CA InterTest™ and CA SymDump® - 11.0

Symbolic Support

Date: 06-Jun-2018

Table of Contents

Symbolic Support

This section is intended as a reference for programmers using the symbolic support features of CA Application Quality and Testing Tools.

What is Symbolic Support?

The term *symbolic support* refers to the use of source code information from application programs to enhance and simplify the use of CA Application Quality and Testing Tools products for z/OS.

Some of these products include:

- CA InterTest Batch
- CA InterTest for CICS
- CA Optimizer/II
- CA SymDump Batch
- CA SymDump for CICS

These products provide application programmers with the critical tools needed to improve productivity throughout the application life cycle. Symbolic support makes these products easier to learn and use by speaking to programmers using terms that they recognize and understand from their own source code.

For example, using symbolic support with an interactive debugger like CA InterTest Batch lets programmers do the following:

- Enter breakpoint commands right on the source listing display.
- Stop execution at every label in a program.
- Automatically display the values of referenced variables at each statement.
- Easily display the value of any program variable.
- Set conditional breakpoints based on variable values.
- View a trace of all previously executed source statements.

Symbolic support eliminates the need to manually locate variables in storage, compute program offsets for source statements, or determine which statements were executing. You do not need to keep program listings open while debugging. All of this is done for you automatically when you use symbolic support with the CA Application Quality and Testing Tools products for z/OS.

How Does Symbolic Support Work?

When your application programs are compiled or assembled, symbolic information about the program is written to various reports in the output listing. A program called a postprocessor reads the output listing, collects the symbolic information, and stores it in a symbolic repository called a PROTSYM.

Using the listing postprocessors to collect symbolic information does not alter your program in any way. The listing produced by your compiler or assembler is used only as input. Your object module is not altered. Only the PROTSYM is updated.

After the symbolic information has been stored in the PROTSYM, you can access the information by any of the CA Application Quality and Testing Tools products to provide symbolic support for your application.

Symbolic Support for Optimized Applications

CA Application Quality and Testing Tools supports programs that have been optimized, either by the COBOL compiler's OPTIMIZE option or by CA Optimizer or CA Optimizer/II. However, debugging and post mortem analysis of these programs can sometimes result in unexpected behavior.

⚠

Note: The PL/I compilers are optimizing compilers.

Often as part of the optimization process, a compiler will relocate individual instructions, statements, or even entire paragraphs so that the optimized program will run more efficiently. This means that some or all of the instructions generated for a given statement may be moved to another statement, or that some or all of the statements in a paragraph may be moved to another paragraph. When this type of optimization occurs, the resulting object program and corresponding listing may not accurately represent the relationship between the source statements and their generated object code, or even between a paragraph label and the statements contained within the paragraph. As a result, there may be times when the breakpoint intercept does not occur, or when the wrong sequence of statements appears to be executed while single-stepping, or when the abending object code does not correspond to the correct source statement. There may also be times when the debugger appears to highlight the wrong statement at a breakpoint intercept or the dump analysis identifies the wrong statement as the abending source statement.

These unexpected displays do not indicate that a program is being executed incorrectly or that an abend is being incorrectly analyzed. They simply indicate that the debugger or dump analyzer sometimes cannot accurately identify exactly which object code corresponds to which source statement, or which statement is contained within which paragraph.

The CA Application Quality and Testing Tools products use the information in the compiler-generated procedure map or offset report to establish the program offset for each statement and label in the program. During execution or abend processing, the debugger or abend analyzer recognizes the start of the new statement or label by matching the program offset of the currently executing instruction with the PROTSYM information obtained from the compiler listing. Therefore, the accuracy with which the debugger or abend analyzer can represent a breakpoint or other intercept or the abending statement is only as good as the information in the compiler listing.

Inaccuracies may include, but will not be limited to:

- Incorrect execution when using the SKIP, GO stmt# or CS stmt# commands
- Failure to stop at a breakpoint at a paragraph label or statement
- Unexpected or out of sequence highlighting of statements when single-stepping
- Incorrect identification of the statement which contains the abending object code

Additionally, application abends may result from the use of the SKIP, GO stmt# or CS stmt# commands because the optimized object code may have register requirements that do not support changes to the flow of control. These commands should be avoided when debugging an optimized program.

For the best debugging results, avoid using optimization whenever possible in your testing environment. Production applications may be compiled with optimization, and debugging these applications as they exist without recompiling is supported. However, be aware that you may experience some of the inaccuracies listed previously under these circumstances.

Supported Compilers and Assemblers

Symbolic information is currently supported for programs compiled or assembled by the following IBM products:

- OS/VS COBOL
- OS PL/I
- VS COBOL II
- AD/CYCLE COBOL/370
- COBOL for MVS
- COBOL for VM
- Enterprise COBOL for z/OS
- Visual Age PL/I
- PL/I for MVS and VM
- Enterprise PL/I for z/OS
- High Level Assembler for MVS and VM and VSE
- Assembler H

Considerations for Using the Integrated Preprocessors

The integrated CICS translator and integrated SQL coprocessor of Enterprise COBOL for z/OS are fully supported by the postprocessor. It should be noted, however, that duplicate statement numbers for those statements generated by the integrated preprocessors are not saved in the PROTSYM. The CA Application Quality and Testing Tools products required this modification to the saved listing. In addition, the compiler's LIST option is required to correctly load the symbolic information into the PROTSYM file.

The postprocessor also supports the integrated CICS and SQL preprocessors of PL/I for z/OS. However, programs that contain EXEC SQL INCLUDE statements for user-defined members still require a separate precompile step. (EXEC SQL INCLUDE statements for SQLCA and SQLDA are supported when using the integrated SQL preprocessor.) It should also be noted that duplicate statement numbers for those statements generated by the integrated preprocessors are not saved in the PROTSYM. The CA Application Quality and Testing Tools products required this modification to the saved listing.

The integrated INCLUDE and MACRO preprocessors of PL/I for z/OS are not supported. A separate precompile step is required when incorporating external files into your program.

PROTSYM File

The PROTSYM file is a VSAM relative record data set (RRDS) with an upper limit of approximately four million 2 KB data records and capable of storing symbolic information for up to 147,000 application programs at one time.

The PROTSYM file is defined by IDCAMS and must be initialized by program IN25UTIL before you can add symbolic information.

Member CAVHPROT in CAI.CAVHJCL contains sample JCL that you can use to allocate and initialize a PROTSYM file.

⚠ **Note:** The PROTSYM file cannot reside in the LSR pool.

Sharing PROTSYM Files

Your PROTSYM files can be shared between CA Technologies products and across multiple systems and environments. A single PROTSYM file contains symbolic information for both CICS and batch programs.

Use RESERVE and DEQ macros when updating the PROTSYM file to allow sharing of the file between regions and systems. The resource major name used in the RESERVE and DEQ macros is INTERTST. If your installation uses a service that converts RESERVEs into cross-system ENQs, define the major name INTERTST to the service.

Depending on your needs, you can maintain more than one PROTSYM file at your installation. All of the CA Application Quality and Testing Tools products support the use of multiple PROTSYM files.

Loading Symbolic Information

By modifying the JCL procedures used to compile or assemble your applications, you can automatically update the symbolic information in your PROTSYM file every time a program is rebuilt. This is the easiest way to help ensure that the symbolic information in your PROTSYM file matches the executable for every program. This is also the method that CA Technologies recommends for maintaining symbolic information.

Alternatively, you can save the listings from your compiles or assemblies and load the symbolic information later as needed. If you choose this method, you can load symbolic information into your PROTSYM file using a separate batch job. CA Technologies provides batch utilities that let you load one or more program listings residing in partitioned data sets (PDS or PDSE), CA Librarian, CA Panvalet, or CA Endevor SCM format.

Some of the CA Application Quality and Testing Tools products provide additional online functionality for viewing and maintaining PROTSYM files. For more information about the online utilities for any CA Technologies product, see the documentation for that product.

Creating a PROTSYM File

CAVHPROT

Member CAVHPROT in CAI.CAVHJCL contains sample JCL for defining and initializing a PROTSYM file. The INITIALIZE, UPDATE, DELETE, PURGE, and RELOAD functions are protected by a password or external security. The default password is 12345678. To modify the password or enable external security, run the member CAVHCONF in CAI.CAVHJCL with a modified version of member CAVHCONF in CAI.CAVHMAC as input before you create your PROTSYM files. You can specify external security or password protection, but not both.

Note: If you are installing one of the CA Testing and Fault Management products, be sure to follow the instructions for creating the PROTSYM file in Configuring for the product you are installing. Some products may provide custom JCL members that have been tailored for use with the product.

CAVHPROT contains the following two steps:

- Step 1 (DEFSYM) -- Invokes IDCAMS to define the PROTSYM file.
- Step 2 (LOAD) -- Invokes IN25UTIL to initialize each of the PROTSYM records.

The following JCL for member CAVHPROT shows these two steps:

```
//CAVHPROT JOB
//DEFSYM   EXEC PGM=IDCAMS,REGION=1024K
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
   DELETE $PROTSYM$ CLUSTER PURGE
   SET MAXCC=0
   DEFINE CLUSTER (NAME($PROTSYM$)          -
                   REC($RECS$)                  -
                   CISZ(2048)           /* DO NOT CHANGE */        -
                   VOLUME($SYMVOL$)             -
                   RECSZ(2040 2040)             -
                   SHR(4 4)                 -
                   NUMBERED)                    -
          DATA (NAME($PROTSYM$.DATA))
/*
//LOAD     EXEC PGM=IN25UTIL,REGION=2048K
//STEPLIB  DD DSN=$LOADLIB$,DISP=SHR
//MESSAGE  DD SYSOUT=*
//PROTSYM  DD DSN=$PROTSYM$,DISP=SHR
//CARDS    DD *
PASSWORD=$PASSWORD$
INITIALIZE
REPORT
/*
//
```

Make the following substitutions in member CAVHPROT:

- **$PROTSYM$**
  The fully-qualified name of your new PROTSYM library.

- **$SYMVOL$**
  The volume on which the PROTSYM resides.

- **$RECS$**
  The primary space allocation in records. (See Notes 1 and 2.)

- **$PASSWORD$**
  The one- to eight-character PROTSYM update password for your installation, from CAVHCONF. (See Note 3.)
  If external security for the symbolic component is enabled in CAVHCONF, this keyword is not required and will be ignored if it is specified.

- **$LOADLIB$**
  The product target library.

Submit the JCL to allocate and initialize a new PROTSYM file.

1. Do not allocate any secondary space.

2. The space required depends on many factors including the size of your programs, the number of variables and labels, the average length of their names, and the LISTER options used for loading symbolic information. We recommend an initial allocation of 10,000 records. You can allocate new PROTSYM files as needed, and expand and reorganize existing files.

3. If you have not altered the installation default, specify PASSWORD=12345678.

4. The PROTSYM share parameters must be SHR(4,4).

5. To allow multiple program version support, include the MAXPGMVER=parameter.

   > **Note:** For more information about the MAXPGMVER parameter, see Maintaining a PROTSYM File (see page 53).

Adding Symbolic Information

You can add symbolic information to your PROTSYM files using the following postprocessors:

- IN25SYMC (see page 20)
  Loads symbolic information for programs compiled using:
  - OS/VS COBOL version 2.3 plus PTF8 or higher
  - CA Optimizer
- IN25COB2 (see page 25)
  Loads symbolic information for programs compiled using:
  - Enterprise COBOL for z/OS
  - IBM COBOL for VM
  - IBM COBOL for MVS and VM
  - AD/CYCLE COBOL/370
  - VS COBOL II
  - CA Optimizer/II
- IN25SYMP (see page 33)
  Loads symbolic information for programs compiled using:
  - Enterprise PL/I for z/OS
  - IBM PL/I for MVS and VM
  - Visual Age PL/I
  - OS PL/I
- IN25SYMA (see page 39)
  Loads symbolic information for programs compiled using:
  - High level Assembler for MVS and VM and VSE
  - Assembler H
- IN25LINK (see page 44)
  Reads IBM linkage editor output to collect and load subroutine mapping information for composite load modules.

- IN25SYMD (see page 50)
  Loads multiple COBOL, C, PL/I, and Assembler listings residing in PDS, PDSE, CA Librarian, CA Panvalet, or CA Endevor SCM format.

IN25SYMC

Use program IN25SYMC to load symbolic information for programs compiled using OS/VS COBOL or CA Optimizer.

You can execute IN25SYMC as a standalone batch job to load a single COBOL listing that has been previously saved to a permanent file, or add it to your existing OS/VS COBOL or CA Optimizer JCL procedure. The method you select depends entirely on the procedures at your own installation. Both methods are described in this article.

- IN25SYMC JCL (see page 20)
- IN25SYMC Options (see page 21)
- Required OS/VS COBOL Options (see page 23)
- Executing IN25SYMC as a Standalone Program (see page 23)
- Adding IN25SYMC to Your OS/VS COBOL Procedure (see page 24)

IN25SYMC JCL

The following table describes the DD statements used by IN25SYMC:

DDname	Description
STEPLIB	The load library containing IN25SYMC.
INPUT	The listing that was written to SYSPRINT by the OS/VS COBOL compiler, or by CA Optimizer, during compilation.
OUTPUT	All or part of the original compiler listing is written to this file, depending on your request.
MESSAGE	Any messages produced by IN25SYMC during postprocessing are written here.
PROTSYM	The file to which the symbolic information is written.
CARDS	The input control statements that define the request.

Note: If you are adding a new step for IN25SYMC to a JCL procedure, use program IN25PARM to write your input control statements to the CARDS file.

IN25SYMC Options

Options are passed to IN25SYMC using a parameter statement in the CARDS DD. Specify the parameter statement as an in-stream control card, or when using a JCL procedure, generate it using program IN25PARM.

The following JCL shows these options:

```
//IN25PARM EXEC PGM=IN25PARM,PARM='parameter statement'
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//CARDS    DD DISP=(,PASS),DSN=&&CARDS,UNIT=SYSDA,SPACE=(TRK,(1,1))
```

Parameter statements in the CARDS DD must begin in column 1.

The program name is the only required field on the parameter statement. This positional parameter defines the name that is used to store the symbolic information in the PROTSYM file. This name is used by the CA Application Quality and Testing Tools products to locate the symbolic information and is displayed when listing the contents of your PROTSYM.

In most cases, this name should be the same as the PROGRAM-ID. However, when loading symbolic information for use with CA InterTest for CICS, you must specify the name of the CICS program definition, or when using composite support, specify the monitor name.

The following example shows an in-stream parameter statement that you can use to save symbolic information using the name ORDEDIT:

```
//CARDS  DD *
ORDEDIT
/*
```

Controlling Printed Output with the CUTPRINT Option

Because you can load symbolic information from a permanent data set or a temporary listing file, you can also print all or part of the listing generated by the compiler.

Append the CUTPRINT option to your parameter statement to control printing of the compiler listing as follows:

- **,CUTPRINT=ALL**
  Do not print any of the compiler listing.
- **,CUTPRINT=MAP**
  Print the listing up to, but not including, the Data Division Map report.
- **,CUTPRINT=REF**
  Print the listing up to, but not including, the cross reference of data names.

The following sample parameter statement saves symbolic information for program ORDEDIT and prints only the source code section of the compiler listing:

```
//CARDS   DD *
ORDEDIT,CUTPRINT=MAP
/*
```

Note: Specify the CUTPRINT parameter only when you do not want all or part of your listing printed. The entire listing is printed if this parameter is omitted.

Saving Your Listing for Online Display with the LISTER Option

Append the LISTER option to your parameter statement to control which portion of your source listing is saved to the PROTSYM file as follows:

- **,LISTER=ALL**
  Saves the entire OS/VS COBOL listing.
- **,LISTER=MAP**
  Saves the OS/VS COBOL listing up to, but not including, the Data Division Map report.
- **,LISTER=REF**
  Saves the OS/VS COBOL listing up to, but not including, the cross reference of data names.

The following sample parameter statement saves symbolic information for program ORDEDIT, does not print any of the listing, and saves the listing up to, but not including, the Data Division Map report to the PROTSYM file:

```
//CARDS DD  *
ORDEDIT,CUTPRINT=ALL,LISTER=MAP
/*
```

Notes:

- If the LISTER parameter is omitted, no listing is saved in the symbolic file.
- The LISTER parameter is required for use with CA Optimizer, CA SymDump Batch, and CA InterTest Batch.
- To reduce overhead and save space in your PROTSYM file, we recommend that you specify LISTER=MAP when executing IN25SYMC.

Setting Data as Nonpurgeable

You can mark any saved symbolic data for this program as nonpurgeable. If a program's data is marked as nonpurgeable, the data is not removed from the PROTSYM when deleting programs using a purge interval batch run. However, you can delete the data by program name. For instructions on deleting data from the symbolic file, see Maintaining a PROTSYM File (see page 53).

To mark data as nonpurgeable, add the NOPURGE option to your parameter statement as the last option.

The following sample parameter statement saves symbolic information for program ORDEDIT, prints the entire listing, saves the entire listing in the PROTSYM file, and does not let symbolic data be removed from the symbolic file by a purge interval batch run.

```
//CARDS DD  *
ORDEDIT,LISTER=ALL,NOPURGE
/*
```

Required OS/VS COBOL Options

The following compiler options are required to load symbolic information for OS/VS COBOL programs into the PROTSYM file:

Option	Description
CLIST or PMAP	Produces a condensed Procedure Division map or full Assembler Procedure Division map.
DMAP	Produces a Data Division map.
NONUM	Suppresses compiler-generated line numbers.
SXREF	Produces a cross-reference of data and paragraph names.
VERB	Produces a report of verb names.

The following compiler options are required to load symbolic information for a program compiled using CA Optimizer into the PROTSYM file:

Option	Description
DMAP or MDMAP	Produces a Data Division map or merged Data Division map.
MLIST	Produces a merged Procedure Division map.
NONUM	Suppresses compiler-generated line numbers.
XREF	Produces a cross-reference of data and paragraph names.

To use symbolic references in OS/VS COBOL, you must declare at least one data item in working storage.

Executing IN25SYMC as a Standalone Program

Member CAVHSYMC in CAI.CAVHPROC contains sample JCL for executing postprocessor IN25SYMC as a standalone batch job. Use this member to load symbolic information from previously saved OS/VS COBOL listings.

```
//CAVHSYMC PROC PROTSYM=CAI.PROTSYM,
//             NAME=XXXXXXXX,
//             LISTLIB=USER.LISTLIB,
//             MEMBER=XXXXXXXX,
//             LISTER=ALL,
//             CUTPRINT=ALL
//*
//IN25PARM EXEC PGM=IN25PARM,REGION=512K,
//             PARM='&MEMBER,LISTER=&LISTER,CUTPRINT=&CUTPRINT'
```

```
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//CARDS    DD DSN=&&CARDS,DISP=(,PASS),
//            UNIT=SYSDA,SPACE=(TRK,(1,1))
//*
//IN25SYMC EXEC PGM=IN25SYMC,REGION=2M
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//PROTSYM  DD DISP=SHR,DSN=&PROTSYM
//INPUT    DD DISP=SHR,DSN=&LISTLIB(&MEMBER)
//CARDS    DD DSN=&&CARDS,DISP=(OLD,DELETE)
//OUTPUT   DD SYSOUT=*,DCB=(LRECL=121,BLKSIZE=2440,RECFM=FBA)
//MESSAGE  DD SYSOUT=*
//
```

You can override the following procedure variables:

Variable	Description
PROTSYM	Specifies the name of the symbolic file being updated.
NAME	Specifies the name that is used to store the symbolic information in the PROTSYM file. This name is used by the CA Application Quality and Testing Tools products to locate the symbolic information and is displayed when listing the contents of your PROTSYM. In most cases, this name should be the same as the PROGRAM-ID. However, when loading symbolic information for use with CA InterTest for CICS, specify the name of the CICS program definition, or for composite support, specify the monitor name.
LISTLIB	Specifies the name of the partitioned data set containing the listing from the OS/VS COBOL compiler or CA Optimizer.
MEMBER	Specifies the name of the member in the listing library that contains the compiler listing for the program being added.
LISTER	Specifies how much of the listing to write to the OUTPUT file.
CUTPRINT	Specifies how much of the listing to write to the OUTPUT file.

Adding IN25SYMC to Your OS/VS COBOL Procedure

To automatically update the symbolic information in your PROTSYM file whenever a OS/VS COBOL program is compiled, you can add a postprocessor step directly to the JCL procedure that you use to compile your programs. These same steps also apply to your CA Optimizer procedure.

Follow these steps to update your existing compile procedure:

1. Ensure that your compile step specifies all of the required OS/VS COBOL options.
2. Change the DD statement so that a temporary disk file is created for your listing, if the SYSPRINT output from your compile step is written to SYSOUT.
3. Add a new IN25PARM step following your compile step to generate the parameter statement for the postprocessor.
4. Add a new IN25SYMC step to postprocess the listing from the compile step. The INPUT DD on this step refers to the same file as the SYSPRINT DD from the compile step.
5. Add a new IEBGENER step to print the compiler listing only if the compiler detects errors.

The following example shows modifications to a compile procedure:

```
//COB      EXEC PGM=IKFCBL00,REGION=4M,
//  PARM='SOURCE,DMAP,SXREF,PMAP,VERB,NONUM,&OPTIONS'    <= 1

     (Your existing DD statements for OS/VS COBOL)

//SYSPRINT DD DSN=&&LST,DISP=(NEW,PASS),                 <= 2
//            UNIT=SYSDA,SPACE=(CYL,(1,2))
//*
//*   GENERATE THE PARAMETER STATEMENT FOR IN25SYMC
//*
//CARDS   EXEC PGM=IN25PARM,REGION=1M,COND=(4,LT),       <= 3
//   PARM='&MEMBER,LISTER=ALL'
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//CARDS    DD DSN=&&CARDS,DISP=(NEW,PASS),
//            UNIT=SYSDA,SPACE=(TRK,(1,1))
//*
//*    POST-PROCESS THE COMPILER LISTING
//*
//SYM      EXEC PGM=IN25SYMC,REGION=4M,COND=(4,LT)       <= 4
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//PROTSYM  DD DSN=USER.PROTSYM,DISP=SHR
//OUTPUT   DD SYSOUT=*,
//            DCB=(LRECL=121,BLKSIZE=2420,RECFM=FBA)
//INPUT    DD DSN=&&LST,DISP=(OLD,PASS)                      (See Note 1)
//CARDS    DD DSN=&&CARDS,DISP=(OLD,DELETE)                  (See Note 2)
//MESSAGE  DD SYSOUT=*
//*
//PRINT    EXEC PGM=IEBGENER,COND=(5,GT,COB)         <= 5
//SYSUT1   DD DSN=&&LST,DISP=(OLD,DELETE)
//SYSUT2   DD SYSOUT=*
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
```

Notes:

- If the SYSPRINT DD on your compile step refers to a permanent data set, the INPUT DD for IN25SYMC must point to the same data set.
- If you prefer to pass your parameter statement as an override in the invoking JCL, delete the CARDS step, delete this DD statement, and add SYM.CARDS DD to your invoking JCL member.

IN25COB2

Use program IN25COB2 to load symbolic information for programs compiled using any of the following products:

- Enterprise COBOL for z/OS
- IBM COBOL for VM
- IBM COBOL for MVS and VM
- AD/CYCLE COBOL/370

- VS COBOL II
- CA Optimizer/II

Note: In this article, the term COBOL refers to any of the COBOL dialects supported by the IBM compilers listed previously.

Execute IN25COB2 as a standalone batch job to load a single COBOL listing that has been previously saved to a permanent file, or add it to your existing COBOL or CA Optimizer/II JCL procedure. The method you select depends entirely on the procedures at your own installation. Both methods are described in this article.

IN25COB2 JCL

The following table describes the DD statements used by IN25COB2:

DDname	Description
STEPLIB	The load library containing IN25COB2.
INPUT	The listing that was written to SYSPRINT by the COBOL compiler, or by CA Optimizer/II, during compilation.
OUTPUT	All or part of the original compiler listing is written to this file, depending on your request.
MESSAGE	All messages produced by IN25COB2 during post processing are written to this file.
PROTSYM	The file to which the symbolic information is written.
CARDS	The input control statements that define the request.

Note: If you are adding a new step for IN25COB2 to a JCL procedure, use program IN25PARM to write your input control statements to the CARDS file.

IN25COB2 Options

Options are passed to IN25COB2 using a parameter statement in the CARDS DD. Specify the parameter statement as an in-stream control card, or when using a JCL procedure, generate it using program IN25PARM as follows:

```
//IN25PARM EXEC PGM=IN25PARM,PARM='parameter statement'
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//CARDS    DD DISP=(,PASS),DSN=&&CARDS,UNIT=SYSDA,SPACE=(TRK,(1,1))
```

Parameter statements in the CARDS DD must begin in column 1.

The program name is the only required field on the parameter statement. This positional parameter defines the name that is used to store the symbolic information in the PROTSYM file. This name is used by the CA Application Quality and Testing Tools products to locate the symbolic information and is displayed when listing the contents of your PROTSYM.

In most cases, this name should be the same as the PROGRAM-ID. However, when loading symbolic information for use with CA InterTest for CICS, you must specify the name of the CICS program definition, or when using composite support, specify the monitor name.

The following example shows an in-stream parameter statement that can be used to save symbolic information using the name ORDEDIT:

```
//CARDS  DD *
ORDEDIT
/*
```

Controlling Printed Output with the CUTPRINT Option

Because you can load symbolic information from a permanent data set or a temporary listing file, you can also print all or part of the listing generated by the compiler.

Append the CUTPRINT option to your parameter statement to control printing of the compiler listing as follows:

- **,CUTPRINT=ALL**
  Do not print any of the compiler listing.
- **,CUTPRINT=MAP**
  Print the listing up to, but not including, the Data Division Map report.
- **,CUTPRINT=REF**
  Print the listing up to, but not including, the cross reference of data names.

The following sample parameter statement saves symbolic information for program ORDEDIT and prints only the source code section of the compiler listing:

```
//CARDS   DD *
ORDEDIT,CUTPRINT=REF
/*
```

Note: Specify the CUTPRINT parameter only when you do not want all or part of your listing printed. The entire listing is printed if this parameter is omitted.

Saving Your Listing for Online Display with the LISTER Option

Append the LISTER option to your parameter statement to control which portion of your source listing is saved to the PROTSYM file, as follows:

- **,LISTER=ALL**
  Saves the entire COBOL listing.
- **,LISTER=MAP**
  Saves the COBOL listing up to, but not including, the Data Division map report.
- **,LISTER=REF**
  Saves the COBOL listing up to, but not including, the cross reference of data names.

The following sample parameter statement saves symbolic information for program ORDEDIT, does not print any of the listing, and saves the listing up to, but not including, the Data Division map report to the PROTSYM file:

```
//CARDS DD  *
ORDEDIT,CUTPRINT=ALL,LISTER=MAP
/*
```

⚠ **Notes:**

- If the LISTER parameter is omitted, no listing is saved in the symbolic file.
- The LISTER parameter is required for use with CA Optimizer/II, CA SymDump Batch, and CA InterTest Batch.
- To reduce overhead and save space in your PROTSYM file, we recommend that you specify LISTER=MAP when executing IN25COB2 unless compiling with Optimizer/II, which requires LISTER=MMAP.

Setting Data as Nonpurgeable

You can mark any saved symbolic data for this program as nonpurgeable. If a program's data is marked as nonpurgeable, the data is not removed from the PROTSYM when deleting programs using a purge interval batch run. However, you can delete the data by program name. See Maintaining a PROTSYM File (see page 53) for instructions on deleting data from the symbolic file.

To mark data as nonpurgeable, add the NOPURGE option to your parameter statement as the last option.

The following sample parameter statement saves symbolic information for program ORDEDIT, prints the entire listing, saves the entire listing in the PROTSYM file, and does not let symbolic data be removed from the symbolic file by a purge interval batch run.

```
//CARDS DD  *
ORDEDIT,LISTER=ALL,NOPURGE
/*
```

Required COBOL Options

The following compiler options are required to load symbolic information for COBOL programs into the PROTSYM file:

Option	Description
MAP	Produces a Data Division map.
NONUMBER	Suppresses compiler-generated line numbers.
OFFSET or LIST*	Produces a condensed Procedure map or full Assembler Procedure map.
XREF	Produces a cross-reference of data and procedure names.
NOPT or OPT (0)**	Produces breakpoints synchronized with source.
NOSTGOPT	Prevents the compiler from discarding unreferenced data items. The NOSTGOPT option is only valid for COBOL 5.1 and above.

* The LIST option is required when using the integrated CICS translator or integrated SQL coprocessor of COBOL for z/OS.

** When a COBOL program is OPTIMIZED, your breakpoints may not get stopped exactly where you think they should because the optimization is adding or modifying the generated code, and it may not be synchronized with the related source statements in the listing.

The following compiler options are required to load symbolic information for a program compiled using CA Optimizer/II into the PROTSYM file:

Option	Description
INTERTST	Required only when optimizing programs that are monitored using CA InterTest for CICS.
MAP or MMAP	Produces a Data Division map or merged Data Division map.
MMAP	Required when optimizing programs that are monitored using CA InterTest for CICS.
MOFFSET	Produces a merged Procedure map.
NONUM	Suppresses compiler-generated line numbers.
XREF	Produces a cross-reference of data and paragraph names.

⚠

Note: If you are using CA Optimizer/II r7 or higher, you can use the SYM compile-time option to automatically load symbolic information into your PROTSYM file during optimization. When using the SYM option, add a PROTSYM DD statement to your compile /optimize step. No additional option requirements exist when using this method.

To use symbolic references in COBOL, you must declare at least one data item in working storage.

Executing IN25COB2 as a Standalone Program

Member CAVHCOB2 in CAI.CAVHPROC contains sample JCL for executing postprocessor IN25COB2 as a standalone batch job. Use this member to load symbolic information from previously saved COBOL listings.

```
//CAVHCOB2 PROC PROTSYM=CAI.PROTSYM,
//             NAME=XXXXXXXX,
//             LISTLIB=USER.LISTLIB,
//             MEMBER=XXXXXXXX,
//             LISTER=ALL,
//             CUTPRINT=ALL
//*
//IN25PARM EXEC PGM=IN25PARM,REGION=512K,
//             PARM='&MEMBER,LISTER=&LISTER,CUTPRINT=&CUTPRINT'
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//CARDS    DD DSN=&&CARDS,DISP=(,PASS),
//            UNIT=SYSDA,SPACE=(TRK,(1,1))
//*
//IN25COB2 EXEC PGM=IN25COB2,REGION=2M
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//PROTSYM  DD DISP=SHR,DSN=&PROTSYM
//INPUT    DD DISP=SHR,DSN=&LISTLIB(&MEMBER)
//CARDS    DD DSN=&&CARDS,DISP=(OLD,DELETE)
//OUTPUT   DD SYSOUT=*,DCB=(LRECL=133,BLKSIZE=3990,RECFM=FBA)
//MESSAGE  DD SYSOUT=*
//
```

You can override the following procedure variables:

Variable	Description
PROTSYM	Specifies the name of the symbolic file being updated.
NAME	Specifies the name that is used to store the symbolic information in the PROTSYM file. This name is used by the CA Application Quality and Testing Tools products to locate the symbolic information and is displayed when listing the contents of your PROTSYM. In most cases, this name should be the same as the PROGRAM-ID. However, when loading symbolic information for use with CA InterTest for CICS, specify the name of the CICS program definition, or for composite support, specify the monitor name.
LISTLIB	Specifies the name of the partitioned data set containing the listing from the COBOL II compiler or CA Optimizer/II.
MEMBER	Specifies the name of the member in the listing library that contains the compiler listing for the program being added.
LISTER	Specifies how much of the listing to save in the PROTSYM file.
CUTPRINT	Specifies how much of the listing to write to the OUTPUT file.

Adding IN25COB2 to Your COBOL Procedure

To automatically update the symbolic information in your PROTSYM file whenever a COBOL program is compiled, you can add a postprocessor step directly to the JCL procedure that you use to compile your programs.

Note: These same steps also apply to the CA Optimizer/II procedure.

Follow these steps to update your existing compile procedure:

1. Ensure that your compile step specifies all of the required COBOL options.
2. Change the DD statement so that a temporary disk file is created for your listing, if the SYSPRINT output from your compile step is written to SYSOUT.
3. Add a new IN25PARM step following your compile step to generate the parameter statement for the postprocessor.
4. Add a new IN25COB2 step to postprocess the listing from the compile step. The INPUT DD on this step refers to the same file as the SYSPRINT DD from the compile step.
5. Add a new IEBGENER step to print the compiler listing only if the compiler detects errors.

The following example shows modifications to a compile procedure:

```
//COB      EXEC PGM=IGYCRCTL,REGION=4M,
//  PARM='S,MAP,X,LIST,NONUM,&OPTIONS'                   <= 1

     (Your existing DD statements for COBOL II)

//SYSPRINT DD DSN=&&LST,DISP=(NEW,PASS),                 <= 2
//            UNIT=SYSDA,SPACE=(CYL,(1,2))
//*
//*   GENERATE THE PARAMETER STATEMENT FOR IN25COB2
//*
//CARDS   EXEC PGM=IN25PARM,REGION=1M,COND=(4,LT),       <= 3
//   PARM='&MEMBER,LISTER=ALL'
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//CARDS    DD DSN=&&CARDS,DISP=(NEW,PASS),
//            UNIT=SYSDA,SPACE=(TRK,(1,1))
//*
//*    POST-PROCESS THE COMPILER LISTING
//*
//SYM      EXEC PGM=IN25COB2,REGION=4M,COND=(4,LT)       <= 4
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//PROTSYM  DD DSN=USER.PROTSYM,DISP=SHR
//OUTPUT   DD SYSOUT=*,
//            DCB=(LRECL=133,BLKSIZE=3990,RECFM=FBA)
//INPUT    DD DSN=&&LST,DISP=(OLD,PASS)                      (See Note 1)
//CARDS    DD DSN=&&CARDS,DISP=(OLD,DELETE)                  (See Note 2)
//MESSAGE  DD SYSOUT=*
//*
//PRINT    EXEC PGM=IEBGENER,COND=(5,GT,COB)         <= 5
//SYSUT1   DD DSN=&&LST,DISP=(OLD,DELETE)
//SYSUT2   DD SYSOUT=*
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
```

Notes:

- If the SYSPRINT DD on your compile step refers to a permanent data set, the INPUT DD for IN25COB2 must point to the same data set.

- If you prefer to pass your parameter statement as an override in the invoking JCL, delete the CARDS step, delete this DD statement, and add SYM.CARDS DD to your invoking JCL member.

Controlling Printed Output with the CUTPRINT Option

Because you can load symbolic information from a permanent data set or a temporary listing file, you can also print all or part of the listing generated by the compiler.

Append the CUTPRINT option to your parameter statement to control printing of the compiler listing as follows:

- **,CUTPRINT=ALL**
  Do not print any of the compiler listing.
- **,CUTPRINT=REF**
  Print the listing up to, but not including, the cross reference of data names.

The following sample parameter statement saves symbolic information for program ORDEDIT and prints only the source code section of the compiler listing:

```
//CARDS    DD *
ORDEDIT,CUTPRINT=REF
/*
```

Note: Specify the CUTPRINT parameter only when you do not want all or part of your listing printed. The entire listing is printed if this parameter is omitted.

Saving Your Listing for Online Display with the LISTER Option

Append the LISTER option to your parameter statement to control which portion of your source listing is saved to the PROTSYM file, as follows:

- **,LISTER=ALL**
  Saves the entire compiler listing.
- **,LISTER=REF**
  Saves the compiler listing up to, but not including, the cross reference of data names.

The following sample parameter statement saves symbolic information for program ORDEDIT, does not print any of the listing, and saves the listing up to, but not including, the Data Division map report to the PROTSYM file:

```
//CARDS DD  *
ORDEDIT,CUTPRINT=ALL,LISTER=MAP
/*
```

Notes:

- If the LISTER parameter is omitted, no listing is saved in the symbolic file.
- The LISTER parameter is required for use with CA Optimizer/II, CA SymDump Batch, and CA InterTest Batch.
- To reduce overhead and save space in your PROTSYM file, we recommend that you specify LISTER=MAP when executing IN25CPPR unless compiling with Optimizer/II, which requires LISTER=MMAP.

Setting Data as Nonpurgeable

You can mark any saved symbolic data for this program as nonpurgeable. If a program's data is marked as nonpurgeable, the data is not removed from the PROTSYM when deleting programs using a purge interval batch run. However, you can delete the data by program name. See Maintaining a PROTSYM File (see page 53) for instructions on deleting data from the symbolic file.

To mark data as nonpurgeable, add the NOPURGE option to your parameter statement as the last option.

The following sample parameter statement saves symbolic information for program ORDEDIT, prints the entire listing, saves the entire listing in the PROTSYM file, and does not let symbolic data be removed from the symbolic file by a purge interval batch run.

```
//CARDS DD  *
ORDEDIT,LISTER=ALL,NOPURGE
/*
```

IN25SYMP

Use program IN25SYMP to load symbolic information for programs compiled using any of the following products:

- Enterprise PL/I for z/OS
- IBM PL/I for MVS and VM
- Visual Age PL/I
- OS PL/I

Note: In this section, the term PL/I refers to any of the PL/I dialects supported by the IBM compilers previously listed.

Execute IN25SYMP as a standalone batch job to load a single PL/I listing that has been previously saved to a permanent file, or add it to your existing PL/I JCL procedure. The method you select depends entirely on the procedures at your own installation. Both methods are described in this article.

IN25SYMP JCL

The following table describes the DD statements used by IN25SYMP:

DDname	Description
STEPLIB	The load library containing IN25SYMP.
INPUTT	The listing that was written to SYSPRINT by the PL/I compiler during compilation.
SYSPRINT	All or part of the original compiler listing is written to this file, depending on your request.
MESSAGE and MSGS	Messages produced by IN25SYMP during postprocessing are written to these files.
PROTSYM	The file to which the symbolic information is written.
CARDS	The input control statements that define the request.

Note: If you are adding a new step for IN25SYMP to a JCL procedure, use program IN25PARM to write your input control statements to the CARDS file.

IN25SYMP Options

Options are passed to IN25SYMP using a parameter statement in the CARDS DD. Specify the parameter statement as an in-stream control card, or when using a JCL procedure, generate it using program IN25PARM as follows:

```
//IN25PARM EXEC PGM=IN25PARM,PARM='parameter statement'
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//CARDS    DD DISP=(,PASS),DSN=&&CARDS,UNIT=SYSDA,SPACE=(TRK,(1,1))
```

Parameter statements in the CARDS DD must begin in column 1.

The program name is the only required field on the parameter statement. This positional parameter defines the name that is used to store the symbolic information in the PROTSYM file. This name is used by the CA Application Quality and Testing Tools products to locate the symbolic information and is displayed when listing the contents of your PROTSYM.

When loading symbolic information for use with CA InterTest for CICS, you must specify the name of the CICS program definition, or when using composite support, specify the monitor name.

The following example shows an in-stream parameter statement that you can use to save symbolic information using the name ORDEDIT:

```
//CARDS  DD *
ORDEDIT
/*
```

Controlling Printed Output with the CUTPRINT Option

Because you can load symbolic information from a permanent data set or a temporary listing file, you can also print all or part of the listing generated by the compiler.

Append the CUTPRINT option to your parameter statement to control printing of the compiler listing as follows:

- **,CUTPRINT=ALL**
  Prints none of the compile listing.
- **,CUTPRINT=REF**
  Terminates printing after the XREF table.
- **,CUTPRINT=OFFSET**
  Terminates printing after the table of offsets.

The following sample parameter statement saves symbolic information for program ORDEDIT and terminates printing after the XREF table:

```
//CARDS   DD *
ORDEDIT,CUTPRINT=REF
/*
```

Note: Specify the CUTPRINT parameter only when you do not want all or part of your listing printed. The entire listing is printed if you omit this parameter.

Saving Your Listing for Online Display with the LISTER Option

Append the LISTER option to your parameter statement to control which portion of your source listing is saved to the PROTSYM file, as follows:

- **,LISTER=ALL**
  Saves the entire PL/I listing.
- **,LISTER=REF**
  Saves only the source and XREF sections.
- **,LISTER=OFFSET**
  Saves the listing up to, and including, the table of offsets.

The following sample parameter statement saves symbolic information for program ORDEDIT, does not print any of the listing, and saves only the source and XREF sections of the listing:

```
//CARDS DD  *
ORDEDIT,CUTPRINT=ALL,LISTER=REF
/*
```

Notes:

- If the LISTER parameter is omitted, no listing is saved in the symbolic file.
- The LISTER parameter is required for use with CA Optimizer/II, CA SymDump Batch, and CA InterTest Batch.

Setting Data as Nonpurgeable

You can mark any saved symbolic data for this program as nonpurgeable. If a program's data is marked as nonpurgeable, the data is not removed from the PROTSYM when deleting programs using a purge interval batch run. However, you can delete the data by program name. See Maintaining a PROTSYM File (see page 53) for instructions about deleting data from the symbolic file.

To mark data as nonpurgeable, add the NOPURGE option to your parameter statement as the last option.

The following sample parameter statement saves symbolic information for program ORDEDIT, prints the entire listing, saves the entire listing in the PROTSYM file, and does not let symbolic data be removed from the symbolic file by a purge interval batch run.

```
//CARDS DD  *
ORDEDIT,LISTER=ALL,NOPURGE
/*
```

Required PL/I Options

The following compiler options are required to load symbolic information for PL/I programs when using the OS PL/I or IBM PL/I for MVS and VM compiler:

AGGREGATE
ATTRIBUTES(FULL)
MAP
NEST
NOGONUM
NONUMBER
OPTIONS
SOURCE
STMT or GOSTMT
STORAGE
XREF(FULL)

The following compiler options are required to load symbolic information for PL/I programs when using Enterprise PL/I for z/OS or Visual Age PL/I:

AGGREGATE
ATTRIBUTES(FULL)
LIMITS(NAME(31))
NOGONUM
NOSTMT
NUMBER

LIST	OPTIONS
MAP	SOURCE
NATLANG(ENU)	STORAGE
NEST	XREF(FULL)

⚠ **Notes:**

- Because of special considerations, if you must use the %NOPRINT compiler option, contact CA Support.
- For the CA Application Quality and Testing Tools products to support date/time stamp comparison between your symbolic information and your executables, you must select TSTAMP=YES when installing your PL/I compiler.
- CA InterTest Batch, CA SymDump Batch, and CA Optimizer/II display only controlled variables.
- When using the IBM PL/I for MVS and VM compiler, the ESD option is required for programs that have controlled variables.

Executing IN25SYMP as a Standalone Program

Member CAVHSYMP in CAI.CAVHPROC contains sample JCL for executing postprocessor IN25SYMP as a standalone batch job. Use this member to load symbolic information from previously saved PL/I listings as follows:

```
//CAVHSYMP PROC PROTSYM=CAI.PROTSYM,
//              NAME=XXXXXXXX,
//              LISTLIB=USER.LISTLIB,
//              MEMBER=XXXXXXXX,
//              LISTER=ALL,
//              CUTPRINT=ALL
//*
//IN25PARM EXEC PGM=IN25PARM,REGION=512K,
//              PARM='&MEMBER,LISTER=&LISTER,CUTPRINT=&CUTPRINT'
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//CARDS    DD DSN=&&CARDS,DISP=(,PASS),
//             UNIT=SYSDA,SPACE=(TRK,(1,1))
//*
//IN25SYMP EXEC PGM=IN25SYMP,REGION=2M
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//PROTSYM  DD DISP=SHR,DSN=&PROTSYM
//INPUTT   DD DISP=SHR,DSN=&LISTLIB(&MEMBER)
//CARDS    DD DSN=&&CARDS,DISP=(OLD,DELETE)
//SYSPRINT   DD SYSOUT=*
//MESSAGE  DD SYSOUT=*
//MSGS     DD SYSOUT=*
//
```

You can override the following procedure variables:

Variable	Description
PROTSYM	Specifies the name of the symbolic file being updated.

Variable	Description
NAME	Specifies the name that is used to store the symbolic information in the PROTSYM file. This name is used by the CA Application Quality and Testing Tools products to locate the symbolic information and is displayed when listing the contents of your PROTSYM. When loading symbolic information for use with CA InterTest for CICS, specify the name of the CICS program definition, or for composite support, specify the monitor name.
LISTLIB	Specifies the name of the partitioned data set containing the listing from the PL/I compiler.
MEMBER	Specifies the name of the member in the listing library that contains the compiler listing for the program being added.
LISTER	Specifies how much of the listing to save in the PROTSYM file.
CUTPRINT	Specifies how much of the listing to write to the OUTPUT file.

Adding IN25SYMP to Your PL/I Procedure

To automatically update the symbolic information in your PROTSYM file whenever a PL/I program is compiled, add a postprocessor step directly to the JCL procedure you use to compile your programs.

Follow these steps to update your existing compile procedure:

1. Ensure that your compile step specifies all of the required PL/I options.
2. Change the DD statement so that a temporary disk file is created for your listing, if the SYSPRINT output from your compile step is written to SYSOUT.
3. Add a new IN25PARM step following your compile step to generate the parameter statement for the postprocessor.
4. Add a new IN25SYMP step to postprocess the listing from the compile step. The INPUTT DD on this step refers to the same file as the SYSPRINT DD from the compile step.
5. Add a new IEBGENER step to print the compiler listing only if the compiler detects errors.

The following example shows modifications to a compile procedure:

```
//PLI      EXEC PGM=IBMZPLI,REGION=4M,
//  PARM=('OBJ,X(F),A(F),OP,MAP,STG,AG,NEST,LIST',       <= 1
//        'NIS,S,NOPT,LIMITS(NAME(31)),FLAG(W)')

     (Your existing DD statements for PL/I)

//SYSPRINT DD DSN=&&LST,DISP=(NEW,PASS),                  <= 2
//            UNIT=SYSDA,SPACE=(CYL,(1,2))
//*
//*   GENERATE THE PARAMETER STATEMENT FOR IN25SYMP
//*
//CARDS   EXEC PGM=IN25PARM,REGION=1M,COND=(4,LT),       <= 3
//   PARM='&MEMBER,LISTER=ALL'
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//CARDS    DD DSN=&&CARDS,DISP=(NEW,PASS),
//            UNIT=SYSDA,SPACE=(TRK,(1,1))
//*
//*    POST-PROCESS THE COMPILER LISTING
```

```
//*
//SYM      EXEC PGM=IN25SYMP,REGION=4M,COND=(4,LT)       <= 4
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//PROTSYM  DD DSN=USER.PROTSYM,DISP=SHR
//SYSPRINT DD SYSOUT=*
//INPUTT   DD DSN=&&LST,DISP=(OLD,PASS)                        (See Note 1)
//CARDS    DD DSN=&&CARDS,DISP=(OLD,DELETE)                    (See Note 2)
//MESSAGE  DD SYSOUT=*
//MSGS     DD SYSOUT=*
//*
//PRINT    EXEC PGM=IEBGENER,COND=(5,GT,PLI)         <= 5
//SYSUT1   DD DSN=&&LST,DISP=(OLD,DELETE)
//SYSUT2   DD SYSOUT=*
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
```

Notes:

- If the SYSPRINT DD on your compile step refers to a permanent data set, the INPUTT DD for IN25SYMP must point to the same data set.
- If you prefer to pass your parameter statement as an override in the invoking JCL, delete the CARDS step, delete this DD statement, and add SYM.CARDS DD to your invoking JCL member.

IN25SYMA

Use program IN25SYMA to load symbolic information for Assembler programs into your PROTSYM file.

Execute IN25SYMA as a standalone batch job to load a single Assembler listing that has been previously saved to a permanent file, or add it to your existing Assembler JCL procedure. The method you select depends entirely on the procedures at your own installation. Both methods are described in this article.

IN25SYMA JCL

The following table describes the DD statements used by IN25SYMA:

DDname	Description
STEPLIB	The load library containing IN25SYMA.
INPUT	The listing that was written to SYSPRINT by the Assembler.

DDname	Description
OUTPUT	All or part of the original Assembler listing is written to this file, depending on your request.
MESSAGE	Messages produced by IN25SYMA during postprocessing are written to this file.
PROTSYM	The file to which the symbolic information is written.
CARDS	The input control statements that define the request.

⚠ **Note:** If you are adding a new step for IN25SYMA to a JCL procedure, use program IN25PARM to write your input control statements to the CARDS file.

IN25SYMA Options

Options are passed to IN25SYMA using a parameter statement in the CARDS DD. Specify the parameter statement as an in-stream control card, or when using a JCL procedure, generate it using program IN25PARM as follows:

```
//IN25PARM EXEC PGM=IN25PARM,PARM='parameter statement'
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//CARDS    DD DISP=(,PASS),DSN=&&CARDS,UNIT=SYSDA,SPACE=(TRK,(1,1))
```

Parameter statements in the CARDS DD must begin in column 1.

The program name is the only required field on the parameter statement. This positional parameter defines the name that is used to store the symbolic information in the PROTSYM file. This name is used by the CA Application Quality and Testing Tools products to locate the symbolic information and is displayed when listing the contents of your PROTSYM.

In most cases, this name should be the same as the first CSECT in the Assembler listing. When loading symbolic information for use with CA InterTest for CICS, you must specify the name of the CICS program definition, or when using composite support, specify the monitor name.

The following example shows an in-stream parameter statement that can be used to save symbolic information using the name ORDEDIT:

```
//CARDS  DD *
ORDEDIT
/*
```

Controlling Printed Output with the CUTPRINT Option

Because you can load symbolic information from a permanent data set or a temporary listing file, you can also print all or part of the listing generated by the assembler.

Append the CUTPRINT option to your parameter statement to control printing of the assembler listing as follows:

- **,CUTPRINT=ALL**
  Do not print any of the assembler listing.

- **,CUTPRINT=REF**
  Stops printing the listing after the Cross Reference report.

The following sample parameter statement saves symbolic information for program ORDEDIT without printing any of the listing:

```
//CARDS   DD *
ORDEDIT,CUTPRINT=ALL
/*
```

Note: Specify the CUTPRINT parameter only when you do not want all or part of your listing printed. The entire listing is printed if this parameter is omitted.

Saving Your Listing for Online Display with the LISTER Option

Append the LISTER option to your parameter statement to control which portion of your source listing is saved to the PROTSYM file as follows:

- **,LISTER=ALL**
  Saves the entire assembler listing.

- **,LISTER=REF**
  Saves the listing up to, but not including, the Cross Reference report.

The following sample parameter statement saves symbolic information for program ORDEDIT while printing and saving the listing up to, but not including, the Cross Reference report:

```
//CARDS DD  *
ORDEDIT,CUTPRINT=REF,LISTER=REF
/*
```

Notes:

- If the LISTER parameter is omitted, no listing is saved in the symbolic file.
- The LISTER parameter is required for use with CA Optimizer/II, CA SymDump Batch, and CA InterTest Batch.

Setting Data as Nonpurgeable

You can mark any saved symbolic data for this program as nonpurgeable. If a program's data is marked as nonpurgeable, the data is not removed from the PROTSYM when deleting programs using a purge interval batch run. However, you can delete the data by program name. See Maintaining a PROTSYM File (see page 53) for instructions on deleting data from the symbolic file.

To mark data as nonpurgeable, add the NOPURGE option to your parameter statement as the last option.

The following sample parameter statement saves symbolic information for program ORDEDIT, prints the entire listing, saves the entire listing in the PROTSYM file, and does not let symbolic data be removed from the symbolic file by a purge interval batch run.

```
//CARDS DD   *
ORDEDIT,LISTER=ALL,NOPURGE
/*
```

Required Assembler Options

The following listing options are required to load symbolic information for assembler programs:

Option	Description
DXREF	DSECT Cross-Reference
ESD	External Symbol Dictionary
NOBATCH	Only one assembler source program is in the input source file
USING	Using Map report
XREF(FULL) or XREF(SHORT)	Full cross-reference or cross-reference of referenced names

⚠ **Notes:**

- Do not suppress statements that define the start of a DSECT in the listing by PRINT OFF or PRINT NOGEN.
- High Level Assembler r2.0 users must also specify the LIST(121) option.
- High Level Assembler r4.0 users must specify the THREAD option. The NOTHREAD option is not supported.

Executing IN25SYMA as a Standalone Program

Member CAVHSYMA in CAI.CAVHPROC contains sample JCL for executing postprocessor IN25SYMA as a standalone batch job. Use this member to load symbolic information from previously saved Assembler listings.

```
//CAVHSYMA PROC PROTSYM=CAI.PROTSYM,
//              NAME=XXXXXXXX,
//              LISTLIB=USER.LISTLIB,
//              MEMBER=XXXXXXXX,
//              LISTER=ALL,
//              CUTPRINT=ALL
//*
//IN25PARM EXEC PGM=IN25PARM,REGION=512K,
//              PARM='&MEMBER,LISTER=&LISTER,CUTPRINT=&CUTPRINT'
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//CARDS    DD DSN=&&CARDS,DISP=(,PASS),
//            UNIT=SYSDA,SPACE=(TRK,(1,1))
//*
//IN25SYMA EXEC PGM=IN25SYMA,REGION=2M
```

```
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//PROTSYM  DD DISP=SHR,DSN=&PROTSYM
//INPUT    DD DISP=SHR,DSN=&LISTLIB(&MEMBER)
//CARDS    DD DSN=&&CARDS,DISP=(OLD,DELETE)
//OUTPUT   DD SYSOUT=*, DCB=(RECFM=FBM,LRECL=121,BLKSIZE=2420)
//MESSAGE  DD SYSOUT=*
//
```

You can override the following procedure variables:

Variable	Description
PROTSYM	Specifies the name of the symbolic file being updated.
NAME	Specifies the name that is used to store the symbolic information in the PROTSYM file. This name is used by the CA Application Quality and Testing Tools products to locate the symbolic information and is displayed when listing the contents of your PROTSYM. In most cases, this is the name of the first CSECT in the assembler listing. When loading symbolic information for use with CA InterTest for CICS, specify the name of the CICS program definition, or for composite support, specify the monitor name.
LISTLIB	Specifies the name of the partitioned data set containing the Assembler listing.
MEMBER	Specifies the name of the member in the listing library that contains the Assembler listing for the program being added.
LISTER	Specifies how much of the listing to save in the PROTSYM file.
CUTPRINT	Specifies how much of the listing to write to the OUTPUT file.

Adding IN25SYMA to Your Assembler Procedure

To automatically update the symbolic information in your PROTSYM file whenever a program is assembled, add a postprocessor step directly to the JCL procedure that you use to assemble your programs.

Follow these steps to update your existing assembly procedure:

1. Ensure that your assemble step specifies all of the required Assembler options.
2. If the SYSPRINT output from your assemble step is written to SYSOUT, change the DD statement so that a temporary disk file is created for your listing.
3. Add a new IN25PARM step following your assemble step to generate the parameter statement for the postprocessor.
4. Add a new IN25SYMA step to postprocess the listing from the assemble step. The INPUT DD on this step refers to the same file as the SYSPRINT DD from the assemble step.
5. Add a new IEBGENER step to print the Assembler listing only if errors were detected during the assembly.

The following example shows modifications to an assembly procedure:

```
//ASM      EXEC PGM=ASMA90,REGION=4M,
//  PARM='LIST,OBJECT,XREF(FULL),ESD'                    <= 1
```

```
     (Your existing DD statements for the Assembler)

//SYSPRINT DD DSN=&&LST,DISP=(NEW,PASS),                <= 2
//            UNIT=SYSDA,SPACE=(CYL,(1,2))
//*
//*   GENERATE THE PARAMETER STATEMENT FOR IN25SYMA
//*
//CARDS   EXEC PGM=IN25PARM,REGION=1M,COND=(4,LT),       <= 3
//   PARM='&MEMBER,LISTER=ALL'
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//CARDS    DD DSN=&&CARDS,DISP=(NEW,PASS),
//            UNIT=SYSDA,SPACE=(TRK,(1,1))
//*
//*    POST-PROCESS THE COMPILER LISTING
//*
//SYM      EXEC PGM=IN25SYMA,REGION=4M,COND=(4,LT)       <= 4
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//PROTSYM  DD DSN=USER.PROTSYM,DISP=SHR
//OUTPUT   DD SYSOUT=*,DCB=(RECFM=FBM,LRECL=121,BLKSIZE=2420)
//INPUT    DD DSN=&&LST,DISP=(OLD,PASS)                      (See Note 1)
//CARDS    DD DSN=&&CARDS,DISP=(OLD,DELETE)                  (See Note 2)
//MESSAGE  DD SYSOUT=*
//*
//PRINT    EXEC PGM=IEBGENER,COND=(5,GT,ASM)             <= 5
//SYSUT1   DD DSN=&&LST,DISP=(OLD,DELETE)
//SYSUT2   DD SYSOUT=*
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
```

Notes:

- If the SYSPRINT DD on your compile step refers to a permanent data set, the INPUT DD for IN25SYMA must point to the same data set.
- If you prefer to pass your parameter statement as an override in the invoking JCL, delete the CARDS step, delete this DD statement, and add SYM.CARDS DD to your invoking JCL member.

IN25LINK

Use program IN25LINK to define the additional symbolic information required for testing composite modules using CA InterTest for CICS.

A composite module consists of separately compiled or assembled parts that are brought together under a single module name by the IBM Linkage Editor. In CICS, a composite module has only one CICS program definition. You can write the main program and the called subroutines in the same or different languages.

The Composite Support feature of CA InterTest for CICS lets you test the subroutines of a composite module as if they were separate programs with separate CICS program definitions. For more information about this feature, see Composite Support (https://docops.ca.com/display/CAITSD11/Composite+Support).

IN25LINK uses output from the IBM Linkage Editor and your own additional input cards to associate the subroutines of a composite module with the monitor names you define. Typically, this program is executed as a standalone batch job to load output from a single link step.

If the Symbolic File in the JCL PROTSYM DD statement was initialized to allow multiple program version support, the IN25LINK control statement information will be associated with the latest or most recently compiled program version specified by the *monitor-name*.

> ⚠
>
> **Note:** While it is more efficient to use IN25LINK for composite support, you can provide the same information online using the CA InterTest for CICS COMPOSITE command or CNTL menu.

- IN25LINK JCL (see page 45)
- IN25LINK Options (see page 45)
- Required Linkage Editor Options (see page 48)
- Executing IN25LINK as a Standalone Program (see page 48)
- Adding IN25LINK to Your Link-Edit Procedure (see page 48)

IN25LINK JCL

The following table describes the DD statements used by IN25LINK:

DDname	Description
STEPLIB	The load library containing IN25LINK.
INPUT	The listing that was written to SYSPRINT by the IBM Linkage Editor.
OUTPUT	All or part of the original listing is written to this file, depending on your request.
MESSAGE	Messages produced by IN25LINK during postprocessing are written to this file.
PROTSYM	The file to which the symbolic information is written.
CARDS	The input control statements that define the relationships between your monitor names and the subroutines of your composite module.

IN25LINK Options

Options are passed to IN25LINK using parameter statements in the CARDS DD. Parameter statements in the CARDS DD must begin in column 1.

Identifying the Composite Module

The first parameter card is *required*. It specifies the CICS program definition name of the composite module, beginning in column 1, as follows:

```
composite-name[,NOPURGE]
```

The program name specified in this parameter statement must be the same as the CICS program definition name of the composite module.

You can also specify the NOPURGE option on this statement. This option specifies that symbolic information for the composite module *cannot* be purged from the PROTSYM file during a purge interval batch job.

Identifying the Main Program and Subroutines

Subsequent parameter cards optionally identify the main program and each subroutine that you want to test separately. These cards can be entered in any order and must specify two names, separated by commas, as follows:

```
link-name, monitor-name
```

link-name identifies the name of the control section as listed in the link-edit map, and must begin in column 1. For a PL/I program, specify the name of the compiler-generated control section that ends with '1'.

monitor-name specifies the name under which the program is monitored. Follow these rules in selecting a *monitor-name*:

- Each *monitor-name* must be unique.
- The *monitor-name* of a subroutine cannot be the same as a CICS program definition name.
- The *monitor-name* can be identical to the *link-name*.

Only the first parameter card is required; omit all subsequent cards. If you omit the subsequent parameter cards, only the name of the composite module is stored in the PROTSYM file. When you attempt to monitor the composite module with CA InterTest for CICS, you are prompted for additional composite information.

Excluding Subroutines

By default, IN25LINK excludes subroutines with CEE, DFH, DLZ, IBM, IGZ, and ILB prefixes when it reads the link-edit map. Usually, you will not want to test these programs.

You can change the default exclusion rules using a parameter card, positioned anywhere in the CARDS file after the first card, beginning in column 1, as follows:

```
EXCLUDE=
```

(or)

```
EXCLUDE=name[,name,…,name]
```

Specifying EXCLUDE without any names instructs IN25LINK not to exclude any subroutines.

Specifying EXCLUDE=*xxxxxxxx* instructs IN25LINK to exclude subroutines whose names are represented by *xxxxxxxx*. Specify the entire *link-name* to exclude a specific subroutine, or specify a prefix to exclude a group of subroutines.

For example, to exclude all subroutines with the prefix ACA1, specify:

```
EXCLUDE=ACA1
```

Example

A composite module with the CICS program definition name BIGMOD consists of several separately compiled programs and library modules. Its main program is named MAINMOD and is written in COBOL.

BIGMOD also has three Assembler subroutines named SUBMODA, SUBMODB, and SUBMODC that you want to test separately. None of the subroutines has its own CICS program definition entry.

After assembling SUBMODA, SUBMODB, and SUBMODC, you loaded their symbolic information by executing postprocessor IN25SYMA. You selected monitor names of ASMMODA, ASMMODB, and ASMMODC for the subroutines.

Next, you compiled MAINMOD and loaded its symbolic information by executing postprocessor IN25COB2. You specified BIGMOD as the monitor name, matching the CICS program definition as required.

The linkage editor combines the main program and its three subroutines, creating the composite load module named BIGMOD. The link-edit map output for BIGMOD shows a main entry address of 160, that BIGMOD has been replaced in the load library, and that it contains the following modules:

```
Control Section Origin  Length  Description
DFHECI      00  160 Command level COBOL stub
MAINMOD     160 78A8    Main program-COBOL
ILBOATB     7A08    11A COBOL library module
ILBOCOM0    7B28    73  COBOL library module
SUBMODA 7CA0    1200    Subprogram-Assembler
SUBMODB 8EA0    100 Subprogram-Assembler
SUBMODC 9EA1    93  Subprogram-Assembler
```

Sample JCL for the link-edit step and IN25LINK is shown as follows:

```
//*
//* Link Edit Step
//*
//LKED     EXEC PGM=IEWL,......
//SYSLIB   DD.....
//SYSLMOD  DD.....
//SYSUT1   DD.....
//SYSLIN   DD.....
//SYSPRINT DD DSN=&&INPUT,DISP=(,PASS),
//   UNIT=SYSDA,SPACE=(TRK,(2,5)),
//   DCB=(DSORG=PS,LRECL=121,BLKSIZE=2420,RECFM=FB)
//*
//* IN25LINK Step
//*
//POSTLINK EXEC PGM=IN25LINK,REGION=512K
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//INPUT    DD DSN=&&INPUT,DISP=(OLD,DELETE)
//MESSAGE  DD SYSOUT=*
//OUTPUT   DD SYSOUT=*,
//   DCB=(RECFM=FBA,LRECL=121,BLKSIZE=2420)
//PROTSYM  DD DSN=INTRTST.PROTSYM,DISP=SHR
//CARDS    DD *
BIGMOD             CICS program definition name of composite module
MAINMOD,BIGMOD     link-name and monitor-name of main program
SUBMODA,ASMMODA    link-name and monitor-name of subprogram
SUBMODB,ASMMODB    link-name and monitor-name of subprogram
SUBMODC,ASMMODC    link-name and monitor-name of subprogram
/*
```

Required Linkage Editor Options

If you are using DFSMS 1.1, you must specify the MAP parameter on the link-edit step.

Executing IN25LINK as a Standalone Program

Member CAVHLINK in CAI.CAVHPROC contains sample JCL for executing postprocessor IN25LINK as a standalone batch job. Use this member to load composite information from a previously saved Linkage Editor listing.

```
//CAVHLINK PROC PROTSYM=CAI.PROTSYM,
//              LISTLIB=USER.LISTLIB
//              MEMBER=XXXXXXXX
//*
//IN25LINK EXEC PGM=IN25LINK,REGION=2M
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//PROTSYM  DD DISP=SHR,DSN=&PROTSYM
//INPUT    DD DISP=SHR,DSN=&LISTLIB(&MEMBER)
//OUTPUT   DD SYSOUT=*,DCB=(LRECL=121,BLKSIZE=2440,RECFM=FBA)
//CARDS    DD DDNAME=CARDS
//MESSAGE  DD SYSOUT=*
```

You can override the following procedure variables:

Variable	Description
PROTSYM	Specifies the name of the symbolic file being updated.
LISTLIB	Specifies the name of the partitioned data set containing your saved IBM Linkage Editor output listings.
MEMBER	Specifies the name of the member in the listing library that contains the listing for the composite module.

When invoking this JCL procedure, you must include a CARDS DD statement in the invoking JCL.

Adding IN25LINK to Your Link-Edit Procedure

To automatically update the symbolic information in your PROTSYM file whenever a program is link-edited, you can add a postprocessor step directly to the JCL procedure that you use to link-edit your programs.

Follow these steps to update your existing link-edit procedure:

1. Ensure that your link-edit step specifies all of the required link-edit options.
2. If the SYSPRINT output from your link-edit step is written to SYSOUT, change the DD statement so that a temporary disk file is created for your listing.
3. Add a new IN25PARM step following your link-edit step to generate the parameter statement for the postprocessor.

4. Add a new IN25LINK step to postprocess the listing from the link-edit step. The INPUT DD on this step refers to the same file as the SYSPRINT DD from the link-edit step.

5. Add a new IEBGENER step to print the linkage editor listing only if errors were detected during the link.

The following example shows modifications to a link-edit procedure:

```
//LINK      EXEC PGM=IEWL,REGION=2M,
//  PARM='LIST,LET,XREF,MAP'                             <= 1

     (Your existing DD statements for the link edit)

//SYSPRINT DD DSN=&&LST,DISP=(NEW,PASS),                 <= 2
//            UNIT=SYSDA,SPACE=(CYL,(1,2))
//*
//*   GENERATE THE PARAMETER STATEMENT FOR IN25LINK
//*
//CARDS   EXEC PGM=IN25PARM,REGION=1M,COND=(4,LT),       <= 3
//   PARM='&MEMBER'
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//CARDS    DD DSN=&&CARDS,DISP=(NEW,PASS),
//            UNIT=SYSDA,SPACE=(TRK,(1,1))
//*
//*    POST-PROCESS THE LINK EDIT OUTPUT
//*
//SYM      EXEC PGM=IN25LINK,REGION=4M,COND=(4,LT)       <= 4
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//PROTSYM  DD DSN=USER.PROTSYM,DISP=SHR
//OUTPUT   DD SYSOUT=*,DCB=(RECFM=FBM,LRECL=121,BLKSIZE=2420)
//INPUT    DD DSN=&&LST,DISP=(OLD,PASS)                      (See Note 1)
//CARDS    DD DSN=&&CARDS,DISP=(OLD,DELETE)                  (See Note 2)
//MESSAGE  DD SYSOUT=*
//*
//PRINT    EXEC PGM=IEBGENER,COND=(5,GT,LINK)            <= 5
//SYSUT1   DD DSN=&&LST,DISP=(OLD,DELETE)
//SYSUT2   DD SYSOUT=*
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
```

Notes:

- If the SYSPRINT DD on your link-edit step refers to a permanent data set, the INPUT DD for IN25LINK must point to the same data set.
- If you prefer to pass your parameter statement as an override in the invoking JCL, delete the CARDS step, delete this DD statement, and add SYM.CARDS DD to your invoking JCL member.

Note: Using this method, IN25PARM only generates the first parameter card identifying the name of the composite module. No subsequent cards are generated. This serves only to identify the name of the composite module, which is stored in the PROTSYM file. When you attempt to monitor the composite module with CA InterTest for CICS, you may be prompted for additional composite information.

IN25SYMD

Use program IN25SYMD to load symbolic information from multiple COBOL, Assembler, C, or PL/I listings into your PROTSYM in a single run.

The following table describes the DD statements used by IN25SYMD:

DDname	Description
STEPLIB	The load library containing IN25SYMD.
PROTSYM	The file to which the symbolic information is written.
LISTLIB	The data set name of the PDS, PDSE, CA Librarian library, CA Panvalet library, or CA Endevor SCM library containing the listings to be added.
REPORT	An execution summary is written to this file.
OPTIN	The input control statements that define the request.

When you load symbolic information from CA Endevor SCM, ensure that the CA Endevor SCM AUTHLIB, CONLIB, and the data set containing C1DEFLTS are either in LINKLIST or in the STEPLIB concatenation. When you load symbolic information from CA Librarian or CA Panvalet, ensure that the CA Librarian or CA Panvalet CAILIB is either in Linklist or in the STEPLIB concatenation.

- IN25SYMD Options (see page 50)

IN25SYMD Options

Options are passed to the IN25SYMD using parameter statements in the OPTIN DD. Specify the parameter statements as an in-stream file.

Parameter statements contain one or more control statements, separated by commas, and each having the following syntax:

```
keyword=value
```

Specify the following option keywords to IN25SYMD:

Keyword	Description
LTYP	Identifies the library type of the listing library specified by the LISTLIB DD statement. This keyword is required. Valid values are: PDS -- Partitioned data set (including PDSE) SEQ -- Sequential data set (see Note) LIB -- CA Librarian library PAN -- CA Panvalet library

Keyword	Description
	NDV -- CA Endevor SCM library
FROM	Identifies the member name for single listings, the starting member name for a range of members, or a name prefix with trailing asterisk. This keyword is required.
TO	Identifies the last member name in a range of members.
MSG	Identifies the message reporting level. Valid values are: ALL -- displays all messages. RC -- displays a one-line return code message for each program. NONE -- suppresses all messages.

Note: When LTYP=SEQ is specified, the sequential file contains only one program listing. Specify the symbolic name using the FROM keyword, omit the TO keyword, and change the LISTLIB DD name to INPUT.

When using LTYP=NDV, you must also change the EXEC card to the following JCL:

```
//STEP1    EXEC PGM=NDVRC1,PARM='IN25SYMD',REGION=4M
```

Examples

This section contains postprocessor IN25SYMD examples.

Example 1

All of the programs with the prefix PAY are loaded into the PROTSYM file from a CA Librarian library, with all of the messages displayed in the REPORT file.

```
//STEP1     EXEC PGM=IN25SYMD,REGION=4M
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//PROTSYM  DD DISP=SHR,DSN=USER.PROTSYM
//LISTLIB  DD DISP=SHR,DSN=USER.LIBRARIAN.LIBRARY
//REPORT   DD SYSOUT=*
//OPTIN     DD *
 LTYP=LIB,FROM=PAY*,MSG=ALL
/*
```

Example 2

Program COBDEMO is loaded into the PROTSYM file from a sequential listing file, with messages suppressed. Note that the DD statement for the LISTLIB has been renamed to INPUT for LTYP=SEQ.

```
//STEP1     EXEC PGM=IN25SYMD,REGION=4M
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//PROTSYM  DD DISP=SHR,DSN=USER.PROTSYM
//INPUT     DD DISP=SHR,DSN=USER.SEQ.LISTING
//REPORT   DD SYSOUT=*
//OPTIN     DD *
 LTYP=SEQ,FROM=COBDEMO,MSG=NONE
/*
```

Example 3

Programs whose names begin with C, D, or E are loaded into the PROTSYM file from a partitioned data set. A one-line return code message is written to the REPORT file for each program.

```
//STEP1    EXEC PGM=IN25SYMD,REGION=4M
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//PROTSYM  DD DISP=SHR,DSN=USER.PROTSYM
//LISTLIB  DD DISP=SHR,DSN=USER.PDS.LIBRARY
//REPORT   DD SYSOUT=*
//OPTIN    DD *
 LTYP=PDS,FROM=C,TO=E9999999,MSG=RC
/*
```

Maintaining a PROTSYM File

The IN25UTIL batch utility program maintains and reports on the symbolic file. This program runs in batch, separate from the postprocessors that are used to load symbolic information into the symbolic file. This article describes how to maintain the PROTSYM file using IN25UTIL.

IN25UTIL JCL

Member CAVHUTIL in CAI.CAVHJCL contains the following sample JCL for executing IN25UTIL:

```
//          JOB
//IN25UTIL EXEC PGM=IN25UTIL,REGION=2M
//STEPLIB  DD DISP=SHR,DSN=CAI.CAVHLOAD
//PROTSYM  DD DISP=SHR,DSN=USER.PROTSYM
//OUTPUT   DD SYSOUT=*,DCB=(LRECL=133,BLKSIZE=3990)
//MESSAGE  DD SYSOUT=*
//CARDS    DD *
(input cards go here)

/*
```

The following table describes the DD statements used by IN25UTIL:

DDname	Description
STEPLIB	The load library containing IN25UTIL.
PROTSYM	The file on which maintenance or reporting is being performed.
MESSAGE	Output and messages from IN25UTIL are written to this file.
OUTPUT	Output from the PRINT function is written to this file.
UNLOAD	Program is written to this file by the UNLOAD function.
RELOAD	Program is read from this file by the RELOAD function.
CARDS	Contains the function request statements.

IN25UTIL Functions

IN25UTIL functions are requested using control statements in the CARDS file. All control statements in the CARDS file must begin in column 1.

⚠

Note: Some of the following functions support parameters that modify the function results. Supported parameters are noted in the function description. For complete parameter descriptions, see IN25UTIL Parameters (see page 58).

- **DELETE=*name***
  Deletes all symbolic data for the program specified by name.
  The PASSWORD control statement, if specified, must precede the DELETE control statement.
  You can use generic program names on the DELETE control statement. Use a trailing wildcard (*) in the program name to delete all matching programs.
  The DELETE function supports the following parameters: AFTERDATETIME, ALL, BEFOREDATETIME, DATETIME, NEWEST, OLDEST. You cannot use these parameters with a generic program name.

Example: Delete Symbolic Information by Program

The following example deletes all symbolic data for program ORDEDIT and the oldest version of program TEST1. It also deletes all programs that start with COB.

```
//UTILITY  JOB
//STEP1    EXEC PGM=IN25UTIL
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//MESSAGE  DD SYSOUT=*
//PROTSYM  DD DSN=CAI.PROTSYM,DISP=SHR
//CARDS    DD *
PASSWORD=12345678
DELETE=ORDEDIT,ALL
DELETE=TEST1,OLDEST
DELETE=COB*
/*
```

- **INITIALIZE**
  Initializes the symbolic file. This function must always be run after a symbolic file is created using VSAM Access Method Services.
  For a newly defined file, the IN25UTIL program preformats all records. If you perform this function for an existing file, all symbolic data is removed.
  The PASSWORD control statement, if specified, must precede the INITIALIZE control statement.
  The INITIALIZE function supports the MAXPGMVER and USEDSPACEMSG parameters.

Examples: Initialize a Symbolic File

The following example initializes a symbolic file without support for multiple program versions:

```
//UTILITY  JOB
//STEP1    EXEC PGM=IN25UTIL
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//MESSAGE  DD SYSOUT=*
//PROTSYM  DD DSN=CAI.PROTSYM,DISP=SHR
//CARDS    DD *
PASSWORD=12345678
INITIALIZE
/*
```

The following example initializes a symbolic file that allows for up to three versions of each program:

```
//UTILITY  JOB
//STEP1    EXEC PGM=IN25UTIL
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//MESSAGE  DD SYSOUT=*
//PROTSYM  DD DSN=CAI.PROTSYM,DISP=SHR
//CARDS    DD *
PASSWORD=12345678
INITIALIZE,MAXPGMVER=3
/*
```

- **PASSWORD=*pw***
  Specifies a password that is required when maintaining the symbolic file. You need to specify the password only once per execution, but it must precede the first update request.
  The value specified must match the value of the SYMPSWD keyword in the CAVHCONF macro.
  If EXTSEC=SYMBOLIC is enabled on the CAVHCONF macro, then the PASSWORD=pw statement is not required and will be ignored.
  The default installation password is 12345678.

- **PRINT=*name***
  Prints the newest or most recent version of the program on the PROTSYM file specified by name. Ensure to provide the OUTPUT DDname.
  The PRINT function supports the following parameters: AFTERDATETIME, ALL, BEFOREDATETIME, DATETIME, NEWEST, OLDEST.

Example: Print a Program Listing

The following example prints the newest saved listing for program ORDEDIT, all saved listings for program TEST1, and all listings for program TEST2 that were saved after 28 January, 2014:

```
//UTILITY  JOB
//STEP1    EXEC PGM=IN25UTIL
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//MESSAGE  DD SYSOUT=*
//OUTPUT   DD SYSOUT=A,DCB=(LRECL=133,BLKSIZE=3990)
//PROTSYM  DD DSN=CAI.PROTSYM,DISP=SHR
//CARDS DD *
PRINT=ORDEDIT
PRINT=TEST1,ALL
PRINT=TEST2,AFTERDATETIME=2014/01/28
/*
```

- **PURGE=*nnn***
  Removes symbolic data for any program that has not been compiled or assembled within the number of days specified by *nnn*, where *nnn* is a decimal number from 1 to 365.
  The PASSWORD control statement, if specified, must precede the PURGE control statement.
  Data for programs loaded using the NOPURGE postprocessor option are not affected by this function.

Example: Purge Symbolic Information by Age

The following example purges all programs that have not been compiled or assembled within the last 20 days:

```
//UTILITY  JOB
//STEP1    EXEC PGM=IN25UTIL
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//MESSAGE  DD SYSOUT=*
//PROTSYM  DD DSN=CAI.PROTSYM,DISP=SHR
//CARDS    DD *
PASSWORD=12345678
PURGE=20
/*
```

- **RELOAD=*name***
  Reloads all symbolic data from a dataset to the symbolic file for the program specified by *name*. To reload symbolic data for all programs, specify RELOAD=ALL. Ensure to provide the RELOAD DDname with the following DCB parameter:

  ```
  DCB=(RECFM=FB,LRECL=2042,BLKSIZE=20420)
  ```

The PASSWORD control statement, if specified, must precede the RELOAD control statement.
You can use generic program names on the RELOAD control statement. Use a trailing wildcard (*) in the program name to reload all matching programs.
The RELOAD function supports the following parameters: AFTERDATETIME, ALL, BEFOREDATETIME, DATETIME, NEWEST, OLDEST. These parameters cannot be used with a generic program name.
You can rename a program that is being a reloaded by providing a new program name after the RELOAD control statement and any multiversion subparameters. You cannot rename a program when you specify a generic program name.

Example: Reload Programs

The following example reloads the newest version of program ORDEDIT and all versions that were processed in February, 2015. It also reloads the oldest version of ORDEDIT while renaming it to ORDEDIT2 and reloads all programs that start with COB.

```
//UTILITY  JOB
//STEP1 EXEC PGM=IN25UTIL
//STEPLIB DD DSN=CAI.CAVHLOAD,DISP=SHR
//MESSAGE DD SYSOUT=*
//RELOAD   DD  DISP=SHR,DSN=USER.UNLOAD,
//         DCB=(RECFM=FB,LRECL=2042,BLKSIZE=20420)
//PROTSYM DD DSN=CAI.PROTSYM,DISP=SHR
//CARDS DD *
PASSWORD=12345678
RELOAD=ORDEDIT
RELOAD=ORDEDIT,DATETIME=2015/02
RELOAD=ORDEDIT,OLDEST,ORDEDIT2
RELOAD=COB*
/*
```

- **REPORT**
  Produces a Symbolic File report that contains statistics and a detailed report on each program.

Example: Generate Reports

```
//UTILITY  JOB
//STEP1    EXEC PGM=IN25UTIL
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//MESSAGE  DD SYSOUT=*
//PROTSYM  DD DSN=CAI.PROTSYM,DISP=SHR
//CARDS    DD *
REPORT
/*
```

- **UNLOAD=*name***
  Unloads all symbolic data from the symbolic file to a dataset for the program specified by name. To unload symbolic data for all programs, specify UNLOAD=ALL. Ensure to provide the UNLOAD DDname with the following DCB parameter:

  ```
  DCB=(RECFM=FB,LRECL=2042,BLKSIZE=20420)
  ```

  You can use generic program names on the UNLOAD control statement. Use a trailing wildcard (*) in the program name to unload all matching programs.
  The UNLOAD function supports the following parameters: AFTERDATETIME, ALL, BEFOREDATETIME, DATETIME, NEWEST. You cannot use these parameters with a generic program name.

Example: Unload Programs

The following example unloads all versions of program ORDEDIT that were processed after 31 December, 2014. It also unloads all programs that start with COB.

```
//UTILITY  JOB
//STEP1 EXEC PGM=IN25UTIL
//STEPLIB DD DSN=CAI.CAVHLOAD,DISP=SHR
//MESSAGE DD SYSOUT=*
//PROTSYM DD DSN=CAI.PROTSYM,DISP=SHR
//UNLOAD   DD  DSN=USER.UNLOAD,
//         DISP=(NEW,CATLG,CATLG),SPACE=(CYL,(5,10),RLSE),
//         DCB=(RECFM=FB,LRECL=2042,BLKSIZE=20420)
//CARDS DD *
UNLOAD=ORDEDIT,AFTERDATETIME=2014/12/31
UNLOAD=COB*
/*
```

- **UPDATE**
  Updates the values for the USEDSPACEMSG parameter. The PASSWORD control statement, if specified, must precede the UPDATE control statement.

Example: Update a PROTSYM Parameter

The following example sets the USEDSPACEMSG parameter of the symbolic file to 80:

```
//UTILITY  JOB
//STEP1    EXEC PGM=IN25UTIL
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//MESSAGE  DD SYSOUT=*
//PROTSYM  DD DSN=CAI.PROTSYM,DISP=SHR
//CARDS    DD *
PASSWORD=12345678
UPDATE,USEDSPACEMSG=80
/*
```

Example: Reorganize the Symbolic File

The following example reorganizes or changes the size of the symbolic file. This job unloads all programs, deletes and defines the symbolic file, initializes the symbolic file, reloads all programs, and generates a system report.

```
//UTILITY  JOB
//UNLOAD   EXEC PGM=IN25UTIL
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//MESSAGE  DD SYSOUT=*
//UNLOAD   DD  DSN=USER.RELOAD,
//         DISP=(NEW,CATLG,CATLG),SPACE=(CYL,(5,10),RLSE),
//         DCB=(RECFM=FB,LRECL=2042,BLKSIZE=20420)
//PROTSYM  DD DSN=CAI.PROTSYM,DISP=SHR
//CARDS    DD *
UNLOAD=ALL
/*
//IDCAMS   EXEC PGM=IDCAMS,COND=(0,NE,UNLOAD)
//SYSUT1   DD  UNIT=SYSDA,VOL=SER=SYMVOL,DISP=SHR
//SYSIN    DD  *
      DELETE 'CAI.PROTSYM'
      DEFINE CLUSTER (NAME(CAI.PROTSYM) -
                  VOLUME(SYMVOL)   -
                  FILE(SYSUT1)     -
                  CYLINDERS(20)    -
                  CISZ(2048)       -
                  RECSZ(2040 2040) -
                  SHR(4 4)         -
                  NUMBERED)        -
     DATA (NAME(CAI.PROTSYM.DATA))
/*
```

```
//RELOAD   EXEC PGM=IN25UTIL,COND=(0,NE,UNLOAD)
//STEPLIB  DD DSN=CAI.CAVHLOAD,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//MESSAGE  DD SYSOUT=*
//RELOAD   DD  DISP=SHR,DSN=USER.RELOAD,
//         DCB=(RECFM=FB,LRECL=2042,BLKSIZE=20420)
//PROTSYM  DD DSN=CAI.PROTSYM,DISP=SHR
//CARDS    DD *
PASSWORD=12345678
INITIALIZE
RELOAD=ALL
REPORT
/*
```

IN25UTIL Parameters

Some functions support parameters that modify the function results. The functions that support these parameters are noted in the function description.

- **AFTERDATETIME=*YYYY/MM/DD HH:MM:SS***
  Applies the function to all versions of the saved source listing for the program specified by *name* which were compiled after the specified date timestamp.
  You can provide a partial timestamp by omitting time values starting from the smallest unit of time.

  > **Note:** A partial timestamp can also be provided for BEFOREDATETIME and DATETIME.

- **ALL**
  Applies the function to all versions of the saved source listing for the program specified by *name*.
  Do not confuse the ALL parameter with the RELOAD=ALL and UNLOAD=ALL functions.

- **BEFOREDATETIME=*YYYY/MM/DD HH:MM:SS***
  Applies the function to all versions of the saved source listing for the program specified by *name* which were compiled before the specified date timestamp.

- **DATETIME=*YYYY/MM/DD HH:MM:SS***
  Applies the function to all versions of the saved source listing for the program specified by *name* which were compiled on the specified date timestamp.

- **MAXPGMVER**
  Indicates the number of versions of a single program the PROTSYM will allow (up to 255). The default value is 1, which indicates no multiple program versions. For assembler programs, multiple program version support uses the HLASM compile date/time on the listing that has the format 2014/08/04 08.36. Hence, multiple program listings for the same program that are compiled within the same minute are not supported.

- **NEWEST**
  Applies the function to the newest version of the saved source listing for the program specified by *name*.

- **OLDEST**
  Applies the function to the oldest version of the saved source listing for the program specified by *name*.

- **USEDSPACEMSG**
  Specifies the limit of used space in PROTSYM in percentage. When the limit is reached, the SYM070 message is issued. If the value equals 0, the SYM070 message is not issued.
  Default: 0
  Values: 0-99

Dynamic Symbolic Support for CA Endevor Software Change Manager

Dynamic symbolic support is an automated way to update your PROTSYM files. Dynamic symbolic support is available to all CA Endevor SCM for Mainframe customers. This feature helps ensure that the correct symbolic is used for debugging, diagnostics, or measurements. The CA SymDump and CA InterTest products support the automatic updating of symbolic files (PROTSYM).

Dynamic Symbolic Support Activation

When symbolic information is required for a program, the testing tools product looks for this information in the PROTSYM files specified for the program that is being debugged or reviewed.

If one of the PROTSYM entries matches the compile date and time, that entry in the PROTSYM is used. If a matching entry is not found, and if dynamic symbolic support is activated, the system attempts to locate the CA Endevor SCM footprint and the listing associated with the load module library, from where the load module was loaded.

If a match is found, the system automatically post-processes the CA Endevor SCM listing into a designated PROTSYM file so that the correct symbolics are used without any interruption or program setup.

Dynamic symbolic support is activated based on individual product installation and option settings. However, the service that dynamic symbolic support provides is centrally delivered through the symbolic common components. We recommend that you install only one copy of the symbolic common components.

Dynamic Symbolic Support Execution

Dynamic symbolic support execution depends on whether your site has installed CA Endevor SCM, and whether your installation uses a single CA Endevor SCM site ID or multiple CA Endevor SCM site IDs.

Single Site ID

Users with a single CA Endevor SCM site ID have the following options:

- Have one copy of C1DEFLTS.
- Run dynamic symbolic support in the address space of the CA Technologies testing tools product.

> ⚠
>
> **Note:** To implement dynamic symbolic support in a testing tools environment, see Interfaces and Compatibility (https://docops.ca.com/display/CAITSD11/Interfaces+and+Compatibility).

Multiple Site IDs

Users with multiple CA Endevor SCM site IDs have the following options:

- Have multiple C1DEFLTS.
- Allow dynamic symbolic support for multiple site IDs to be implemented using the Listing Server.

> ⚠
>
> **Note:** If you are using dynamic symbolic support in a CICS environment, the Listing Server methodology is always used regardless of the number of site IDs.

Listing Server

When dynamic symbolic support service is requested, Listing Server is spawned as a started task (STC) through CA CCI.

> ⚠
>
> **Note:** Data set CAI.CAVHPROC contains a sample PROC that can be used to start the Listing Server.

Define Unique PROC

Define a unique PROC in your PROCLIB for each CA Endevor SCM site ID in your environment.

Each copy of the PROC must have the following PROCNAME:

xxxxxxxy

- *xxxxxxx* is the first seven characters of the PROCNAME defined in the testing tools product.
- *y,* the eighth character, is your CA Endevor SCM site ID.
  For example, if the first seven characters of the PROCNAME are INTNDVR and the CA Endevor SCM site ID is 4, the PROCNAME defined in your PROCLIB is INTNDVR4.

> **Note:** The first seven character names defined in your PROCLIB must match the PROCNAME defined to the CA Technologies testing tools product.

PROCNAME is defined in the following elements by product:

- CA InterTest Batch is defined in IN25SITE
- CA SymDump Batch is defined in CAOUXFDR
- CA InterTest for CICS is defined in IN25OPTS
- CA SymDump for CICS is defined in IN25OPTS
- CA Mainframe Application Tuner is defined in the TUNUDEFS member of the option library

PROC Customization

The sample PROC can be found in CAI.CAVHPROC. When modifying the PROC, consider the following recommendations:

- Specify MSGLEVEL= (1,1) on the JOB statement so that a more comprehensive view of the events leading up to a problem can be obtained. Set MSGCLASS to a class that does not get purged immediately after job termination. A sample JOB statement is shown next:

  ```
  //INTNDVR4 JOB (JOBACNT),MSGLEVEL=(1,1),MSGCLASS=X
  ```

- Remove the STEPLIB DD statement containing the CAI.CAVHLOAD data set name if the data sets that contain the symbolic common components are defined in the LINKLIST. A sample STEPLIB DD statement is shown next:

  ```
  //STEPLIB DD DSN=CAI.CAVHLOAD,DISP=SHR                <== MODIFY
  ```

  > **Note:** The load module data sets containing symbolic common components must be APF-authorized.

- The MODLRESP DD statement specifies the model space allocation (for CA Endevor for SCM response files) to override the default of (TRK, (5,5)). Depending on your requirement, you can override this limit to avoid SB37s. A sample MODLRESP DD statement is shown next:

  ```
  //MODLRESP DD DUMMY,SPACE=(CYL,(1,1))     MODEL FOR RESPONSE DATASET
  ```

- The MODLLIST DD statement specifies the model space allocation (for CA Endevor for SCM listing files) to override the default of (TRK, (5,5). Depending on your requirement, you can override this limit to avoid SB37s. A sample MODLLIST DD statement is shown next:

  ```
  //MODLLIST DD DUMMY,SPACE=(CYL,(5,5))     MODEL FOR LISTING DATASET
  ```

- The SRVPRINT DD statement is used to log CA CCI activities. The presence of SRVPRINT triggers logging information across all dynamic symbolic support components, including CA Endevor for SCM activities, system assigned DDnames and space allocated for all the work files. A sample SRVPRINT DD statement is shown next:

  ```
  //SRVPRINT DD SYSOUT=*     <== COMMENT OUT TO TURN OFF LOGGING
  ```

  Note: We strongly recommend that you do not comment out this DD statement because the information is critical for CA Technical Support to help you resolve any problems you encounter.

- The DSSLOG DD statement is used to log dynamic symbolic support-related diagnostic messages including Binder errors and CA CCI feedback messages. A sample DSSLOG DD statement is shown next:

  ```
  //DSSLOG   DD SYSOUT=*     <== COMMENT OUT TO TURN OFF DSS LOGGING
  ```

  Note: We strongly recommend that you do not comment out this DD statement because the information is critical for CA Technical Support to help you resolve any problems you encounter.

- The JOBLOG DD statement is used to trigger message extraction when any subtask encounters critical errors or abends. The extracted messages, including PSW and the registers when the abend occurred, are reported back to the remote host. A sample JOBLOG DD statement is shown next:

  ```
  //JOBLOG   DD SYSOUT=*     <== REQUIRED FOR REPORTING REMOTE FAILURES
  ```

  Note: We strongly recommend that you do not comment out this DD statement because the information is critical for CA Technical Support to help you resolve any problems you encounter.

If the data sets containing the CA Endevor SCM C1DEFLTS, CA Endevor SCM CSIQLOAD (CONLIB), and CSIQAUTH (AUTHLIB) data sets are not defined in the LINKLIST, also define these data sets in the STEPLIB DD statement concatenation.

⚠

Note: CONLIB must be defined with a DDName of CONLIB. not part of the STEPLIB concatenation.

JCL Considerations

For single C1DEFLTS environment, in addition to the DD statements required to run the CA Technologies testing tools product, you must take the following actions:

- If the data sets containing the load module libraries of the symbolic common components are not defined in the LINKLIST, define them using the STEPLIB DD statement concatenation.

  ⚠

  Note: The load module data sets containing symbolic common components must be APF-authorized.

  If the data sets containing the CA Endevor SCM C1DEFLTS, CA Endevor SCM CSIQLOAD (CONLIB), and CSIQAUTH (AUTHLIB) data sets are not defined in the LINKLIST, also define these data sets in the STEPLIB DD statement concatenation.

  ⚠

  Note: CONLIB must be defined with a DDName of CONLIB, not part of the STEPLIB concatenation.

- Set//SRVPRINT DD SYSOUT=* to trigger dynamic symbolic support related logging
- Set//DSSLOG DD SYSOUT=* to receive the messages extracted from the list server when you have a multiple user environment.

  ⚠

  Note: The DSSLOG DD statement is not required for a single C1DEFLTS environment.

CA Endevor SCM Auto-Populate Activity Log

The following is a sample Auto Populate Activity Log that displays a list of activities with details such as date and time of each dynamic symbolic support event.

```
*** IN25NDVR STARTED AT: 05182011.15021439.
***      L O G   F I L E    DDNAME:SYS00045
*** ALLOCATING PROTSYM INPUTT FILE.
*** PROTSYM INPUTT FILE ALLOCATED. DDNAME: SYS00046 SPACE=(TRK,(50,50))
*** ALLOCATING PROTSYM OUTPUT FILE.
*** PROTSYM OUTPUT FILE ALLOCATED. DDNAME: SYS00047 SPACE=(TRK,(50,50))
*** ALLOCATING PROTSYM CARDS   FILE.
*** PROTSYM CARDS   FLE ALLOCATED. DDNAME: SYS00048 SPACE=(TRK,(01,01))
*** ALLOCATING PROTSYM MESSAGE FILE.
*** PROTSYM MESSAGE FLE ALLOCATED. DDNAME: SYS00049 SPACE=(TRK,(01,03))
*** ALLOCATING PROTSYM   MSGS  FILE.
*** PROTSYM MSGS    FLE ALLOCATED. DDNAME: SYS00050 SPACE=(TRK,(01,03))
*** ALLOCATING PROTSYM REPORT FILE.
*** PROTSYM REPORT FILE ALLOCATED. DDNAME: SYS00051 SPACE=(TRK,(01,03))
*** ALLOCATING NDVR MESSAGE FILE. *
*** NDVR MESSAGE FILE ALLOCATED. DDNAME: SYS00052   SPACE=(TRK,(01,03))
*** ALLOCATING RESPONSE FILE    ***
*** NDVR RESPONSE FILE ALLOCATED.  DDNAME: SYS00053 SPACE=(TRK,(05,05))
*** ALLOCATING LISTING FILE.    ***
*** NDVR LISTING FILE USED AS INPUT TO PROTSYM POST PROCESSOR ALLOCATED. DDNAME: SYS00
054 SPACE=(CYL,(05,05))
** FOOTPRINT:
   @TTOOLSQABASE    COBQAA05  COBCICS PRD     10104...15:35.0......&\...\{.
*** CALLING ENDEVOR API FOR AEPRE OPTION ***
*** BALRING TO ENA$NDVR ****
*** ENDEVOR API STARTED AT:    15:02:14:57 NDVRELEM:COBQAA05   NDVRNAM2:COBQAA05
*** ENDEVOR API ENDED   AT:    15:02:17:05
*** ENDEVOR AEPRE API FUNCTION RETURNED RC 0 ***
*** ALLOCATING PROTSYM FILE. DSN:  AD1QA.SYMDUM85.GUI.NDVRSYM.NDV14
*** PROTSYM FILE ALLOCATED.     DDNAME: SYS00066
*** ADDING  COBQAA05 AS  COBQAA05 TO PROTSYM DSN: AD1QA.SYMDUM85.GUI.NDVRSYM.ND
*** IN25CDRV RETURN CODE = 0    ***
*** CALLING NDVR TO TERMINATE API ***
*** ENA$NDVR API TERMINATED. ***
*** IN25NDVR ENDED AT: 05182011.15021770.
```

Restrictions for PL/I

Follow these restrictions when using symbolic support:

- Controlled Variables are not supported. For example:

  ```
  DCL A CTL;
  ```

  In the DCL above, there would be no symbolic support and CORE='A' would not work.

- Use of redefine is not supported. For example:

  ```
  DCL A CHAR(10);
  DCL B DEF A;
  ```

  In the example above, displaying B with CORE would not work.

- Adjusted variables are not supported. For example:

  ```
  PROC1 : PROC;
  DCL SIZE BIN FIXED(15,0) INIT(10);
  PROC2 : PROC;
  DCL ARRAY(SIZE) BIN(15,0);
  END;
  END;
  ```

  In the example above, an array could not be displayed using CORE.

- Individual array entries are not supported. For example:

  ```
  CORE= 'A(5)'
  ```

  This array entry will not work.

- Duplicate symbol names in the same PROC are not supported. For example:

  ```
  DCL A CHAR(5);
  DCL 1 Z,
  2 P BIN FIXED,
  2 A BIN FIXED;
  ```

  Using the previous example, when CORE='A' is issued, unpredictable results occur. However, the same variable name in different PROCs is supported. For example:

  ```
  PROC1 : PROC;
  DCL A CHAR;
  PROC2 : PROC;
  DCL A BIN FIXED;
  ```

  In the previous example, if CORE='A' were issued in 'PROC1', then the value of 'A' for 'PROC1' would be displayed. If 'PROC2' had been executed to display the value of 'A' in 'PROC2', use the following command: CORE='PROC2:A'.

- Variables that use the PL/I ALLOC and FREE commands can be looked at with CORE only after the ALLOC executes and before the FREE executes. For example:

```
DCL A BASED(P); ... cannot view "A".
ALLOC A;    ... can use CORE to view "A".
FREE A; ... can no longer view "A".
```

- Variables declared in a block can't be viewed until the execution of that PROC's PROC/BEGIN statement. For example:

```
PROC2 : PROC;
DCL Z CHAR; ... cannot view "P".
    ... can view "Z"..
PROC2 : PROC;
DCL P CHAR;     ... can view "P".
    ... can view "Z"..
```

- At least one automatic variable must be declared in a program for symbolics to function.

- Only one label can appear on a PROC statement. For example:

```
LABEL1 : LABEL2 : PROC ;
```

This is not supported.
Use of the PL/I pre-processor options %NOPRINT, NUMBER, GONUM, and MARGINI is prohibited.
If you must use %NOPRINT, call CA.

How Postprocessors Store Symbolic Information

When your application programs are compiled or assembled, symbolic information about the program is written to various reports in the output listing. A program called postprocessor reads the output listing, collects the symbolic information, and stores it in a symbolic repository called a PROTSYM.

PROTSYM

The *PROTSYM* is a VSAM RRDS defined by IDCAMS and initialized using the symbolic utility program IN25UTIL.

For more information about creating a PROTSYM file, see Creating a PROTSYM File (see page 17).

Postprocessors

Several postprocessors exist to extract symbolic information from the supported compilers and assemblers. Each postprocessor is discussed in detail in Adding Symbolic Information (see page 19) .

Postprocessor	Supported Compilers
IN25SYMC	OS/VS COBOL
IN25COB2	Enterprise COBOL for z/OS and OS/390 IBM COBOL for OS/390 and VM IBM COBOL for MVS and VM AD/CYCLE COBOL/370 VS COBOL II
IN25SYMP	Enterprise PL/I for z/OS and OS/390 IBM PL/I for MVS and VM
IN25SYMA	IBM High Level Assembler for MVS, VM, and VSE Assembler H

Additionally, a batch utility driver program, IN25SYMD, is provided to enable loading of more than one member from a library in a single execution.

For more information about IN25SYMD, see Symbolic Support.

Execute Postprocessors

To automatically invoke or execute the postprocessors in batch, use the sample JCL procedures provided in CAI.CAVHPROC or modify your COBOL, PL/I, or Assembler procedures. The method you select depends on the requirements of your installation. Examples of each method are provided in the Adding Symbolic Information (see page 19) section.

Alternatively, load your PROTSYM files online from the CAIPRINT Repository viewer using the SYM primary command in CA SymDump Batch.

For more information about loading symbolic information from the viewer, see CAIPRINT Repository Viewer (https://docops.ca.com/display/CAITSD11/CAIPRINT+Repository+Viewer).

If you have installed and activated the dynamic symbolic support for CA Endevor SCM feature, the system automatically populates or postprocesses symbolic files if there are mismatches, and thus saves you program setup time.

For more information, see Dynamic Symbolic Support (https://docops.ca.com/display/CAITSD11/Dynamic+Symbolic+Support).

Using the listing postprocessors to collect symbolic information does not change your program in any way. The listing produced by your compiler or assembler is used only as input. Your object module is not changed. Only the PROTSYM is updated.