to handle all SORT and MERGE operations.	-use-extsm compiler flag
CASE	Prevents external symbols (such as Program-ID and names of called programs) from conversion to upper case.	Default behavior
CHARSET	Indicates the character set used by the environment.	ebcdic-charset: [yes/no]
CHECK	Causes all run-time checks to be performed in generated code.	-debug compiler flag
CHECKDIV	Indicates behavior for case where a program tries to divide by zero in a statement that has no exception handling.	div-check: yes default is yes
COBFSTATCONV	Names the user-supplied module to be used by the Callable File Handler to convert the file status codes if an I/O error is encountered on a file.	-use-extfh compiler flag
COBIDY	Location of the .idy file	-debugdb=<filename> compiler flag
COMP5-BYTE-ORDER	Sets the byte ordering used for COMP-5 data.	comp5-byteorder: [native/big-endian] Default is comp5-byteorder: native.
COMP-6	Indicates if COMP-6 data is stored in binary or packed decimal format.	signed-comp6-as-comp3:[yes/no]
CONSTANT	Creates a constant for use in the program.	constant "key=value"
COPYEXT	Names the filename extension that the Compiler can use to identify a COPY file that is specified without an extension.	-ext=<extension> compiler flag
COPYLIST	Causes the compiler to list the contents COPY files in a listing. .	Default behavior
COPYPATH	Provides a list of directories for the Compiler to search for copy files.	-I [path] compiler flag, or COB_COPY_DIR, or COBCPY environment variable
DATACOMPRESS	Sets the level of data compression to be done on sequential and indexed files.	datacompress:[integer] or \$SET DATACOMPRESS "x" before the SELECT statement.
DATAMAP	Causes the compiler to produce a datamap in the listing.	"-t " compiler flag. By default, the datamap is included in the listing.
DEFAULTBYTE	Causes uninitialized data items in Working Storage to be initialized to a default byte.	defaultbyte:[any integer] Default is defaultbyte:0 Requires use- defaultbyte:yes compiler configuration flag also be set.

DEFAULTCALLS	Sets the default calling convention.	Default is defaultcall:0 Designates default call-convention used when no CALL-CONVENTION is mentioned in a CALL statement
DIALECT	Causes the compiler and runtime to behave in a manner consistent with the specified dialect.	-conf=<dialect.conf.
EDITOR	Causes the compiler output error messages to a file in a format compatible with a specified editor.	-err <file>
FASTCALL	Optimizes the speed of the CALL operation by not checking whether it is a main program. FASTCALL assumes that the target of a CALL operation is not a main program. EXIT PROGRAM will always cause an exit to the calling program.	-fcall-opt
FCDREG	Causes the compiler to define special registers giving access to File Control Descriptions (FCD) and Key Definition Blocks.	fcdreg:yes -ffcdreg
FILESHARE	Changes the default locking to become AUTOMATIC instead of EXCLUSIVE for files in a multi-user environment. Automatically locks records on a WRITE or REWRITE statement when the program is locking multiple records.	share-all-autolock:yes
FOLD-CALL-NAME	Folds the name of the target of the CALL, CANCEL, ENTRY, and CHAIN statements and the program-name in the PROGRAM-ID paragraph to upper or lower case.	runtime environment variable: COB_CALL_CASE=xul, where x=exact, u=upper,l=lower
FOLD-COPY-NAME	Folds the name of the target of the COPY statement to upper case or lower case.	-ffold-copy-lower compiler flag -ffold-copy-upper compiler flag
FP-ROUNDING	Indicates whether one floating-point receiving item can affect the results of other, nonfloating-point receiving items.	-fround-fp compiler flag round-fp:[yes/no] Default is round-fp:no
HOST- NUMCOMPARE	Affects the comparisons between integer numeric data items of USAGE DISPLAY and alphanumeric literals, figurative constants, or numeric operands.	-mfhostnumcompare compiler flag hostnumcompare: yes
HOST-NUMMOVE	Prevents run-time error 163 (illegal characters in numeric fields) when certain MOVE statements are executed on numeric display data items or numeric operands.	move-picx-to-pic9:raw
IBMCOMP	Indicates word-storage mode.	binary-size:2-4-8

IDENTIFIERLEN	Specifies the portion of an identifier name that the compiler will consider to be significant.	identifer-length: <max-length>
INDD	Allows ACCEPT statements to be read from a specified file.	-sysin=<input file> compiler flag
INITCALL	Names modules to be called immediately before the first statement of a program is executed.	-initcall=<program name> compiler flag initcall: <program-name>
INITPTR	Causes the INITIALIZE statement to initialize DATA-POINTER, OBJECT-REFERENCE, and PROGRAM-POINTER data types.	initialize-pointer{yes/no] Default is yes
LIST	Name of the source listing file.	-t <filename> compiler flag
LISTPATH	Gives the directory location for the list file to be written. The name of the list file is <i>source-name.lst</i> .	-t <directory> compler flag
LISTWIDTH, LW	Limits the width of the list file.	-ftruncate-listing compiler flag truncate-listing: yes
`LNKALIGN	Causes level-01 and level-77 Linkage Section items to always be aligned on a machine-dependent favorable boundary.	-falign-8 compiler flag align-8: yes
MAKESYN	Makes a reserved word synonymous with another reserved word.	-makesyn oldvalue=newvalue compiler flag makesyn: oldvalue=newvalue
MFCOMMENT	Causes an asterisk (*) in column 1 to be treated as a comment line, but does not show the line in the source listing.	-fmfcomment compiler flag mfcomment: yes
MOVE-LEN-CHECK	Causes source and target lengths for alphanumeric MOVE operations to be checked by the compiler.	Default behavior
OBJ	Causes an object file to be generated.	-o compiler flag
ODOSLIDE	Causes the memory location of data items that are located after a variable length table to change as the length of the table changes.	-fodo-slide compiler flag odo-slide: yes
OPT (Intel x86 platforms)	Set the optimization level of the code produced by the compiler on Intel x86 platforms.	-O compiler flag
OPT (Non-Intel x86 platforms)	Set the optimization level of the code produced by the compiler on platforms other than the Intel x86 platforms.	-O compiler flag
OPTIONAL-FILE	Causes the compiler to treat all files opened for I-O or EXTEND as optional.	-foptional-file compiler flag optional-file: yes
OSEXT	Indicates what the file extensions are for COBOL source files.	Not required. COBOL program extension can be set in the Developer Studio, where the compiler needs to build only COBOL programs

OUTDD	Causes the output of DISPLAY and EXHIBIT statements to be written to a specified file.	-sysout=<output file> [,S/L [,Min [,Max]]] compiler flag
PANVALET	Allows ++INCLUDE statements in the source file.	supported by default
PARAMCOUNTCHECK	Allows the program to be called with fewer parameters than are specified in USING clause.	-falloc-unused-linkage alloc-unused-linkage: yes
PCOMP	Allows a user program to be named as a precompiler for COBOL files.	supported by default
PERFORMOPT	Optimizes PERFORM operations.	-freturn-opt compiler flag return-opt: yes
PERFORM-TYPE	Indicates behavior of return jumps from nested PERFORM statements.	perform-osvs: yes equivalent to PERFORM-TYPE (OSVS) PERFORM-TYPE(COB370) PERFORM-TYPE(ENTCOBOL) PEFORM-TYPE(VSC2)
PREPROCESS, P	Causes the source file to be pre-compiled, and the output file to be compiled.	-preprocess compiler flag
PRINT	Names the source listing file.	-t <filename> compiler flag
PROFILE	Includes code in your program for purposes of generating performance statistics when the program is run.	-fprofiling compiler flag
PROGID-COMMENT	Permits comments be included following the PROGRAM-ID header in the Program-Id paragraph.	default behavior
QUAL	Permits qualified data-names and procedure-names in your program.	Qualified data-names are allowed in a program.
QUOTE	Causes the Compiler to interpret the figurative constant QUOTE as the double-quote character (").	quote: "
RECMODE	Describes the default format of files.	-frecmode-f compiler flag -frecmode-osvs compiler flag -frecmode-v compiler flag recmode-f: [yes/no] recmode-osvs: [yes/no] recmode-v: [yes/no]
REENTRANT	Causes many program areas to be dynamically allocated, so that it is safe to have multiple copies of the program running.	-fthread-safe compiler flag thread-safe:yes
REMOVE	Causes reserved words to be removed from the reserved word list, so that they can be used as user-defined words.	not-reserved [any reserved word]

RESEQ	Causes the compiler to produce line numbers in the listing file.	Developer Studio capability
RTNCODE-SIZE	Sets the size of the RETURN-CODE register and its alignment in memory.	rtncode-size <integer> Integer may be 2,4,8
SEQUENTIAL	Indicates the default file type for files defined as ORGANIZATION SEQUENTIAL.	sequential-line:[yes/no]
SETTING, SETTINGS	Causes the compiler to include a listing of directives, and their settings in the source listing.	-dump-config compiler flag default behavior
SHOW-DIR	Causes the compiler show the contents of directives files in the source listing.	-dump-config compiler flag Default behavior
SOURCEFORMAT	Selects format for COBOL source.	-free , -fixed compiler flags
SOURCETABSTOP	Sets the rule for expanding tab characters encountered in the source code into spaces that is used by the compiler.	tab-width: [integer] Default is 8
SPZERO	Causes space characters in USAGE DISPLAY numeric data items to be treated as zeros.	spzero: yes
SSRANGE	Turns on runtime bounds checking for subscripting, reference modifications, and indexes.	-debug compiler flag
STDERR	Causes compiler error messages to be written to STDERR.	-err <file>
STICKY-LINKAGE	Affects how parameters passed to a program are handled during a runtime session in which the program is called multiple times.	sticky-linkage [yes/no/fixed/variable]
TRACE	Turns on runtime tracing through READY TRACE and RESET TRACE statements.	-fsimple-trace -ftrace -ftraceall compiler flags
TRUNC	Determines whether data being stored in USAGE COMP, USAGE BINARY or USAGE COMP-4 data items is truncated to the size given by the item's PICTURE clause or to the maximum size the item can hold.	-fnotrunc compiler flag
UNICODE	Describes the encoding used for Unicode characters.	-futf16-le compiler flag utf16-le: [yes/no] default is utf16-le: no default is big-endian (portable). utf16-le is little-endian (native).
USE	Causes the compiler to read directives from a file.	-conf=xxx.conf compiler flag -std=mf.conf compiler flag COBITOPT environment variable
VERBOSE	Causes the compiler to be verbose.	-v compiler flag
WARNING, WARNINGS	Specifies the lowest severity level of errors to report.	-w compiler flags

The COBOL-IT Debugger Engine (cobcdb)

The COBOL-IT Debugger Engine (cobcdb) has been designed to operate as an engine, working in the background, behind a user interface, such as the interface that is provided by the COBOL-IT Debugging Perspective in the Developer Studio. The COBOL-IT Debugger Engine (cobcdb) runs shared object files that have been created by the COBOL-IT Compiler (cobc) and that have been compiled with the `-g` compiler flag.

Conventions Used

The Debugger Prompt

When you start the COBOL-IT Debugger Engine, the COBOL-IT Debugger Window presents a prompt, into which a Debugger Command can be entered. After entering a Debugger Command, the user will see the results of their command returned, with a subsequent debugger prompt. The default debugger prompt is (cobcdb).

To illustrate:

```
C:\COBOL\COBOLIT\samples>cobcdb hello
CreateProcess "cobcrun -d hello ".
command:11516
(cobcdb)
event:11516
-event-end-stepping-range #0 hello () at C:/COBOL/COBOLIT/samples/hello.cbl!8
(The debugger prompt is here. As an example, enter the version command:)
version
~"COBOL-IT cobcdb 3.6.4\n"
^done
(cobcdb)
(Enter a subsequent command here.)
```

Source Location

Source Location is formatted as:
<Absolute source path name>!<line number>

Example: C:/COBOL/COBOLIT/samples/hello.cbl!21

Variables names

<variable-name> is formatted as:
[@<module-name>.]<section>[<upper-level-fields >.]<field-name>

If no <module-name> is given, current module is searched

If no <section> is given, sections are searched in the following order: file section, working-storage section, linkage-section.

If no <upper-level-field> is given, the first matching field as presented in the original source is returned

Example:

WORKING-STORAGE.WrkA.Wrk_G1.Wrk_G1_F1 or Wrk_G1.Wrk_G1_F1

is equivalent to

@PrgA.WORKING-STORAGE.WrkA.Wrk_G1.Wrk_G1_F1

where declarations are:

working-storage section.

01 WrkA.

03 Wrk_F1 PIC 99.

03 Wrk_F2 PIC 99.

03 Wrk_G1.

05 Wrk_G1_F1 PIC 99.

05 rk_G1_F2 PIC 99.

Usage of the COBOL-IT Debugger:

>cobcdb [options] [program name] [command-line parameters]

command-line parameters

are parameters which would be returned to the program through an ACCEPT from COMMAND-LINE statement.

program name

is the name of the shared object file created by the COBOL-IT Compiler (.dll, .so).

options

are parameters that are passed to the COBOL-IT Debugger. These options include:

-listdid

Causes the COBOL-IT Debugger to list all the running processes by PID, as well as debug-id.

As an example:

```
C:\COBOL\COBOLIT>cobcdb -listdid
did: ----- pid: 11412 module:
did: ----- pid: 11956 module:
did: 12345 pid: 11536 module: hello
did: ----- pid: 3296 module:
did: ----- pid: 3324 module:
```

-n

(Windows only). Causes the COBOL-IT Debugger to start the execution of **program name** in a new cmd.exe window.

-p <did>

Causes the COBOL-IT Debugger to connect to the running process identified by *did*. *did* the debug-id. *did* may be a debug-id, set with the runtime environment variable COB_DEBUG_ID, or it may be the process id (pid) of the currently running process. When using the **-p** *did* parameter, there is no need to specify **program name**, as the program is identified by *did*.

-r host:port

Connects two TCP sockets to *host:port*. Debugger commands, and the results returned are transmitted via these sockets. Used by the Remote System Explorer in the COBOL-IT Developer Studio.

Sockets are identified by the first line sent.

Socket1 is used to exchange Command/Result information. As an example, the COBOL-IT Debugger will READ Commands Socket1, and WRITE the results of the command to that socket.

Socket1 is identified by "*command:pid\n*" where *pid* is the process-ID.

Socket2 is used to write Debugger Events. For more information about Debugger Events, See the Chapter below titled "Debugger Events".

Socket2 is identified by "*event:pid\n*" where *pid* is the process-ID.

-trace

Causes the COBOL-IT Debugger to write tracing information to *cobcdb.out*.

-w <did>

Causes the COBOL-IT Debugger to interrupt the process identified by *did* and set it into a "wait for connect" state. *did* is the debug-id. *Did* may be a debug-id, set with the runtime environment variable *COB_DEBUG_ID*, or it may be the process id (*pid*) of the currently running process. A program that has been set into this state can be debugged with the *-p did* command. When using the *-w did* parameter, there is no need to specify **program name**, as the program is identified by *did*.

-y tty

(UNIX/Linux only). Causes the COBOL-IT Debugger to assign stdout/stdin/stderr to *tty*. When running the COBOL-IT Debugger with *-y tty*, **program name** is required.

Debugger Commands

Debugger Commands include:

break

causes a breakpoint to be set in the location that is indicated. With the addition of the -t flag, breakpoints can be created as temporary breakpoints, which are erased after they have been reached the first time. The break command requires a location parameter. Location parameters for the break command are:

module Sets a breakpoint in a module, as identified by program-id.
label Sets a breakpoint at a paragraph name.
line-nr Sets a breakpoint at a line number.

module, label, and line-nr can be combined, with a ! notation.

break [-t] label

sets a breakpoint at a paragraph name .

Example:

(cobcdb)

break -t para-1

Breakpoint 1 in para-1 at C:/COBOL/COBOLIT/samples/hello.cbl

(cobcdb)

break [-t] module!label

sets a breakpoint at a paragraph name (label) in a module. module is identified by source file name. If no module name is specified, then the current module is used. Since module may not be loaded yet, no validation of module!label is made.

Example:

(cobcdb)

break -t C:/COBOL/COBOLIT/samples/hello.cbl!para-1

Breakpoint 2 in para-1 at C:/COBOL/COBOLIT/samples/hello.cbl

(cobcdb)

break [-t] module!line-nr

sets a breakpoint at a line number in a module. module is identified by source file name. if no module name is specified, then the current module is used. Since

module

may not be loaded yet, no validation of module!line-nr is made.

Example:

```
(cobcdb)
break -t C:/COBOL/COBOLIT/samples/hello.cbl!22
Breakpoint 3 at C:/COBOL/COBOLIT/samples/hello.cbl!22
(cobcdb)
```

break [-t] module!0

sets a breakpoint at the entry-point to module. module is identified by source file name. if no module name is specified, then the current module is used.

Example:

```
break -t c:/COBOL/COBOLit/samples/subpgm.cbl!0
Breakpoint 1 at c:/COBOL/COBOLit/samples/subpgm.cbl ! 0
(cobcdb)
Or
break -t subpgm.cbl!0
Breakpoint 1 at subpgm.cbl ! 0
(cobcdb)
```

bt

causes a CALL/PERFORM stack trace to be generated. The format for the stack trace display is : #<frame-number><module>() at <source-location>

Example:

```
bt
#0 hello () at C:/COBOL/COBOLIT/samples/hello.cbl!21
#1 hello () at C:/COBOL/COBOLIT/samples/hello.cbl!16
(cobcdb)
```

frame-number 0 is the current program position

continue

causes execution of program to be continued until the next breakpoint is encountered, or until the end of the program . An event-continue command is issued. As seen in the example below, this is interrupted when an event-breakpoint-hit event takes place.

Example:

```
break -t para-1
Breakpoint 1 in para-1 at C:/COBOL/COBOLIT/samples/hello.cbl
(cobcdb)
continue
-event-continue
-event-breakpoint-hit (cobcdb)#0 hello () at
C:/COBOL/COBOLIT/samples/hello.cbl!22
```

(cobcdb)

Example :

```
break -t C:/COBOL/COBOLIT/samples/hello.cbl!22
```

```
Breakpoint 1 at C:/COBOL/COBOLIT/samples/hello.cbl ! 22
```

(cobcdb)

```
continue
```

```
-event-continue
```

```
-event-breakpoint-hit (cobcdb)#0 hello () at
```

```
C:/COBOL/COBOLIT/samples/hello.cbl!22
```

contreturn

causes execution to continue to the next PERFORM return, or break on the first breakpoint reached, which ever comes first. An event-contreturn command is issued. This is interrupted when an `-event-end-stepping-range` event takes place.

Example :

```
contreturn
```

```
-event-contreturn
```

```
(cobcdb)-event-end-stepping-range #0 hello () at
```

```
C:/COBOL/COBOLIT/samples/hello.
```

```
cbl!17
```

delete <x>

causes breakpoint number *x* to be deleted.

Example:

(cobcdb)

```
delete 3
```

```
^done
```

(cobcdb)

frame <frame-number>

Prints the source location for the designated frame number. The frame numbers of an application run session are the points at which the application has branched either due to a PERFORM <paragraph> statement or a CALL <subprogram> statement.

Example:

(cobcdb)

```
frame 0
```



```
#0 hello () at C:/COBOL/COBOLIT/samples/hello.cbl!25
(cobcdb)
frame 1
#1 hello () at C:/COBOL/COBOLIT/samples/hello.cbl!17
(cobcdb)
```

info

causes information to be displayed about the <info parameter> that is indicated. The info command requires an <info parameter>.

Info parameters for the info command are:

<i>locals</i>	Displays a dump of the current variables in memory
<i>sources</i>	Displays a list of source files corresponding to loaded modules.
<i>target</i>	Displays the Process ID of the runtime session.

info locals

displays a dump of the values of the fields in the modules currently loaded in memory.

Example :

(cobcdb)

info locals

```
@hello.WORKING-STORAGE
@hello.WORKING-STORAGE.RETURN-CODE = [10]"00000000"
@hello.WORKING-STORAGE.TALLY = [10]"00000000"
@hello.WORKING-STORAGE.SORT-RETURN = [10]"00000000"
@hello.WORKING-STORAGE.NUMBER-OF-CALL-PARAMETERS = [10]"00000000"
@hello.WORKING-STORAGE.message-line = [11]" "
@hello.WORKING-STORAGE.ws-dummy = [1]" "
@hello.WORKING-STORAGE.ctr = [6]"000000"
@hello.WORKING-STORAGE.COB-CRT-STATUS = [4]"0000"
(cobcdb)
```

Info is returned in a structured tree using SECTION as a header in the form :

<variable name> = [<size>]"<string>"

<variable name> is the full qualified variable name

<size> is the number of characters in the string

<string> is the data in human readable form. Strings may contain null characters.

info profiling

Causes a profiling dump to be produced, dumping profiling information at the current point in the program. Profiling information is displayed, and then dumped in the .xls file format.

Example:

```
(cobcdb)
info profiling
```

info sources

displays source files associated with objects loaded in memory

Example:

```
(cobcdb)
info sources
```

Source files

```
C:/COBOL/COBOLIT/samples/hello.cbl
```

```
(cobcdb)
```

info target

displays the pid of the currently running process.

Example:

```
(cobcdb)
info target
Child PID 19012
(cobcdb)
```

kill

kills the current process.

Example:

```
(cobcdb)
```

```
kill
```

```
-event-program-exited (cobcdb)#0 hello () at
```

```
C:/COBOL/COBOLIT/samples/hello.cbl!
```

```
10
```

list

Requires that the source file be accessible. The list debugger command allows you to expand the source you can see inside the console debugger as you execute your debugger commands:

```
(cobcdb)
s
```

```
-event-step
(cobcdb)
-event-end-stepping-range #0 CUSTOMER0 () at /opt/cobol-it-
64/samples/customer0.cbl!99
.0000099>          CALL "C$PID" USING PID.
```

list

```
.0000094.
.0000095.
*****
.0000096.      PROCEDURE DIVISION.
.0000097.
.0000098.      Main Section.
.0000099>          CALL "C$PID" USING PID.
.0000100.      DISPLAY "PID = " PID.
.0000101.      *   CALL "C$DEBUG"
.0000102.      ACCEPT W-SYS-DATE FROM DATE.
.0000103.      MOVE W-SYS-YY          TO CURR-YY.
.0000104.      MOVE W-SYS-MM          TO CURR-MM.
(cobcdb)
```

next

causes execution to pass to the next statement- jumping over a CALL or PERFORM statement before breaking, unless the CALL'ed paragraph or PERFORM statement contains a breakpoint. An event-next command is issued. This is interrupted when an -event-end-stepping-range event takes place. The next command can be abbreviated as "n".

Example :

```
(cobcdb)
```

```
next
```

```
-event-next
```

```
-event-end-stepping-range (cobcdb)#0 hello () at
C:/COBOL/COBOLIT/samples/hello.cbl!17
```

print <variable-name>

displays the value of the variable in human readable format.

Example:

```
print message-line
```

```
$1 = @hello.WORKING-STORAGE.message-line [11]"XXXXXXXXXXXX"
```

```
(cobcdb)
```

The information returned is in the format:

```
$1=@module-name.section-name.variable-name[size]"[string]"
```

Where:

module-name is the program-id of the module being executed.
section-name is the section containing the variable being displayed.
size is the size, in bytes of the variable.
string is the contents of the variable in human-readable format.

printh <variable-name>

displays the value of the variable in hexadecimal format.

Example:

```
printh message-line
```

```
$1 = @hello.WORKING-STORAGE.message-line [22]"58585858585858585858"  
(cobcdb)
```

The information returned is in the format:

```
$1=@module-name.section-name.variable-name[size]"[string]"
```

Where:

module-name is the program-id of the module being executed.
section-name is the section containing the variable being displayed.
size is the size, in bytes of the variable.
string is the contents of the variable in hexadecimal format.

quit

causes an exit from the debugger.

Example:

```
(cobcdb)
```

```
quit
```

```
C:\COBOL\COBOLIT\samples>
```

replace

allows the user to replace the prefix of the pathname to the source file associated with the compiled object.

The replace command may be used with the following parameters:

- | | |
|------------------------------------|---|
| >replace <oldprefix> : <newprefix> | Affects the output of the list command.
Example: replace /dirA : /dirB |
| >replace <no arguments> | Resets the list command. |
| >replace ? | Produces a list of active replacements |

>replace <oldprefix> : <newprefix>

The replace <oldprefix>:<newprefix> command affects the output of the >list command. Other commands such as >info sources or >break still use the original pathname as it is stored in the binary code of the program.

Examples: >replace /dirA:/dirB

In this example, /dirA and /dirB are prefixes of a pathname to the source file that is associated with the compiled object. In this example, the >list command would replace the location of source code associated with the compiled object from /dirA/dev/sources to /dirB/dev/sources.

Where two or more replace commands are executed, the commands are stacked internally. Consider this case, where three replace commands are executed:

```
replace /dirA: /dirB
replace /dirC : /dirD
replace /dirE : /dirF
```

This provides a list of three possible replacements. Only the first matching replacement will be executed.

>replace ?

The replace ? command produces a list of active replacements.

Following our previous example, we would see:

```
/dirA : /dirB
/dirC : /dirD
/dirE : /dirF
```

>replace <no arguments>

The replace <no arguments> command re-sets the list command, removing all replacements established in the debugging session.

set

allows the user to set a <set parameter> to a different value.

The set command requires a <parameter>.

Parameters for the set command are:

prompt<prompt-string> Sets the debugger prompt to <prompt-string>

`var` <variable-name> <variable-value> Sets the value of <variable-name>
`varh` <variable-name> <variable-value> Sets the value of <variable-name> in hex notation

set prompt <prompt string>

sets the COBOL-IT Debugger prompt. The default setting for the COBOL-IT Debugger

prompt is (cobcdb).

Example :

(cobcdb)

event:13556

-event-end-stepping-range #0 hello () at C:/COBOL/COBOLIT/samples/hello.cb!9

set prompt >>>

>>>

set var <variable-name> <variable-value>

sets variable content for variable-name to variable-value. Values are converted to the appropriate type. A number stored in a PIC 999 field will be converted before storing.

Example :

(cobcdb)

set var message-line "hello hello"

\$1 = @hello.WORKING-STORAGE.message-line [11]"hello hello"

(cobcdb)

set varh <variable-name> <variable-value-hex>

sets variable content for variable-name to variable-value-hex.

<variable-value-hex> must be a valid hexadecimal string. Note that in a valid hexadecimal string, a single character space is recorded with two characters, so the total string length of <variable-value-hex> must be exactly two times the length of <variable-name>.

(cobcdb)

set varh ws-dummy 41

\$1 = @hello.WORKING-STORAGE.ws-dummy [1]"A"

(cobcdb)

step

causes execution of the program to execute a single step, and then break. An event-step command is issued. This is interrupted when an –event-end-stepping-range event takes place. The step command can be abbreviated as “s”.

Example:

```
(cobcdb)
```

```
step
```

```
-event-step
```

```
(cobcdb)-event-end-stepping-range #0 hello () at  
C:/COBOL/COBOLIT/samples/hello.cbl!14
```

stop

causes execution to stop (break) at the next statement

up -[n]

changes the current frame. When you have several levels of CALLs, the **info** functions relate to the current module. In a CALL'ed subprogram, **up -[n]** can be used to change the frame back to a previous CALL'ing module. **Info locals** can then be viewed for that calling module.

In the example below, the **bt** command shows 3 frames, with frame 0 being the current frame in a called sub-program, and the **info locals** command showing the state of the variables in the subprogram. **up -1** sets the frame to the calling program, so that **info locals** can be viewed for the calling program.

```
bt  
#0 subpgm () at C:/COBOL/COBOLIT/samples/subpgm.cbl!7  
#1 hello () at C:/COBOL/COBOLIT/samples/hello.cbl!25  
#2 hello () at C:/COBOL/COBOLIT/samples/hello.cbl!17  
(cobcdb)  
info locals  
@subpgm.WORKING-STORAGE  
  @subpgm.WORKING-STORAGE.RETURN-CODE = [10]" +000000000"  
  @subpgm.WORKING-STORAGE.TALLY = [10]" +000000000"  
  @subpgm.WORKING-STORAGE.SORT-RETURN = [10]" +000000000"  
  @subpgm.WORKING-STORAGE.NUMBER-OF-CALL-PARAMETERS =  
[10]" +000000000"  
  @subpgm.WORKING-STORAGE.COB-CRT-STATUS = [4]" "  
(cobcdb)  
up -1  
#1 hello () at C:/COBOL/COBOLIT/samples/hello.cbl!25  
(cobcdb)  
info locals  
@hello.WORKING-STORAGE  
  @hello.WORKING-STORAGE.RETURN-CODE = [10]" +000000000"  
  @hello.WORKING-STORAGE.TALLY = [10]" +000000000"  
  @hello.WORKING-STORAGE.SORT-RETURN = [10]" +000000000"
```

```
@hello.WORKING-STORAGE.NUMBER-OF-CALL-PARAMETERS =  
[10]"000000000"  
@hello.WORKING-STORAGE.message-line = [11]"XXXXXXXXXX"  
@hello.WORKING-STORAGE.ws-dummy = [1]" "  
@hello.WORKING-STORAGE.ctr = [6]"000000"  
@hello.WORKING-STORAGE.COB-CRT-STATUS = [4]"0000"  
(cobcdb)
```

version

returns the version of the cobcdb/COBOL-IT runtime.

Example:

```
(cobcdb)
```

```
version
```

```
~"COBOL-IT cobcdb 3.6.4\n"
```

```
^done
```

```
(cobcdb)
```

Debugger Events

-event-breakpoint-hit

Returned when a breakpoint is hit.

-event-continue

Returned by the continue command. Terminated by `-event-breakpoint-hit`.

-event-contreturn

Returned by the contreturn command. Terminated by `-event-end-stepping-range`.

-event-end-stepping-range

Returned when one of the debugger step commands (`step`, `next`, `contreturn`) reaches the end of its stepping range.

-event-next

Returned by the next command. Terminated by `-event-end-stepping-range`.

-event-program-exited

Returned by the kill command.

-event-step

Returned by the step command. Terminated by `-event-end-stepping-range`.

Our Sample Programs

For the purposes of this documentation, we are using a very short `hello.cbl` program as a reference.

(The program contains an `ACCEPT FROM COMMAND-LINE` statement, to illustrate this functionality in `cobcdb`.)

To compile: `>cobc -g hello.cbl`

`>cobc -g subpgm.cbl`

To run: `>cobcdb hello (or)`

To run with parameters: `>cobcd hello hello-world`

hello.cbl

```
000001 identification division.
000002 program-id. hello.
000003 environment division.
000004 data division.
000005 working-storage section.
000006 77 message-line pic x(11) value spaces.
000007 77 ws-dummy pic x value spaces.
000008 77 ctr pic 9(6) value 0.
000009 procedure division.
000010 main.
000011 accept message-line from command-line.
000012 if message-line not = spaces
000013 display message-line line 10 col 10
000014 else
000015 display "hello world" line 10 col 10
000016 end-if.
000017 perform para-1.
000018 display "returned from para-1" line 14 col 10.
000019 display "next line" line 16 col 10.
000020 accept ws-dummy line 16 col 30.
000021 stop run.
000022 para-1.
000023 move all "X" to message-line.
000024 display "in para-1" line 12 col 10.
000025 call "subpgm".
```

subpgm.cbl

```
000001 identification division.  
000002 program-id. subpgm.  
000003 environment division.  
000004 data division.  
000005 working-storage section.  
000006 procedure division.  
000007 main.  
000008     display "In Subpgm" line 20 col 10.  
000009     goback.
```



www.cobol-it.com

June, 2020

