

# **Building Compile procs for Debugging and Abend Analysis**

**Kenneth.Tomiak@Schunk-Associates.com**

**February 25, 2003**

**Session 8250**

## **Trademarks and Copyrights**

- Trademarks and Registered trademarks used in this presentation are the property of their respective owners.
- Copyright ©2003 Schunk and Associates, Inc.
- Permission is hereby granted to SHARE, Inc. to distribute this presentation to conference attendees and in the conference proceedings. All other rights reserved.
- The information presented here has not been subject to any formal review. Use with your own discretion. No warranty is expressed nor implied.

Visit [www.Schunk-Associates.com](http://www.Schunk-Associates.com) to see this and other presentations made by the staff of Schunk and Associates, Inc. The latest and most up-to-date version is always found here.

## **Agenda**

- Performance
- Flexibility
- Old versus New
- Options
- Saving the pieces
- Linking versus Binding
- CICS, DB2, IMS
- Debuggers
- Abend Analysis
- Questions and Comments

Think you fixed your compile procedures as part of the Y2K effort? Whether you are a Systems Programmer or Application Developer, come to this session to hear the good, the bad, and the ugly about what this speaker keeps finding in procs today. Are you about to use the IBM Debug Tool and IBM Fault Analyzer products? Attend those sessions to learn how to use the product but come to this session to learn how to setup your compile procs for BOTH of them. Want to avoid problems getting the IBM Enterprise COBOL V3 compiler to work with your code? Use the right parms. Coding CICS TS applications? Find out if you should be using SCEECICS or COB2CICS. Use REXX with Fault Analyzer to automate dump handling and notification. Other tips that I present may eliminate some of your headaches, too. And I expect the audience will be able to add their two cents, good or bad. Visit [www.Schunk-Associates.com](http://www.Schunk-Associates.com), today, if you want the handout ahead of time or if you missed this session and can not wait for the conference proceedings.

## Performance

- VIO
  - Real 33x0 dasd or arrays
- BLKSIZE
  - 1/2 Track blocking
- PDS versus PDS/E
  - BLKSIZE=32760
- Block versus TRK versus CYL
- BUFNO is a NONO, REGION is GOGO

Why aren't you using VIO for truly temporary data sets? Where are your paging data sets? How much memory do you have?

In days of old, real 3380 and 3390 DASD volumes were used. These large platters had read/write heads on arms that had to traverse great distances. Today's arrays are made of many smaller drives that represent a logical volume. The distance for the arm movement is much shorter. And the data may be physically spread over several of the smaller drives.

Half tracking blocking for non-PDS/E data sets is still preferred. How many old procs still allocate by small blocks, inefficient for the new DASD type being used. The BLKSIZE used by the linkage editor has not been limited to 3200, or less, for a very long time.

PDS/E will efficiently use the track with a blksize of 32760. Just as the linkage editor does with a PDS RECFM=U.

Have your SPACE= parameters kept pace with your DCB BLKSIZE specifications? Did you really know how many units you needed? Does that CYL boundary really improve your performance or is it now an urban legend? In my testing, TRK produced fewer EXCPs than CYL!

Do you have many bogus and duplicate DD statements? Are SYSABEND or SYSUDUMP really required in your compile streams? What about SYSTEMM and SYSPUNCH?

Are you forcing CONTIG and coding secondary space?

Is the temporary SYSLMOD DCB the same as where you will copy the module?

DISP=OLD on your output data sets? You do care about data integrity, right? Do you use DISP=(MOD,PASS) for temporary datasets? In theory, you are asking the system to first

search for it, and if not found, to go ahead and allocate one. Would you really want to use a temporary data set that already existed? Why?

In my testing with BUFNO, not specifying it produced fewer EXCPs than when I tried any BUFNO greater than 2. Coding an adequate REGION size reduced the EXCPs far more. DSN=&&ANYNAME always went to VIO, even if UNIT=SYSDA was specified. If that is what your shop does, why not code UNIT=VIO to begin with?

Assembler was happiest with at least REGION=3M.

Are your SYSLIB concatenations referencing the libraries with the most hits first? LKED EXCPs dropped dramatically with SCEELKED as the first SYSLIB.

And I can not emphasize the value of using the right DCB. The CICS Transaction Server Assembler Translator saw an enormous improvement by specifying LRECL and BLKSIZE.

## Flexibility

- Saving compile listings.
  - For who's use?
- Order of translations
  - CICS or DB2 first?
- Where to specify compiler options.
  - Source contained options.
    - CBL
    - PLIOPTS
    - ASMOPTS
- STANDARDS

Are you saving the compile listing for testing or debugging? How do you ensure it is the right one? Do you compile once for testing and once for production or do you promote a test load module?

Do the programmer's use the compile listing or do they get fed into a software component? If people do now use them, can you save the condensed version instead?

Does your DB2 pre-compile wipe out your previously working DBRM? Do you care which translation happens first, or have you migrated to all of the newer releases of COBOL, DB2, and CICS TS and begun using the integrated compiler features?

As an MVS Systems Programmer, I want to leave compiler options as vendor provided defaults because it makes my life easier. As a former Applications Developer I want to have installation defaults set so my JCL is leaner. Having been around, I want special

need options to be set in with the source so we do not have to discover the right compile options years after you left. And there are times, as in batch versus CICS, that an option or two is best handled in JCL.

Declaring a Standard set of options to be used will make your life simpler when you need to migrate to a higher level of a compiler. There are many examples available for adding statements to every member of a PDS if your 'packaged application' needs something different.

### **Old versus New**

- It is a migration!
- LNKLST not STEPLIB.
- Migrate.
- Compatibility.
- There are differences, change code during the migration.

Just as MVT became MVS became MVS/XA became MVS/ESA became OS/390 became z/OS, compilers change too. Are you keeping up to date? Are you looking to do a major change by jumping over versions or do you do the incremental changes? Y2K should have pushed you to migrate to at least current coding styles. Thereafter, you will still find minor code changes.

Using the LNKLST version of compilers allows for fewer JCL changes. STEPLIB allows a 'temporary' use of the previous or future compiler. At some point, get back to the LNKLST version.

Plan on code changes. Maybe the handling of arithmetic functions has changed, syntax is different, or features are removed. Jumping major releases are going to require more code changes than jumping over minor releases.

Language Environment will run your old modules with few exceptions. As you start using the newer compilers you may have to relink many modules. Plan appropriately. OS/390 R10 introduced the ability to link with a higher level LE and run on a lower level, as long as new features were not used. What do you link with? Are you using a COMPAT option to lag behind for any good reason, or just to avoid making code changes?

Is there a better way to solve your application problem? Were you statically linking called subroutines instead of dynamically? Is now the time to change your method? Are you using 'calculate' when 'add' is all you wanted? Are you using subscripts the correct way? Why not index?

## Options

- WORD
- NOCMPR2
- RES, RENT, DYNAM
- DATA(31)
- AMODE, RMODE
- LIST, SOURCE, MAP, DMAP/PMAP
- COPY xxxxxxxx SUPPRESS

A small sampling of installation options that were the cause of migration problems.

WORD provides a list of reserved words not allowed in your data definitions. This is a handy way to stop your CICS programmers from using 'inconsiderate' functions. Did your installation provide the right modules with the new compiler? I've seen pages of errors when a required WORD reserved list module is not available.

Newer compilers are pushing newer options. Another nudge to migrate that old code. Are you taking advantage of 31 bit code? If not, why? 64 bit support is coming and you are still forcing 24 bit? Uh oh! Where your program resides is different than what it can address. And where did you have CICS put that storage? Does your CICS region go Short-on-storage? Below or above the line? The bar?

IBM's tools warn against using SUPPRESS on copybooks. It won't have all of the information it needs. And do not batch compile, one program per execution, please.

Which options speed up the compilation? I tweaked BUFSIZE(32760) for COBOL and SIZE(8192000) for PL/I saw a difference in the number EXCPs. During an MVS/XA Tuning class I learned the only good I/O is no I/O. Make sure the REGION size supports all of the internal space you are trying to use. I experienced S0F9 abends, wiping out the Initiator, due to an extra zero in the PL/I SIZE!

## Saving the pieces

- Listings
- Translators - CICS, DB2
- COBOL – TEST,SYM,SEPARATE
- ASM ADATA
- Pre-PLI,PLI
- LKED
- Object decks

Listings, in all their expanded glory, can help you find a failing instruction, lead you to fields with invalid data, or logic errors. Are you sure compiling your failing program

three years later produces the same module? Likewise, how do you know the saved listing is right for the module that executed?

DB2 has a DBRM and modified source. CICS has modified source. Some shops prefer saving modified source. In my opinion, this is too dangerous and will lead to problems. Let DB2 use packages if you need multiple versions of a DBRM.

COBOL supports SEPARATE for storing information useful for testing and debugging. The load module is slightly larger, while having access to lots of information. You'll need to manage new pieces. Assembler's ADATA can also help.

Other compilers and components output listings and pieces you may want to save. Do you prefer using only object decks to create a new load module or are you comfortable merging csects?

## Linking versus Binding

- PDS
- DLL (Unix System Services)
- PDS/E

Are you on the bleeding edge of e-computing? I've seen developer's trying to use vendor supplied procs that have requirements the shop does not support. The old tried and true method of linking is going to produce objects stored in a traditional PDS. And that is fine.

When you start looking to do UNIXy kinds of programming, then you will have to start using the BINDER. This is not the DB2 BIND. It's output is either a Unix file or a PDS/E member.

Do you need to use special LKED control cards? Store them somewhere so you don't waste time hunting down those sub-routines you need linked in.

## CICS, DB2, IMS

- Stand-alone step
- Integrated with COBOL
- CICS Language types
  - COBOL, COBOL2, COBOL3
- PLI
- Passing options

So now your shop has more than one group supporting compile procedures of various flavors. You have standards to ensure the best options are used by everyone, right? I've

seen the battles that the MVS group does not support the other pre-compilers and translators, and the other groups do not support the compilers. What to do, what to do? Take a tip from the Language Environment migration effort, 'Form a team to be the experts.'

No one group knows everyone's needs unless they are the 'compiler' group. The latest COBOL compiler incorporates integrated translators. That should make the quantity of compile procs required a smaller number. It will likely raise some issues on which options to use. And you really have to get off of the unsupported run-times and use Language Environment.

Have you changed your translators to output newer code or still using older code? Move along, move along. CICS can do COBOL3 these days. You'll want to use this.

Do you let PL/I precompile before translating? I've seen an options module, no real logic, require this. It is a cute feature, but is it really required?

How do you pass options through a precompile or translator to a compile? Sometimes JCL parms do the job. Sometimes concatenated input is the way to go.

## **Debuggers**

- Expediter
- IBM Debug Tool
- Intertest
- TSO Test
- CEDF
- DISPLAYS

How do you test your code for logic and definitional errors? Are you using a tool that uses the compile listing or are you flipping paper? Is your language supported or merely tolerated. Are you taking advantage of what you have?

Does the combination of products use the same listing? And how do they find them? COBOL has the right answer for me, it stores the listing's data set name in the object deck. IBM's Debug Tool is then able to reference it when requested. If not used, it can be migrated. IBM's Fault Analyzer requires some options to help locate the listing. Expediter keeps it in a VSAM data set, handy for the product, not so handy for you.

## **Abend Analysis**

- SYSUDUMP
- SVC Dumps
- CEEDUMP
- Transaction dumps
- IPCS
- Abendaid
- IBM Fault Analyzer (IDICNFUM)

The listing concerns for debugging apply here as well. How comfortable are you looking at a storage dump? The good news of having frequent abends is that you get good with the product of choice, fewer abends means less frequent use of a product which typically leads to longer analysis times.

Dumps store memory values for analysis. Do you really want to flip through pages of a dump or use a product to help isolate the problem area? As the programming environment gets larger and larger, so too, does the listing of storage values. You need help!

IPCS is available with MVS, but does your installation make all of the tools available to you? Many shops Lots of customization can be done to make this environment truly rewarding.

Abendaid and Fault Analyzer attempt to locate the problem instruction for you. The more listings you have available the better the analysis can be. Both will offer explanations and solutions.

Fault Analyzer will look for IDICNFUM in your tasklib concatenation (JOBLIB/STEPLIB). I discovered this was the only way to steer FA to the correct listing data sets across a large shop. Assemble into each project's LOADLIB to use different configuration members.

## Summary

- Save the pieces.
- Optimize your JCL.
- Standardize options.
- Imbed program specific options.
- Use the right stubs.
- Keep up with the times.

Keeping the right listing will serve you better than a listing that is almost right.

Is your mainframe ‘sucking wind’? A few JCL changes here and there will make your compilation process run faster and with fewer resources.

When every program has special needs you will become very proficient with the compiler options. Does that really help your programming needs? Pick a set of options the majority of programs can use. To quote or not to quote (apostrophes), that is the question. Store special options with the source. Concatenate group options in a separate member that precedes the program source in your compile stream.

There is a better CICS COBOL stub, odds are in my favor you still include the old one. Are you using DFHELII? Use Language Environment, not the old run-time libraries.

Move along, move along . . . . .

Visit <http://www.schunk-associates.com/presentations> to download the Adobe PDF formats of this presentation and handout. They will appear under *Kenneth Tomiak – Share100*.

<b>Listing Attributes</b>				
<i>DDNAME</i>	<i>DSNTYPE</i>	<i>RECFM</i>	<i>LRECL</i>	<i>BLKSIZE</i>
DEBGASM	PO-E	VB	8188	32760
DEBGVSCB	PO-E	FB	1024	31744
LISTASM	PO-E	VBM	133	32760
LISTKIXF	PO-E	FBA	121	32670
LISTKIXV	PO-E	VBA	125	32760
LISTLKED	PO-E	FBA	121	32670
LISTPLI2	PO-E	VBA	125	32760
LISTPLI3	PO-E	VBA	137	32760
LISTPLKD	PO-E	FBA	121	32670
LISTPPL2	PO-E	VBA	125	32760
LISTPPL3	PO-E	VBA	137	32760
LISTVSCB	PO-E	FBA	133	32718

I used PDS/E for all of these data sets to avoid having to run a compress.

All BLKSIZEs were system determined.

Attempt to use other RECFM and LRECL values will result in ABENDs!

DEBG\* files store modified data, not actual listings.

## Sample PL/I Job Stream Skeleton

```

/** TPP2D JCL CREATED AT &T2 ON &D2 BY &SYSUID
/**
/** LIB: DSN=SKELS(TPP2D)
/** GDE: Kenneth E. Tomiak
/** DOC: THIS PROCEDURE IS USED TO COMPILE A PL/I PROGRAM FOR CICS TS.
/**
/**-----*
/** FUNCTION - PRECOMPILE A PL/I V2R3M0 PROGRAM.
/** INPUT - MEMBER FROM SOURCE LIBRARY
/** OUTPUT - COMPILE LISTING
/** - MODIFIED SOURCE
/**-----*
//P1 EXEC PGM=IEL0AA,REGION=4M,
// PARM=( 'SZ(1024K),INC,NIS,NOSYNTAX,MDECK,M,NS' )
//STEPLIB DD DISP=SHR,DSN=PLI.V2R3M0.PLICOMP
//SYSPRINT DD DISP=SHR,DSN=&EPENV..&SSA..LISTPPL2(&MBRNAME)
//SYSLIN DD DUMMY
//SYSPUNCH DD DSN=&&SRCIN,DISP=(MOD,PASS),UNIT=SYSDA,
// SPACE=(TRK,(25,10)),DCB=(BLKSIZE=0)
//SYSUT1 DD DSN=&&SYSUT1,UNIT=SYSDA,
// SPACE=(1024,(200,50),,CONTIG,ROUND),DCB=BLKSIZE=1024
//SYSIN DD DISP=SHR,DSN=&SSDSN($IDIPLI2)
// DD DISP=SHR,DSN=&SSDSN(&MBRNAME)
//SYSLIB DD DISP=SHR,DSN=&SSDSN
// DD DISP=SHR,DSN=SYS1.CEE.SCEESAMP
)CM ++++++
)CM COMPILER SYSLIB CONCATENATION, THREE FROM USER
)CM ++++++
// DD DISP=SHR,DSN=&CSLB1
)SEL &CSLB2 ^= &Z
// DD DISP=SHR,DSN=&CSLB2
)ENDSEL
)SEL &CSLB3 ^= &Z
// DD DISP=SHR,DSN=&CSLB3
)ENDSEL
/**
/**-----*
/** FUNCTION - COPY THE TRANSLATE LISTING IF IT FAILED
/** INPUT - MEMBER FROM LISTING LIBRARY
/** OUTPUT - TRANSLATOR LISTING
/**-----*
//PG EXEC PGM=IEBGENER,REGION=2048K,
// PARM=' ',COND=(0,GT,P1)
//SYSPRINT DD DUMMY
//SYSIN DD DUMMY
//SYSUT1 DD DISP=SHR,DSN=&EPENV..&SSA..LISTPPL2(&MBRNAME)
//SYSUT2 DD SYSOUT=*
/**
/**-----*
/** FUNCTION - TRANSLATE THE SOURCE PROGRAM
/** INPUT - MEMBER FROM SOURCE LIBRARY
/** OUTPUT - TRANSLATOR LISTING
/** - EXPANDED SOURCE
/**-----*

```

```

//T1      EXEC PGM=DFHEPP1$,REGION=4M,PARM=' SP '
//STEPLIB DD DISP=SHR,DSN=CICS.TS.V1R3M0.SDFHLOAD
//SYSPRINT DD DISP=SHR,
//          DSN=&EPENV..&SSA..LISTKIXV(&MBRNAME)
//SYSPUNCH DD DSN=&&PLIOUT,DISP=(NEW,PASS),UNIT=TEMP,
//          SPACE=(8368,(5,5)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=8240)
//OUTPUT  DD DUMMY
//SYSDUMP DD DUMMY
//SYSIN   DD DISP=SHR,DSN=&&SRCIN
//*
/*-----*
/* FUNCTION - COPY THE TRANSLATE LISTING IF IT FAILED
/* INPUT    - MEMBER FROM LISTING LIBRARY
/* OUTPUT   - TRANSLATOR LISTING
/*-----*
//TG      EXEC PGM=IEBGENER,REGION=2048K,
//          PARM='',COND=(5,GT,T1)
//SYSPRINT DD DUMMY
//SYSIN   DD DUMMY
//SYSUT1  DD DISP=SHR,DSN=&EPENV..&SSA..LISTKIXV(&MBRNAME)
//SYSUT2  DD SYSOUT=*
//*
/*-----*
/* FUNCTION - COMPILE A PL/I V3R1M0 PROGRAM.
/* INPUT    - MEMBER FROM SOURCE LIBRARY
/* OUTPUT   - COMPILE LISTING
/*          - OBJECT DECK
/*-----*
//C1      EXEC PGM=IEL0AA,PARM='CMPAT(V1),OBJ,ND,INCLUDE,GN',
//          COND=(5,LT,T1),REGION=4M
//STEPLIB DD DISP=SHR,DSN=PLI.V2R3M0.PLICOMP
//SYSIN   DD DSN=&&PLIOUT,DISP=(OLD,DELETE)
//SYSPRINT DD DISP=SHR,DSN=&EPENV..&SSA..LISTPLI2(&MBRNAME)
//SYSLIN  DD DSN=&&LOADSET,DISP=(NEW,PASS),UNIT=TEMP,
//          SPACE=(80,(250,100))
//SYSUT1  DD UNIT=TEMP,SPACE=(1024,(300,60),,CONTIG),
//          DCB=BLKSIZE=1024
//SYSLIB  DD DISP=SHR,DSN=&SSDSN
//          DD DISP=SHR,DSN=CICS.TS.V1R3M0.SDFHPL1
//          DD DISP=SHR,DSN=CICS.TS.V1R3M0.SDFHMAC
)CM ++++++
)CM COMPILER SYSLIB CONCATENATION, THREE FROM USER
)CM ++++++
//          DD DISP=SHR,DSN=&CSLB1
)SEL &CSLB2 ^= &Z
//          DD DISP=SHR,DSN=&CSLB2
)ENDSEL
)SEL &CSLB3 ^= &Z
//          DD DISP=SHR,DSN=&CSLB3
)ENDSEL
//*
/*-----*
/* FUNCTION - COPY THE TRANSLATE LISTING IF IT FAILED
/* INPUT    - MEMBER FROM LISTING LIBRARY
/* OUTPUT   - TRANSLATOR LISTING
/*-----*
//CG      EXEC PGM=IEBGENER,REGION=2048K,

```

```

//          PARM=' ',COND=( ( 5,LT,C1 ) , ( 5,LT,T1 ) )
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
//SYSUT1   DD DISP=SHR,DSN=&EPENV..&SSA..LISTPLI2(&MBRNAME)
//SYSUT2   DD SYSOUT=*
//*
/*-----*
/*  FUNCTION - LINK NEW PROGRAM INTO TEMPORARY LOAD LIBRARY
/*  INPUT   - OBJECT DECK FROM COMPILE STEP
/*  OUTPUT  - NEW TEMPORARY LOAD MODULE
/*          - BINDER LISTING
/*-----*
//L1       EXEC PGM=IEWL,COND=( ( 5,LT,T1 ) , ( 5,LT,C1 ) ) ,REGION=7M,
//          PARM=( 'LET,LIST,XREF,MAP' )
//SYSLIB   DD DISP=SHR,DSN=&SLDSN
//          DD DISP=SHR,DSN=CICS.TS.V1R3M0.SDFHLOAD
//          DD DISP=SHR,DSN=PLI.V2R3M0.SIBMBASE
//          DD DISP=SHR,DSN=PLI.V2R3M0.PLIBASE
//          DD DISP=SHR,DSN=SYS1.CEE.SCEELKED
)CM ++++++
)CM LKED SYSLIB CONCATENATION, THREE FROM USER
)CM ++++++
//          DD DISP=SHR,DSN=&BSLB1
)SEL &BSLB2 ^= &Z
//          DD DISP=SHR,DSN=&BSLB2
)ENDSEL
)SEL &BSLB3 ^= &Z
//          DD DISP=SHR,DSN=&BSLB3
)ENDSEL
//SYSUT1   DD UNIT=TEMP,SPACE=( 8368,( 20,5 ) )
//SYSPRINT DD DISP=SHR,DSN=&EPENV..&SSA..LISTLKED(&MBRNAME)
//SYSLMOD  DD DSNAME=&&PGMLIB,
//          SPACE=(TRK,(10,10,1)),
//          UNIT=SYSDA,DISP=(MOD,PASS)
//SYSLIN   DD DSN=&&LOADSET,DISP=(OLD,DELETE)
)SEL &LKCTL ^= &Z
//          DD DISP=SHR,DSN=&LKCTL
)ENDSEL
//          DD *
//          INCLUDE SYSLIB(DFHPL1OI)
//          NAME &MBRNAME(R)
/*
/*
/*-----*
/*  FUNCTION - COPY THE TRANSLATE LISTING IF IT FAILED
/*  INPUT   - MEMBER FROM LISTING LIBRARY
/*  OUTPUT  - TRANSLATOR LISTING
/*-----*
//LG       EXEC PGM=IEBGENER,REGION=2048K,
//          PARM=' ',COND=( ( 5,LT,T1 ) , ( 5,LT,C1 ) , ( 5,LT,L1 ) )
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
//SYSUT1   DD DISP=SHR,DSN=&EPENV..&SSA..LISTLKED(&MBRNAME)
//SYSUT2   DD SYSOUT=*
/*
/*
//IF&MYIF IF ( &STEPCOND ) THEN

```

```
//*
//*-----*
//* FUNCTION - COPY NEW PROGRAM INTO PERMANENT LOAD LIBRARY
//* INPUT - LOAD MODULE FROM BINDER STEP
//* OUTPUT - PERMANENT LOAD MODULE
//* - UTILITY LISTING
//*-----*
//I1 EXEC PGM=IEBCOPY,COND=((5,LT,C1),(4,LT,L1)),REGION=1024K
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COPY INDD=((SYSUT1,R)),OUTDD=SYSUT2
SELECT M=((&MBRNAME,,R))
/*
//SYSUT1 DD DISP=(OLD,DELETE),DSNAME=&&&PGMLIB
//SYSUT2 DD DISP=SHR,DSNAME=&SLDSN
//IF&MYIF ENDIF
//* ENDIF&MYIF ENDIF&MYIF ENDIF&MYIF
//*
//
```