Date of Last Review :08/01/2002

Compiler :Enterprise Cobol for Z/OS and OS/390 Version 3.1

Compiler options can be specified in the PARM parameter of the JCL's EXEC statement of the compile step. Though options can also be specified on a CBL card within the source code but this method is restricted. If none are specified, the default values will be used that are specified within MVS. Some of those MVS setting can not be overridden and some can only be overridden using a CBL cards. Check your compile listing for a definitive list of options in effect during your compile. The highlighted options are candidates for change from the current settings under COBOL/370 and COBOL II.

Recommended compiler setting, before Enterprise Cobol:

CURRENT DEFAULT	BATCH RECOMMENDED SETTI	CICS RECOMMENDED SETTING	STORED PROCEDURE RECOMMENDED SETTING
ADV	NOADV	n/a	NOADV
NOLIB	LIB	LIB	LIB
BUFSIZE(4096)	BUFSIZE(16K)	BUFSIZE(16K)	BUFSIZE(16K)
QUOTE - (LITCHAR)	APOST	APOST	APOST
NOMAP	MAP	MAP	MAP
NOLIST	LIST	LIST	LIST
TRUNC(OPT)	TRUNC(BIN)	TRUNC(BIN)	TRUNC(BIN)
NOXREF - (XREFOPT)	XREFOPT(FULL)	XREFOPT(FULL)	XREFOPT(FULL)
NOVBREF	VBREF	VBREF	VBREF
RENT	NORENT	RENT	RENT
RMODE(AUTO)	RMODE(ANY)	RMODE(AUTO)	RMODE(ANY)
FLAG(I,W)	FLAG(I,I)	FLAG(I,I)	FLAG(I,I)
NOOPTIMIZE	OPT(STD)	OPT(STD)	OPT(STD)
NOWORD	NOWORD	WORD(CICS)	WORD(OO)
SEQUENCE	NOSEQ	NOSEQ	NOSEQ
NOTYPECHK	n/a	n/a	TYPECHK
NOIDLGEN	n/a	n/a	IDLGEN ???
PGMNAME(COMPAT)	PGMNAME(COMPAT)	PGMNAME(COMPAT)	PGMNAME(LONGMIXED)
INTDATE(ANSI)	INTDATE(LILIAN)	INTDATE(LILIAN)	INTDATE(LILIAN)
NODYNAM	DYNAM	NODYNAM	DYNAM

I) General

1	DATA/NODATA	Creates a SYSDATA file required for remote compiles using VisualAge COBOL.
2	ANALYZE/NOANALY	Compiler will not check the syntax of embedded SQL and CICS EXEC's. The CICS Translator and DB2 Precompiler steps through their return code should tell us if something is syntactically incorrect.
3	ADV/NOADV	The default ADV setting causes IBM COBOL to add a carriage control character in the first byte of the record with "WRITE...ADVANCING" syntax
4	APOST	Compiler default is QUOTE. APOST has been IMServices setting. Use single quotes to delineate literals.
5	AWO/NOAWO	For AWO an implicit APPLY WRITE-ONLY clause is activated for all files in the program that are eligible For this clause. Eligible files are physical sequential, variable blocked type datasets. Activates APPLY WRITE-ONLY processing for Physical sequential files with VB format. NOAWO is the default and its being used as it is. Using APPLY WRITE-ONLY (AWO) can result in a performance savings since this will generally result in fewer calls to Data Management Services to handle.
6	BUFSIZE(15360)	Amount of main storage to the buffer for each compiler work data set. Larger sizing improves the efficiency. Related to option SIZE.
7	COMPILE(S)	Stops compile after "Severe" errors are encountered
8	**CURRENCY**	To define a default currency symbol for COBOL programs. We have NOCURRENCY for our shop
9	**THREAD**	To enable CBL program for execution in an LE enclave with multiple POSIX threads or PL/I tasks. The shop uses the default option NOTHREAD.
10	**ARITH** (EXTEND/COMPAT)	Set the maximum no of digits that you can specify for decimal data and effects the precision of intermediate results. For ARITH(EXTEND) the maximum number of digits is 31; with ARITH(COMPAT), the maximum number of digits is 18
11	**OPTIMIZE(FULL)**	OPTIMIZE with the new suboption of FULL optimizes object programs and provides improved run-time performance over both the OS/VS COBOL and VS COBOL II OPTIMIZE options. The compiler discards unused data items and does not generate code for any VALUE clauses for the discarded data items. We are using NOOPTIMIZE.
12	**PGMNAME** (COMPAT,LONGUPP LONGMIXED)	Controls the handling of names defined in a pgm. Refer COBOL for OS/390/VM Programming Guide.
13	DATA(31)	NORENT has not effect as it run in 24-bit AMODE addressing mode. For RENT compiled programs stores data above the line which in turn implies that all called subroutines must be able to use any passed data addresses above the line.
14	NODATEPROC(new)	Can only be activated if the IBM VisualAge Millennium Language Extensions of the compiler are installed. Could be used to find potential Y2K bugs.
15	NODBCS	DBCS cause the compiler to recognize shift codes for the double-byte portion of a nonnumeric literal
16	NODECK	Compiler output will not go to SYSPUNCH card deck.
17	DISK/PRINT (new)	Unique to COBOL under CMS
18	NODUMP	DUMP is not intended for general usage
19	DYNAM	DYNAM forces calls to subprograms to dynamic. Dynamic calls must be used if you wish an IBM COBOL

		program running in 31 bit addressing mode to call or be called by a program Link-Edited as AMODE(24). This option is set in combination with option RENT. DYNAM does not work with the CICS translator or with programs linked as dynamic link libraries (DLLs)
20	NOEXPORTALL (new OS/390 only)	EXPORTALL applies to when object deck is link-edited to form a DLL. Instructs the compiler to automatically export certain symbols when the object deck is link-edited to form a DLL
21	**FASTSRT**	This option allows IBM DFSORT to perform the input/output instead of COBOL. Use of FASTSRT requires review for programs with internal sorts to see if the sort file has an associated File Status or involve the same VSAM datasets for input and output. This option improves efficiency of internal sorts on QSAM files. If the Syncsort/COBOL accelerator cannot be used due to the aforementioned limitations it automatically falls back to a standard COBOL internal sort. The use of File Status on the sort file however is the programmers responsibility.
22	**FLAG(I,W)**	puts error messages of severity I and above at the end of the listing
23	NOFLAGMIG	conforms to COBOL 85 standard
24	NOFLAGSTD	To provide information by inserting messages about new and old CBL standards used within the pgm.
25	NOIDLGEN	Use the IDLGEN option to indicate whether SOM Interface Definition Language (IDL) should be generated for COBOL class definitions contained in the COBOL source file. Not to be used with setting SIZE(MAX). Review documentation before using this option
26	INTDATE(ANSI) (new)	Day 1 is Jan 1, 1601. Usage of INTDATE(LILIAN) will render all COBOL/370 programs with Intrinsic Functions obsolete
27	LANGUAGE(ENGLIS	Prints compiler listings in the language selected.
28	LINECOUNT(60)	Controls the number of lines printed on each page. Value 0 omits page ejects. 3 lines will always be used for headings
29	LIST (formerly PMAP under VS COBOL)	Produces an ASSEMBLER listing, a Program Global Table, a Constant Global Table, a Task Global Table, location and size of I/O & working storage control blocks, a dynamic storage information if NORENT is specified, a program initialization listing in ASM format, and locations of internal tables when the TEST (or COBOL II's FDUMP) setting is specified. The LIST setting is required for Xpeditor and mutually exclusive with OFFSET
30	MAP	Produces a data description listing showing fields used and their locations.
31	NONAME	Option NAME is used to generate a link-edit NAME card for each object module.
32	NONUMBER	NUM instructs when line numbers existing in columns 1 thru 6 of the source code are to be used. NONUM instructs COBOL to use its own generated line numbers.
33	NUMPROC(NOPFD)	Affects Packed Decimal and External Decimal fields only and pertains to signage. COBOL/370 book recommends NUMPROC(PFD) be used, but changing it can cause us data exception errors since more limited signage characters are allowed under that option. This setting performs invalid sign processing but is not as efficient.
34	OBJECT	Puts compiler output to the SYSLIN dataset
35	NOOFFSET	mutually exclusive with LIST shows helpful info for debugging using a condensed listing. OFFSET is

		more efficient then LIST but File-Aid Xpeditor requires LIST for its usage.
36	OUTDD(SYSOUT)	DISPLAY statements and internal sorts deposit their output in this DD. It can be used to resolve conflict when generating output in both ways, or if you wish to segregate certain output.
37	RMODE(AUTO)	With RENT, RMODE will be ANY and with NORENT it will be set to 24. This is the same as it was under COBOL/370 and COBOL II.
38	SEQUENCE/ NOSEQUENCE	When the numbers are supplied by the programmer in columns 1 thru 6 of the source code, the SEQUENCE option indicates that the compiler sould check for ascending order of line numbers, and issues a warning message if not. NOSEQUENCE IgnoreS checking the sequence of the line numbers.
39	SIZE(MAX)	Ensures use of maximum storage for any compile under OS/390
40	SOURCE	Used to get a listing of your program following the compile
41	SPACE	Source code listing spacing parameter
42	NOSSRANGE	Although SSRANGE can sometimes be useful for development, it is very inefficient. This feature detects addressing outside of the table range. To be used with the CHECK option. Avoid SSRANGE in production at all times
43	NOTERMINAL	Used to send progress and diagnostic messages to the SYSTERM data set
44	**TEST(NONE,SYM)**	Setting TEST(NONE,SYM) replaces FDUMP. Use of the TEST option allows a program to be debugged using IBM COBOL interactive and batch debuggers. Steps through Code displays and makes editable data items and allows path of program execution be altered. The compiler option TEST under IBM COBOL should be set to TEST(NONE,SYM) to replace former FDUMP under COBOL II. It is the most efficient choice to obtain formatted dumps during production ABENDs without affecting optimization. Right now a lot of COBOL/370 program have this option specified in the source on the first card because the FDUMP went away
45	**TRUNC(OPT)**	Applies only to USAGE BINARY receiving fields. Long argument in COBOL/370 book, it states TRUNC(OPT) as the preferred option to use. TRUNC(BIN) is brutally inefficient because arithmetic is done in packed decimal and converted back to binary. The TRUNC(OPT) option was added to improve efficiency of TRUNC(STD). The TRUNC(BIN) setting is currently in use for our COBOL II compiler. This setting is similar to NOTRUNC in VS COBOL. This setting prevents truncation of binary fields based on picture clause, which is good, but very inefficient and also rare. TRUNC(OPT) is efficient and as long as values don't exceed the associated picture clauses should provide consistent satisfactory results. Recommendation based on COBOL/370 by Harvey Bookman p 267.
46	NOTYPECHK	TYPECHK enforces the rules for object-oriented type checking. Stored Procedure compile skeletons will make use of this option. For large object-oriented programs do not use SIZE(MAX)
47	NOVBREF - compi	Used to obtain a COBOL Verb cross reference listing with your
48	XREF(FULL)	Produces a sorted cross reference on data and procedure names. Adopted from COBOL II.

49	DLL/NODLL (new)	from OS/390 version. Enables the compiler to generate an object module that is enabled for Dynamic Link Library (DLL) support. Our shop uses NODLL as default. The DLL setting is required for object- oriented COBOL syntax in combination with RENT, NODYNAM, and NOCMPR2. We will have to build an alternate CCF/MMF skeleton for Object Oriented modules. Pl find the explanation for the DLL support below
50	RENT	To make programs reentrant. One copy shared by many. RENT allows programs to run above or below the 16 MB line. NORENT compiled programs have to run below the line. All modules called from RENT compiled programs must be compiled with RES or they will ABEND with a U1005. Setting compile option RENT forces the RES option on the compiler. Compile option RENT produces AMODE(ANY) and RMODE(ANY). With THE NORENT option, AMODE & RMODE depend on the RES setting of the compile. Setting RES, produces AMODE(ANY) & RMODE(24). Setting NORES produces AMODE(24) & RMODE(24). AMODE(24) means a pgm can address other pgms and data with 24 bit addresses i.e. below the line. AMODE(31) means the pgm can address other pgms and data with 31 bit addresses, i.e. above the line. AMODE(ANY) can address programs and data in 31 or 24 bit mode. AMODE applies to dynamic calls only and not static calls. Programs in RMODE(24) are loaded, i.e. reside, under the 16 MB line. Programs in RMODE(ANY) can be loaded on either side of the line. Mode ANY can be overridden to 24 or 31 mode through the PARM of the linkage editor. Called modules should be converted first, but are no guarantee that data addressing errors won't occur while trying to communicate with a AMODE(24) compiled program. The RENT setting recommendation goes in conjunction with the Storage (re)initialization to low value setting and the DYNAM setting. Not making this change today will become a lingering issue in the future as storage and capacity demand continuous to grow. If all programs in an application are RENT compiled then we should link-edit the modules with the RENT link-edit attribute
51	WORD/NOWORD	WORD specifies the ending 4 characters of the name of the reserved-word table (as in IGYCXXXX) where XXXX are the first 4 standard characters of the name. This option facilitates the use of alternate reserved word tables as defined by the systems programmer. Reserved-word table NOOO keeps object oriented reserved words from being reserved in case an older program made use of that word. Compile skeletons for object-oriented programs must specify NOWORD. See the documentation for Reserved Words to look out for during conversion.
52	ZWB	Compiler removes the sign from a signed external decimal (DISPLAY) field when comparing this field to an alphanumeric elementary field during execution. NOZWB should be used to test input numeric fields for SPACES. Under COBOL II the setting has been ZWB.

We must make sure all programs within an application are compiled with RENT (and subsequently RES) to avoid data addressability errors. Partial recompilations of application programs will cause data communication failures with COBOL II programs compiled with AMODE(24). AMODE(24) are set by using compile option NORENT and/or NORES. Our current setting is NORENT so we have to spread the message that entire application must be re-compiled and tested. Ideally we should compile with RENT and DATA(31) to achieve virtual storage constraint relief but we must be careful as to how we get there. If you need to communicate with a AMODE(24) compiled program you must use the DATA(24) setting. Be careful of programs calling sub-programs calling sub-programs and universal subprograms that create data storage above the line.

II) Newly introduced in Enterprise Cobol

1) CICS - enables the integrated CICS translator capability and specifies CICS options. NOCICS is the default. We at present use NOCICS.

2) CODEPAGE - specifies the code page used for encoding contents of alphanumeric and DBCS data items at run-time as well as alphanumeric, national, and DBCS literals in a COBOL source program.

3) NSYMBOL (NATIONAL, DBCS) - controls the interpretation of the .N. symbol used in literals and picture clauses, indicating whether national or DBCS processing is assumed. Our shop uses NSYMBOL(NATIONAL)

4) SQL-Enables DB2 co-processor capability and specifies DB2 suboptions. Default is NOSQL.

III) Compiler Options not supported in Z/OS and OS/390:

CMPR2 , FLAGMIG , ANALYZE , **TYPECHK and IDLGEN

IV)Binder /Link Editor

Link-Edit Options:

There are 4 groups of link-edit options available for the linkage editor (also called binder or linker). They are:

- Module Attribute
- Special Processing
- Space Allocation
- Output

Link-Edit Options often have the same syntax names as compiler options however they serve a link-edit purpose which is quite different then its compiler purpose.

We currently specify the following parameters on the binder EXEC;

DFSMS/MVS V1 R5.0 BINDER
Calling IEWL using LET,LIST(ALL),MAP(YES),LET,XREF,AMODE(24)

Remove AMODE(24) let it default to what the compiler put out.
LIST, MAP, XREF,CALL, LET (Currently supplied for the binder and should continue) ,NORENT (Batch only)

For the Link-Edit step in Stored Procedure skeletons:

CASE (It allows for lowercase characters to be processed.)

V) RUN-TIME options

Cobol/370 uses a number of options that can be turned ON and OFF at the time the program is executed. This is accomplished by coding a PARM statement on the EXEC statement in the run time JCL.

e.g PARM='05/01/90/NOSSRANGE' passes 05/01/90 as a parameter to the program and specifies the NOSSRANGE run time option. COBOL searches backward throug the parameter list and if it finds a slash, it assumes that all inforamtion after the slash consists of run time parameters. Therefore code a slash at the end of the user parameter.

DLL Support in COBOL programs

Dynamic Link Library (DLL) support allows developers to split applications into units that are loaded only when required. Object -oriented (OO) applications can separate client code and class libraries into separate modules. DLL support improves system memory usage and offers greater flexibility for building, packaging, and delivering applications.

In addition to exploiting the runtime libraries available in OS/390 Language Environment (R), COBOL provides COBOL developers the means to easily build Dynamic Link Libraries (DLLs).

DLL support allows developers to split their applications into units and to direct these units to be loaded only when required. DLLs provide developers with more flexibility. Programmers can use DLLs to split applications into smaller modules and improve system memory usage. DLLs also offer more flexibility for building, packaging, and redistributing applications.

COBOL for OS/390 & VM provides DLL support for generating DLLs in a way similar to the way OS/2 (R) DLLs are generated. DLLs allow a function call or a variable reference in one load module to use a definition located in another load module at run time. The COBOL DLL support uses the same mechanisms as are used in OS/390 C/C++ and OS/390 SOMobjects, thus enabling object-oriented COBOL applications to smoothly interact with them

A) What is Dynamic Link Libraries - DLLs ?

A COBOL program runs another COBOL program by issuing "CALL" or "CHAIN" to a 'program' program.

Can COBOL run programs written in other languages? Can other languages call COBOL programs?

The answer to these questions is - yes/no/maybe depending on a host of factors such as which language on what platform and what standards. For example, COBOL has a high degree of affinity with C and these two languages are mixable - if you are careful with interface parameters.

C (and C++) has discrete field lengths of numeric in binary (user-defined types excepting), whereas COBOL allows continuous length of numeric in display, packed, binary or edited formats. Even though COBOL does some type matching for you, you need make sure that interface parameters are of the same data type.

A DLL (dynamic link library) is a **special container** of programs (*.obj) and resources (*.rc/*.res), which may or may not be related.). A DLL is essentially a file folder

that has a table of contents that makes it easy to call on a file in the folder. Unlike a file
folder, a DLL packs its contents into a single file - the dll.

Reasons to pack the contents into a single file:-

Reason 1 :-

Imagine you have 400 COBOL programs doing different functionality. If all of them is in single folder you will have gigantic folder and managing it would be cumbersome.

Instead of keeping all of my eggs in one folder, I keep them in several folders:

Folder 0: all programs that are common to all clients: print modules, user-access security, file reorgs, etc. Folder 1: general accounting Folder 2: inventory Folder 3: order entry
And so on. By the way, these programs are intermediate codes (*.int) that are called from a set of trigger (*.exe) programs.
If a customer wants general accounting programs, I copy Folder 0 and Folder 1 and send just these two folders. I don't have to worry about extra programs being sent to wrong clients.
Each folder has, on the average, 500-600 programs. In principle, I could pack these folders and create a DLL for each folder. At one time, I thought it was a good idea and did create DLLs but then keeping them updated with so many programs became impractical and I went back to the folder way. The lesson learned is that DLLs are OK for small folder - say a dozen or so programs.

Before any of its programs can be called, a DLL itself must be called. When a DLL is called, the runtime system loads its table of contents into RAM. Programs are loaded into RAM only if and when they are called.

In the code fragment below, COBAPI is a DLL and ALLOCPGM is a program

```
call "cob32api".

call winapi .
..
..
```

Loading the DLL either by CALL or SET first and then CALLing its contents work only if the contents are known by name. Some DLLs list only the ordinal number of its contents and so tghe above scheme does not work and one has to use special commands as shown in the code fragment below:

```
                    call winapi "LoadLibraryA" using
                         by reference z"kernel32.dll"
                         returning osKernel32
                    end-call

                    call winapi "GetProcAddress" using
                         by value osKernel32
                         by value 391          *> ordinal #
                *>          by reference z"GetTickCount"
                         returning ofPtr

                    call winapi ofptr
                        returning boola
```

B) How does this benefit?

For many applications DYNAM option provides the benefit of splitting the calling function to individual run time entity. So going DLL way is not productive and unneccessary. Also building DLLs for every change in the individual functions is counter productive.

This would help the applications when calling programs supplied by vendors and vendors can distribute the individual elements to one or more DLLs.

One place this could benefit is in the general purpose programs where no or very miniscule maintenance is done like (date and time modules, adm* modules and other assembly modules)

Web Links

Performance tuning for cobol programs