Automatic Binary Optimizer for z/OS 2.2

User's Guide

SC27-9587-01

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 133.

First edition (August 2022)

This edition applies to Version 2.2 of IBM® Automatic Binary Optimizer for z/OS® (program number 5697-AB2), and IBM Automatic Binary Optimizer for z/OS Trial (program number 5697-TR2), and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

You can view or download softcopy publications free of charge at www.ibm.com/shop/publications/order/.

Contents

Tables vii

Preface ix
About this book ix
Abbreviated terms ix
How to read syntax diagrams ix
Summary of changes x
How to send your comments xi
Accessibility features for Automatic Binary Optimizer for z/OS xi

Chapter 1. Overview 1
Benefits 1
Using ABO and Enterprise COBOL together 2

Chapter 2. System requirements 3
Supported operating systems 3
Target hardware levels 4
System Requirements for the RTI Profiler 5

Chapter 3. COBOL module requirements 7
Supported program modules 7
Eligible compilers 8
COBOL language feature and compiler option support 9
Handling ineligible CSECTs 10
Handling CSECT eligibility for COBOL compilers that are released after ABO 11

Chapter 4. Installing and verifying installation 13
Installing IBM Automatic Binary Optimizer for z/OS 13
Verifying installation using the Installation Verification Program (IVP) 13

Chapter 5. Optimizing modules 17
Required DD statements 17
Optimizer directives 18
BOPT 19
IEFOPZ 20
Optimizer options 22
ALLOW 23
ARCH 23
CSECT 24
LIST 26
LOG 27
REPLACE 28
RTIBIND and the IBM Run Time Instrumentation Profiler 28
SCAN 33
Comments 34
Line continuation 34
JCL examples 36
Specifying optimization with BOPT 36
Specifying optimization with IEFOPZ 38

Recommended settings for the z/OS JCL REGION and JCL MEMLIMIT parameters 39
Specifying the language to be used for ABO messages 40
Invoking ABO from TSO, REXX and assembler code 40
Optimizing under TSO 40
Starting the optimizer from an assembler program 45

Chapter 6. Understanding output from the optimization process 49
Log files 49
Listing transform 51
Listing transform contents 51
How to detect ABO optimized modules and ABO specific PPA4 58
SYSPRINT DD and LIST option 59

Chapter 7. Using the ABO Assistant 63
Components of the ABO Assistant 64
How to use the ABO Assistant 68
How to use the Batch SMF Analyzer 68
How to use the Program Analyzer and Optimizer 68
How to use the CICS SMF Analyzer 69
Example reports 70
Example report from the Batch SMF Analyzer 71
Example report from the Program Analyzer and Optimizer 71
Example reports from the CICS SMF Analyzer 72
BOZPAJ parameter error messages 73
Limitations and requirements on Program Analyzer and Optimizer 73
SMF DUMP generation 75
Gathering SMF 110 data 76

Chapter 8. Managing optimization and optimized module deployment process 81
Optimization and deployment usage scenarios 82
Scenario 1: Optimization process with static deployment 82
Scenario 2: Optimization process with dynamic deployment 82
Scenario 3: Optimization process using a hybrid approach 84
Testing information 85

Chapter 9. Resolving problems with optimization and optimized module deployment 87
Resolving problems that occur during optimization time 87
Resolving problems encountered during execution 87
Changes in COBOL module size after optimization 88
Error message and abend code differences 88
Application Delivery Foundation for z/OS 89

Appendix A. JCL sample 91

Appendix B. Return codes 93

Appendix C. Messages 95
ABO messages 95
RTI Profiler messages 115
ABO Assistant messages 118

Appendix D. Run Time Instrumentation report 123

Appendix E. Manual RTI rebinding instructions 131

Notices **133**
Trademarks 133
List of resources **135**
IBM Automatic Binary Optimizer for z/OS publications 135
Related publications 135

Tables

1. Comparing ABO and Enterprise COBOL 6 compiler use cases..... 2

2. Supported hardware levels..... 4

3. COBOL modules that ABO does not support..... 7

4. Ineligible CSECTs and message issued..... 10

5. GOAFTER1 step return code and corresponding missing LE PTFs..... 16

6. Missing Binder PTFs for EXCEPT2 step..... 16

7. The ddnames used for binary optimization..... 17

8. Optimizer options..... 22

9. Recommended allocation parameters..... 30

10. Input modules and their containing CSECTs..... 60

11. Output 1: Optimized modules and their CSECTs..... 60

12. Output 2: Listing transforms..... 60

13. Determination of Updated Side File Name under TEST(SEPARATE(DSNAME))..... 90

14. Determination of Updated Side File Name when SYSDEBUG is specified..... 90

15. Recommended allocation parameters..... 91

16. IBM Automatic Binary Optimizer for z/OS return codes..... 93

Preface

About this book

This book is for IBM Enterprise COBOL for z/OS compiler customers who use IBM Automatic Binary Optimizer for z/OS to improve the performance of their already compiled COBOL programs.

Abbreviated terms

Certain terms are used in a shortened form in this information. Abbreviations for the terms used most frequently are listed alphabetically in the following table.

Term used	Long form
ABO	IBM Automatic Binary Optimizer for z/OS
CSECT	Control section
EBCDIC	Extended binary coded decimal interchange code
HFS	Hierarchical file system
JCL	Job control language
PDS	Partitioned data set
PDSE	Partitioned data set extended

Other terms, if not commonly understood, are spelled out the first time they appear.

How to read syntax diagrams

Use the following description to read the syntax diagrams in this information:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The **>>---** symbol indicates the beginning of a syntax diagram.

  The **--->** symbol indicates that the syntax diagram is continued on the next line.

  The **>---** symbol indicates that the syntax diagram is continued from the previous line.

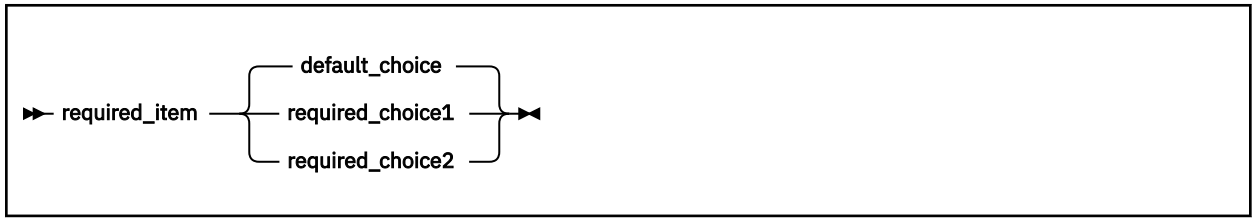
  The **---><** symbol indicates the end of a syntax diagram.

  Diagrams of syntactical units other than complete statements start with the **>---** symbol and end with the **--->** symbol.
- Required items appear on the horizontal line (the main path).

- Optional items appear below the main path.

- If you can choose from two or more items, they appear vertically, in a stack. If you must choose one of the items, one item of the stack appears on the main path.

If one of the items is the default, it appears above the main path and the remaining choices are shown below:



- An arrow returning to the left, above the main line, indicates an item that can be repeated.

- Keywords appear in uppercase (for example, FROM). They must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Summary of changes

This section lists the major changes that have been made to this document for IBM Automatic Binary Optimizer for z/OS 2.2. The latest technical changes are highlighted in the HTML version, or marked by vertical bars (|) in the left margin in the PDF version.

Automatic Binary Optimizer for z/OS 2.2 document refresh in August 2022

- PTF updates (UI82050, UI82051)
  - Introduces a new mechanism for handling CSECT eligibility. (See "Handling CSECT eligibility for COBOL compilers that are released after ABO" on page 11)
  - Improves the LIST optimizer option. (See Usage of the LIST option)
  - Improves the Installation Verification Program (IVP) process. (See Verifying installation using the IVP)
  - Updates some message text and adds the following messages: BOZ1458W, BOZ4136I. (See "ABO messages" on page 95)

Automatic Binary Optimizer for z/OS 2.2

- Supports the ARCH=14 option for generating code that targets the latest z/Architecture® features in IBM z16. (See "ARCH" on page 23)
- Expands the range of compiler releases that are eligible for optimization to include compiled programs produced by Enterprise COBOL for z/OS 5.1.1 to 6.4. (See "Eligible compilers" on page 8)
- Introduces a new component CICS® SMF Analyzer to the ABO Assistant, and a new parameter PROCNAME to the Program Analyzer and Optimizer component of the ABO Assistant. (See Chapter 7, "Using the ABO Assistant," on page 63)

- Introduces a way to detect if a program was optimized by ABO and to reach the ABO specific PPA4 block. (See "How to detect ABO optimized modules and ABO specific PPA4" on page 58)
- Enhances support for the ABO optimized modules in the IBM Application Delivery Foundation for z/OS tools. (See "Application Delivery Foundation for z/OS" on page 89)
- Introduces new messages. (See Appendix C, "Messages," on page 95)
- Removes the following options: ARCH=10 and ARCH=11.

How to send your comments

Your feedback is important in helping us to provide accurate, high-quality information. If you have comments about this documentation, send your comments to the following address: compinfo@cn.ibm.com.

Be sure to include the name of the document, the publication number, the version of the product, and, if applicable, the specific location (for example, the page number or section heading) of the text that you are commenting on.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way that IBM believes appropriate without incurring any obligation to you.

Accessibility features for Automatic Binary Optimizer for z/OS

Accessibility features assist users who have a disability, such as restricted mobility or limited vision, to use information technology content successfully. The accessibility features in z/OS provide accessibility for Automatic Binary Optimizer for z/OS (ABO).

Accessibility features

z/OS includes the following major accessibility features:

- Interfaces that are commonly used by screen readers and screen-magnifier software
- Keyboard-only navigation
- Ability to customize display attributes such as color, contrast, and font size

z/OS uses the latest W3C Standard, WAI-ARIA 1.0 (http://www.w3.org/TR/wai-aria/), to ensure compliance to US Section 508 (https://www.access-board.gov/ict/) and Web Content Accessibility Guidelines (WCAG) 2.0 (http://www.w3.org/TR/WCAG20/). To take advantage of accessibility features, use the latest release of your screen reader in combination with the latest web browser that is supported by this product.

Keyboard navigation

Users can access z/OS user interfaces by using TSO/E or ISPF.

Users can also access z/OS services by using IBM Developer for z/OS Enterprise Edition.

For information about accessing these interfaces, see the following publications:

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Volume I*
- *IBM Developer for z/OS Documentation*

These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Interface information

The ABO online product documentation is available in IBM Documentation (https://www.ibm.com/docs/en/abo), which is viewable from a standard web browser.

PDF files have limited accessibility support. With PDF documentation, you can use optional font enlargement, high-contrast display settings, and can navigate by keyboard alone.

To enable your screen reader to accurately read syntax diagrams, source code examples, and text that contains period or comma PICTURE symbols, you must set the screen reader to speak all punctuation.

Assistive technology products work with the user interfaces that are found in z/OS. For specific guidance information, see the documentation for the assistive technology product that you use to access z/OS interfaces.

Related accessibility information

In addition to standard IBM help desk and support websites, IBM has established a TTY telephone service for use by deaf or hard of hearing customers to access sales and support services:

TTY service
800-IBM-3383 (800-426-3383)
(within North America)

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see IBM Accessibility (www.ibm.com/able).

Chapter 1. Overview

IBM Automatic Binary Optimizer for z/OS (ABO) improves the performance of already compiled IBM COBOL programs. ABO does not require source code, source code migration, or performance options tuning. It uses modern optimization technology to target the latest IBM zSystems, including IBM z16, to accelerate the performance of COBOL applications.

The main function of ABO is to optimize COBOL program modules to improve performance. ABO optimizes directly from the binary code inside the program modules, and this allows ABO to ensure the program logic remains exactly the same. In addition to optimization ABO has some other useful features:

- Use the ABO RTIBIND option to enable your COBOL program modules for performance profiling by the Run Time Instrumentation (RTI) Profiler included with ABO. Both the original and optimized program modules can be enabled for profiling. A detailed performance report is produced to help you identify performance bottlenecks.
- Use the ABO SCAN=Y option to obtain insight into your COBOL application inventory. Using this feature provides details on the makeup of your COBOL modules, including compiler versions, compilation dates and more.

ABO comes with tools to improve the process of realizing performance gains of your COBOL applications.

- The ABO Assistant simplifies the ABO evaluation process, accelerates the deployment of ABO optimized modules, and allows you to easily see the performance improvements from using ABO. The ABO Assistant is a suite of tools to automate the identification and optimization of your top CPU consuming COBOL batch and CICS applications. The ABO Assistant enables you to drill down from high level SMF data all the way to obtain concrete performance evaluation results, and to help prepare your key optimized modules for deployment.
- The Run Time Instrumentation (RTI) Profiler allows you to profile COBOL applications to determine performance bottlenecks. It collects and reports on the execution time CPU performance characteristics of your batch z/OS and CICS applications.

Benefits

Using IBM Automatic Binary Optimizer for z/OS (ABO) to optimize your COBOL applications lowers your IBM zSystem operating costs. The ABO-optimized modules behave the same as the original modules but consume fewer CPU resources and have shorter processing time.

Using ABO offers the quickest time to value. ABO 2.2 delivers comparable performance benefits to the latest Enterprise COBOL 6 compiler. ABO directly optimizes already-compiled modules and testing cost is significantly reduced. Using the ABO Assistant, an evaluation is simplified, and deployment is accelerated.

Using ABO allows you to maximize the return on investment (ROI) of your new IBM zSystems. IBM Z® server delivers performance with hardware features and advanced optimization technology. COBOL applications compiled by earlier COBOL compilers do not utilize new hardware instructions available on the latest IBM zSystems. Optimizing these earlier COBOL applications with ABO provides delivers up to a 25-year jump forward in the evolution of hardware technology, with access to over 600 new hardware instructions that are already on the IBM z16, z15™, and z14 systems.

ABO and Enterprise COBOL are designed to work together. Use Enterprise COBOL to compile programs that are under active development; and ABO to optimize programs that are not frequently compiled.

Starting with this release, ABO can now optimize modules from the latest Enterprise COBOL compilers. This enables COBOL developers to use the best set of compiler options during development, debug and test for the fastest application delivery. ABO can then be used for the best production performance with significantly reduced build complexity and testing cost.

Using ABO and Enterprise COBOL together

From this release, IBM Automatic Binary Optimizer for z/OS can now optimize the latest IBM Enterprise COBOL program modules, up to and including Enterprise COBOL 6.4. IBM Automatic Binary Optimizer for z/OS and the IBM Enterprise COBOL compiler serve complementary purposes.

- ABO optimizes COBOL program modules compiled by Enterprise COBOL; Enterprise COBOL compiles and optimizes COBOL source code.
- There is no performance trade-off when using ABO. In performance tests, ABO 2.2 and Enterprise COBOL 6.4 deliver comparable performance.
- ABO offers a faster path to performance gains. Migrating to Enterprise COBOL 6 can require multiple rounds of compile and test. Source code changes will be required if problems like invalid data are detected in existing applications.
- Use the latest Enterprise COBOL to compile new COBOL programs or to recompile COBOL programs requiring changes. Use ABO to optimize your COBOL programs that are not in your recompilation plan or if the program source code is not available.
- Use the latest Enterprise COBOL to compile new programs with the best options for debug, development and test, and then use ABO to optimize for best performance in production. This combination offers quicker time to value, with significantly reduced testing cost, and suits an iterative development model.

You can choose which product to use according to Table 1 on page 2.

Table 1. Comparing ABO and Enterprise COBOL 6 compiler use cases

Use case	Automatic Binary Optimizer	Enterprise COBOL 6	ABO + Enterprise COBOL
Improve performance and reduce CPU usage: • source available, and • recompilation and test in plan	√ In this case, you can use ABO to optimize modules directly	√ In this case, you can use Enterprise COBOL 6, but migration, recompilation, and options tuning are required	√
Improve performance and reduce CPU usage: • source not available, or • no recompilation plan	√ In this case, you can use ABO to optimize modules directly		√
Implement new features and fix application defects		√ In this case, change your source, recompile with Enterprise COBOL 6 and test	√
Improve performance of your COBOL 4 and earlier programs that interoperate with OS/VS COBOL	√ Your ABO optimized COBOL 4 and earlier programs will interoperate with OS/VS COBOL	The OS/VS programs must be migrated to Enterprise COBOL 6 to interoperate with new compiled code	√
Testing requirements	Significantly less as no risk due to invalid data/code and no source changes. Optimize modules directly.	Detecting problems (e.g. invalid data) requires a 2-step compile/test approach. Fixes require source changes.	

Chapter 2. System requirements

Supported operating systems

IBM Automatic Binary Optimizer for z/OS can be run on the following operating systems:

- z/OS 2.5
- z/OS 2.4
- z/OS 2.3

Some PTFs are required on the systems where Automatic Binary Optimizer for z/OS is installed and running. Other PTFs are required on systems where the ABO generated optimized modules will be running, even if ABO is not installed on these systems.

These PTFs are required on systems where ABO is running:

- z/OS 2.5
  - OA63070/UJ90046 (Binder)
- z/OS 2.4
  - OA57354/UJ04427 (Binder)
  - OA59816/UJ04608 (Binder)
  - OA63070/UJ90045 (Binder)
- z/OS 2.3
  - OA55985/UA97356 (Binder)
  - OA57354/UJ04426 (Binder)
  - OA59816/UJ04607 (Binder)
  - OA63070/UJ90044 (Binder)

These PTFs are required on systems where ABO optimized modules are running:

- z/OS 2.4
  - PH15921/UI65058 (Language Environment®)[1]
  - PH15921/UI65059 (Language Environment - Japanese)[1]
  - PH14705/UI64418 (Language Environment Automatic Binary Optimizer Runtime Engine)[1]
- z/OS 2.3
  - PI84561/UI49013 (Language Environment Automatic Binary Optimizer Runtime Engine)[1]
  - PH14705/UI64417 (Language Environment Automatic Binary Optimizer Runtime Engine)

If the same system is going to be used to both run ABO and run the ABO optimized modules then all the PTFs listed above per z/OS version must be installed on the system.

Optional programs that can be used with ABO:

- Application Delivery Foundation for z/OS 3.2
  - Developer for z/OS Enterprise Edition 14.2
  - Debug for z/OS 14.2
  - Fault Analyzer for z/OS 14.1.8
  - Application Performance Analyzer for z/OS 14.2

It is highly recommended that the latest IBM Automatic Binary Optimizer or IBM Automatic Binary Optimizer Trial PTFs be installed. See the fix list and new features page.

Note:

1. The PTF is included in the GA or GA PTF versions of z/OS 2.3 and z/OS 2.4 so is only required if running early release versions of these z/OS levels.

Target hardware levels

IBM Automatic Binary Optimizer for z/OS can generate program modules for the latest IBM Z servers.

Automatic Binary Optimizer for z/OS uses the same hardware numbering scheme as the COBOL compilers. Table 2 on page 4 lists the hardware levels that are supported by IBM Automatic Binary Optimizer for z/OS 2.2. You can use the ARCH option to specify which hardware level you want the ABO produced modules to target.

Table 2. Supported hardware levels

Hardware level	Description
12	Generates code that uses instructions available on 3906-xxx (IBM z14) and 3907-xxx (IBM z14® ZR1) models in z/Architecture mode. Specifically, these level 12 machines and their follow-ons add instructions that support the vector packed-decimal facility, which accelerates packed-decimal computation by storing intermediate results in vector registers instead of in memory.
13	Generates code that uses instructions available on 8561-xxx (IBM z15) and 8562-xxx (IBM z15 T02) models in z/Architecture mode. Specifically, these level 13 machines and their follow-ons add instructions supported by the following facilities: • Vector Packed-Decimal Enhancement Facility • Vector-Enhancements Facility 2 • Miscellaneous Instruction-Extensions-Facility 3 • Aligned Vector Load/Store Hints
14	Generates code that uses instructions available on the 3931-xxxx (IBM z16) model in IBM z/Architecture mode. Specifically, this level 14 machine and its follow-ons add instructions supported by the new vector packed-decimal enhancement facility 2. This new facility adds performance improvements for COBOL programs that contain one or more of the following types of statements: • Exponentiation operations on packed or zoned decimal data items where the exponent is declared with one or more fractional digits • Arithmetic statements involving mixed decimal and floating-point data items • Statements using numeric-edited data items

Table 2. Supported hardware levels (continued)

Hardware level	Description

Notes:

1. ABO can run on any system supported by the z/OS level. For a complete list of IBM Z servers and z/OS support, see z/OS Server Support.
2. On the system where the ABO optimized modules will run, the SYS1.IPLPARM or SYS1.PARMLIB member LOADxx must not include MACHMIG VEF. This will disable the vector extension facility and cause an 0C7 data exception abend when executing vector instructions produced by ABO at all ARCH levels.

System Requirements for the RTI Profiler

For information on RTI Profiler usage, see "RTIBIND and the IBM Run Time Instrumentation Profiler" on page 28.

The supported z/OS versions for using the RTI Profiler in batch are z/OS 2.2 and above.

The supported z/OS versions for using the RTI Profiler in CICS are z/OS 2.4 and above.

- z/OS 2.4 prerequisite: UJ92369
- z/OS 2.5 prerequisite: UJ92370

The supported IBM Z systems for using the RTI Profiler-enabled modules are zEC12 and later: zEC12, zBC12, z13®, z13s®, z14, z14 ZR1, z15, z15 T02, and z16.

Chapter 3. COBOL module requirements

The input to the IBM Automatic Binary Optimizer for z/OS is your COBOL program modules. ABO verifies that the program modules provided are supported by ABO. ABO then scans the CSECTs within the program modules for those that are eligible for optimization. A CSECT is eligible for optimization by ABO if all of the following conditions apply:

- It was generated by an eligible IBM COBOL compiler.
- All the COBOL language features and options are supported.
- The optimization verification passes all succeed.

ABO optimizes all the eligible CSECTs in the program modules and produces new program modules containing the CSECTs that are successfully optimized.

Supported program modules

IBM Automatic Binary Optimizer for z/OS optimizes program modules output from the binder and load modules output from the linkage editor. The program modules output from the binder can be either program objects or load modules. Load modules produced by the linkage editor must be acceptable input to the binder for ABO to optimize them.

ABO is able to optimize both fully bound or partially bound program modules. A partially bound module is one that has been bound with CALL=NO or NCAL option and are often contained in a link library. If the ALLOW=NOUNRESEXE option has been specified, ABO will not optimize partially bound program modules. See the ALLOW option for more details.

ABO does not support the following COBOL modules:

Table 3. COBOL modules that ABO does not support

COBOL modules that ABO does not support	**Messages issued**
Modules that the binder will not process	BOZ4116I followed by BOZ1429U For example, the binder will not process load modules that have ESD names with invalid characters. When ABO encounters a module with an invalid ESD name, ABO will produce a BOZ4116I message and the BOZ4116I message includes the text of binder message IEW2512E. The text of IEW2512E includes the name of the invalid ESD name.
Modules bound with EDIT=NO	BOZ1423S
Signed modules	BOZ1424S
Modules marked not executable	BOZ1422S

Table 3. COBOL modules that ABO does not support (continued)	
COBOL modules that ABO does not support	**Messages issued**
Modules that include object files from a prelink step but the prelink step was not done properly	BOZ4116I followed by BOZ1419S For example, prelinking may be improper if it was not performed on all object files. This improper prelink could result in load modules that the binder and ABO will not process. ABO would produce a BOZ4116I message followed by a BOZ1419S message if the binder would not process the module. Prelinking is also not proper if the module includes output from multiple prelink steps. In this case, the original module would normally not run properly and ABO would produce a module that would also not run properly.

ABO scans the CSECTs within the program modules for those that are eligible for optimization. A CSECT is eligible for optimization by ABO if it was generated by an eligible COBOL compiler and all COBOL features used in the original COBOL program are supported by ABO.

Eligible compilers

IBM Automatic Binary Optimizer for z/OS 2.2 can optimize CSECTs within program modules that were generated by the following COBOL compilers:

- Enterprise COBOL for z/OS 6[2]
- Enterprise COBOL for z/OS 5[4]
- Enterprise COBOL for z/OS 4
- Enterprise COBOL for z/OS 3
- COBOL for OS/390® & VM 2
- COBOL for MVS™ & VM 1.2
- COBOL/370 1.1
- VS COBOL II 1.4.0[5]
- VS COBOL II 1.3.x [5]

Notes:

1. COBOL modules that have been processed by CA-Optimizer cannot be optimized by ABO. For these types of modules it is recommended to use ABO to optimize the original module created by the COBOL compiler before it was processed by CA-Optimizer.
2. Enterprise COBOL 6.4 programs must have been compiled with the SMARTBIN compiler option to be eligible for ABO optimization. Programs compiled with SMARTBIN contain extra binary metadata that allows ABO to optimize them. This binary metadata is only produced by COBOL 6.4 and is required for ABO optimization for Enterprise COBOL 6.4 produced CSECTs. Compiled programs produced by all earlier releases and versions of the COBOL compiler do not produce this extra binary metadata, nor is it required for ABO optimization.
3. The binary metadata produced under SMARTBIN is placed in a NOLOAD section of the module by the compiler. These SMARTBIN enabled modules will be larger on disk compared to using NOSMARTBIN but there is no memory increase when the compiled code is running, because NOLOAD sections are not loaded by default. This binary metadata is only loaded when needed for later ABO optimization.
4. CSECTs produced by the initial release 5.1.0 version of Enterprise COBOL for z/OS 5 are not eligible for optimization. CSECTs produced by version 5.1.1 and later are eligible for optimization assuming other eligibility checks pass.

5. To be eligible for ABO optimization, VS COBOL II programs must have been migrated away from using the VS COBOL II runtime. For the steps required, see https://www.ibm.com/docs/en/cobol-zos/4.2?topic=environment-moving-from-vs-cobol-ii-run-time.

COBOL language feature and compiler option support

Supported COBOL language features and compiler options

The vast majority of COBOL language features are supported using IBM Automatic Binary Optimizer for z/OS.

Here is a list of key COBOL features that are supported in IBM Automatic Binary Optimizer for z/OS 2.2.

- ARCH(6 | 7 | 8 | 9 | 10 | 11 | 12 | 13)"4" on page 10
- ARITH(EXTEND | COMPAT)
- CICS
- CICS HANDLE ABEND
- CICS HANDLE AID
- CICS language translator generated SERVICE LABEL statements
- CMPR2
- Db2®
- DLL
- ENTRY
- IMS
- I/O and debugging declaratives
- JSON
- NOOPTIMIZE, OPTIMIZE(STD | FULL)"3" on page 10
- NUMCHECK
- NUMPROC(NOPFD | PFD | MIG)
- OPTIMIZE(0 | 1 | 2)
- Program segmentation"1" on page 10
- RECURSIVE
- RENT and NORENT
- SORT and MERGE
- SQL
- SSRANGE
- TEST"2" on page 10
- THREAD
- TRUNC(STD | BIN | OPT)
- User written SERVICE LABEL statements
- XML

Unsupported COBOL language features and compiler options

IBM Automatic Binary Optimizer for z/OS 2.2 does not optimize program modules that use the following COBOL features:

- ACCEPT FROM SYSIPT used in the LABEL declarative
- ARCH(14)"5" on page 10

- CLASS
- DISPLAY UPON SYSLST used in the LABEL declarative
- DISPLAY UPON SYSPCH used in the LABEL declarative
- ENTER"3" on page 10
- INVOKE
- Java™-based object oriented (OO) syntax
- LP(64)#unique_34/unique_34_Connect_42_supnote5_6.3andlater
- RERUN "3" on page 10

Notes:

1. Most cases of Program Segmentation are supported. For COBOL 4 or earlier compiled programs only, the remaining unsupported case is when the source for the CSECT processed by ABO contains independent segments, altered GO TO statements and GO TO DEPENDING ON statements. In this case the following message is issued and the CSECT is skipped: `BOZ1455W: unsupported feature "Program Complexity 176" found`
2. For COBOL 4 or earlier modules, although programs compiled with TEST and any sub-option can be optimized by ABO, LE will not produce a formatted variable dump for the ABO generated module. LE will produce the following message instead: `<prog> was not compiled with the SYM suboption of the TEST. A formatted variable dump cannot be produced.`
3. This applies to Enterprise COBOL 4 and earlier releases.
4. This applies to Enterprise COBOL 5 and 6.
5. This applies to Enterprise COBOL 6.4.

Related reference

- *Enterprise COBOL for z/OS Programming Guide*

Handling ineligible CSECTs

Although IBM Automatic Binary Optimizer for z/OS (ABO) only optimizes CSECTs generated by the compilers listed in the Eligible compilers section, ABO will tolerate modules containing CSECTs from other COBOL compilers and languages.

ABO examines each CSECT before optimization begins. If any of the conditions in the following table applies, the CSECT is not eligible for optimization, a message is issued and the CSECT is skipped. If at least one eligible CSECT for optimization is found in a module, any ineligible CSECTs are copied over unchanged to the target module along with the optimized CSECTs.

Table 4. Ineligible CSECTs and message issued

Ineligible CSECT	**Message issued**
The CSECT name does not match the CSECT filter expression specified. See CSECT.	BOZ4113I
The CSECT is generated from a language other than COBOL. For example, it is from HLASM, C/C++, or PL/I.	No specific message issued [1]
The CSECT is not compiled by one of the eligible COBOL compilers.	BOZ1455W
The CSECT is compiled by one of the eligible COBOL compilers, but it contains one or more unsupported COBOL statements listed in Summary of unsupported COBOL features.	BOZ1455W

[1] The optimizer option SCAN=Y can be used to determine the types of CSECTs present in a module.

Table 4. Ineligible CSECTs and message issued (continued)	
Ineligible CSECT	**Message issued**
The CSECT is too complex for ABO to safely optimize and generate correct functioning code.	BOZ1455W
The CSECT contains any unexpected data or code. This can include, but is not limited to, any missing, corrupted or other erroneous strings ABO relies on to properly understand the CSECT contents and perform correct optimizations.	BOZ1455W
The CSECT has previously been optimized by ABO.	BOZ1455W
The CSECT is compiled with Enterprise COBOL 6.4 without the SMARTBIN compiler option.	BOZ1498W
The binary metadata in the CSECT mentioned in an earlier message is invalid.	BOZ4125I
The binary metadata in the CSECT mentioned in an earlier message is of a different format than that supported by the optimizer.	BOZ4126I
The CSECT is from a COBOL compiler that compiles at an SLEVEL that is not eligible to be optimized until ABO is updated. See "Handling CSECT eligibility for COBOL compilers that are released after ABO" on page 11.	BOZ1455W, BOZ4136I

Handling CSECT eligibility for COBOL compilers that are released after ABO

When a COBOL compiler version or PTF is released after ABO, the CSECTs produced by that compiler might not be eligible for optimization until ABO is updated. For instance that compiler may introduce features or changes that require updates to ABO.

ABO uses a new mechanism to check whether this is the case. The CSECTs produced by such a compiler will embed a new value, called *SLEVEL*, into the CSECT. ABO determines whether it supports a CSECT by comparing the SLEVEL of that CSECT with the one that ABO supports. If the SLEVEL in the CSECT is above the value supported by ABO, then the CSECT is not eligible for optimization. The optimizer issues an unsupported message BOZ1455W reporting the COBOL release and SLEVEL for the CSECT, and then issues BOZ4136I reporting the SLEVEL that ABO supports.

For instance, suppose that a CSECT XPROG was created by a COBOL 6.4 PTF compiler that produces SLEVEL 2 CSECTs. This requires a corresponding ABO version that supports up to SLEVEL 2, but suppose the version of ABO installed only supports up to SLEVEL 1, then the following messages will be issued by the optimizer:

```
10:47:15      Processing CSECT XPROG
10:47:15           BOZ1455W: unsupported feature "Enterprise COBOL 6.4.0 (SLEVEL=2)" found
10:47:15           BOZ4136I: "Enterprise COBOL 6.4" is supported up to SLEVEL 1.
10:47:15           Skipping optimization of CSECT XPROG, return code 4
```

If there are any ineligible COBOL releases for an ABO PTF, the ABO fix list page will list each one, and what minimum later ABO PTF supports it.

To avoid the above messages, and to allow those CSECT(s) to be eligible for optimization by ABO, the user should install a later version of ABO, as documented in the table. Contact IBM Support if this does not resolve the issue.

Note that the SLEVEL mechanism is intended to be used rarely. The usual case for a new COBOL compiler PTF is that ABO automatically supports its CSECTs without requiring ABO to be updated. In this case, the value of SLEVEL produced by that compiler PTF will not be incremented from that of the previous PTF; and if ABO supported previous PTFs and versions, it will still support these CSECTs.

Chapter 4. Installing and verifying installation

Installing IBM Automatic Binary Optimizer for z/OS

All information about installing IBM Automatic Binary Optimizer for z/OS is included in the Program Directory provided with the product.

It is highly recommended that the latest IBM Automatic Binary Optimizer for z/OS PTFs also be installed. See the fix list and new features page for a list of IBM Automatic Binary Optimizer for z/OS PTFs and APARs.

Related reference

"IBM Automatic Binary Optimizer for z/OS publications" on page 135

Verifying installation using the Installation Verification Program (IVP)

After you complete the SMP/E installation of ABO, use the ABO Installation Verification Program (IVP), BOZJIVP, to verify that ABO is installed correctly and is functional.

Note: The PTFs for APAR OA63070 on z/OS 2.3, 2.4, and 2.5 are not currently checked by the IVP so it must be manually verified that they are installed on the systems where ABO is running.

Overview of BOZJIVP

The ABO Installation Verification Program (IVP), BOZJIVP, is located in the ABO sample library *HLQ*.SBOZJCL, where *HLQ* is the prefix used for the target libraries in your ABO SMP/E installation.

Run the IVP on any system on which you plan to use ABO and on any system where the optimized modules produced by ABO will be running.

Note: ABO can run on any hardware level supported by the minimum z/OS level but the ABO generated optimized modules can only run on z14 Models M01- M05, z14 ZR1, z15 Models T01 and T02, z16 systems. See "Target hardware levels" on page 4 for more information. Keep these minimum hardware requirements in mind when you examine the IVP results.

Using BOZJIVP

To proceed with the IVP process on the selected system, edit BOZJIVP according to the included JCL description, and then submit it.

This job contains the following steps:

1. LKED1 - Link-edits the first COBOL program using as input the object BOZOBJ1 in the same sample library.

   Note: The BOZOBJ1 program was compiled using Enterprise COBOL for z/OS 4.2 with the OPT(STD) option in effect. The program source example, BOZSRC1, is also available in the same library for your convenience.
2. GOBEFOR1 - Runs the original version of the first program.
3. VERIFY1 - Verifies z/OS version eligibility to run ABO.
4. OPTIMIZ1 - Optimizes the first COBOL program using ABO.
5. VERIFY2 - Verifies IBM Z server type eligibility to run ABO optimized modules.
6. GOAFTER1 - Runs the optimized version of the first COBOL program.

7. EXCEPT1 - Notifies about possible MACHMIG VEF specification in SYS1.IPLPARM(LOADxx). Runs if step GOAFTER1 condition code is S0C7 only.

   Note: The LOADxx member may be located on some systems in SYS1.PARMLIB instead of SYS1.IPLPARM
8. POSTABND - Notifies about possible machine architecture level or LE PTF level conflict. Runs if step GOAFTER1 condition code is S0C1, S0C4, or S0C6 only.
9. LKED2 - Link-edits the second COBOL program using as input the object BOZOBJ2 in the same sample library.

   Note: The BOZOBJ2 program was compiled using Enterprise COBOL for z/OS 5.1 with the default options. The program source example, BOZSRC2, is also available in the same library for your convenience.
10. GOBEFOR2 - Runs the original version of the second program.
11. OPTIMIZ2 - Optimizes the second program using ABO.
12. GOAFTER2 - Runs the optimized version of the second COBOL program.
13. EXCEPT2 - Notifies about possible Binder PTF level conflict. Runs if step GOAFTER2 condition code is U4094 only.
14. REPORT - Reports the IVP results.

Results

You will receive a return code of 0 or 4 for each of the preceding steps when the IVP runs successfully. After the REPORT step completes, a report is available in the SYSTSRPT output file and in the JESMSGLG JOBLOG.

The following example shows a sample IVP report in the SYSTSRPT output file:

```
*** The original program start time is: 10:42:22.72

*** The original program end time is: 10:44:10.71

***                                                  ***
***            Optimization successful!              ***
***                                                  ***

*** The optimized program start time is: 10:44:11.41

*** The optimized program end time is: 10:44:15.63

***                                                  ***
*** The elapsed time is reduced by 103.77 sec        ***
***                                                  ***

***                                                  ***
***      Installation verification successful!       ***
***                                                  ***
```

The following example shows a sample JESMSGLG JOBLOG. Note that the `Installation verification successful!` message is present in both the JOBLOG and in the console.

```
10.50.21 JOB02018  -                                        ----TIMINGS (MINUTES)-----
10.50.21 JOB02018  -STEPNAME PROCSTEP    RC   EXCP   CONN       TCB       SRB  CLOCK
10.50.21 JOB02018  -LKED1                00    168     29  0.000077  0.000005    0.0
10.50.42 JOB02018  -GOBEFOR1             00    180     54  0.340171  0.000008    0.3
10.50.42 JOB02018  -VERIFY1              00     34      4  0.000047  0.000000    0.0
10.50.42 JOB02018  -OPTIMIZ1             00  16297    127  0.000933  0.000002    0.0
10.50.42 JOB02018  -VERIFY2              00     32      3  0.000040  0.000000    0.0
10.50.43 JOB02018  -GOAFTER1             00    195     53  0.010942  0.000014    0.0
10.50.43 JOB02018  -EXCEPT1           FLUSH      0      0  0.000000  0.000000    0.0
10.50.43 JOB02018  -POSTABND          FLUSH      0      0  0.000000  0.000000    0.0
10.50.43 JOB02018  -LKED2                00    199     32  0.000178  0.000006    0.0
10.50.43 JOB02018  -GOBEFOR2             00    312     77  0.000089  0.000025    0.0
10.50.43 JOB02018  -OPTIMIZ2             00  16306    128  0.001170  0.000003    0.0
10.50.44 JOB02018  -GOAFTER2             00    311     73  0.000090  0.000021    0.0
10.50.44 JOB02018  -EXCEPT2           FLUSH      0      0  0.000000  0.000000    0.0
10.50.44 JOB02018  +***     Installation verification successful!      *** ABOTEST
10.50.44 JOB02018  -REPORT               00     54      6  0.000061  0.000001    0.0
```

If the VERIFY1 step fails, you will see the following message in both the JOBLOG and in the console:

```
z/OS version: xx.xx is not a supported z/OS version to run ABO.
```

If the VERIFY2 step fails, you will see the following message in both the JOBLOG and in the console:

```
IBM z server: type (name)
is not a supported hardware level to run ABO optimized modules.
```

For example:

```
IBM z server: 2817 (zEnterprise 196)
is not a supported hardware level to run ABO optimized modules.
```

If the OPTIMIZ1 step fails, verify the messages in this step log file to see which system or Language Environment component is possibly missing. Fix the problem, and then run the BOZJIVP job again.

If the GOAFTER1 step fails, verify if step EXCEPT1 or POSTABND was executed. If the step EXCEPT1 was executed, you will see the following message in both the JOBLOG and in the console:

```
If MACHMIG VEF is specified in SYS1.IPLPARM(LOADxx), then it must be removed for ABO optimized
modules.
```

Remove MACHMIG VEF specification from SYS1.IPLPARM(LOADxx), and then run the BOZJIVP job again.

If the step POSTABND was executed, you will see one of two possible messages in both the JOBLOG and in the console.

POSTABND possible message #1:

```
IVP is incomplete. The ARCH level specified is 13 (z15), but the actual ARCH level is 12 (z14)
```

If you see message #1 then correct the specified ABO ARCH option to be at, or lower than, the current level of IBM Z where the IVP is being run. Run the BOZJIVP job again after this correction.

POSTABND possible message #2:

```
If any required Binder/LE PTFs is missing, then install it. Otherwise, contact IBM service to
report the problem
```

If you see message #2 then verify which Language Environment PTFs are missing.

Verify which Language Environment PTF is possibly missing. If one or more of the "Language Environment Automatic Binary Optimizer Runtime Engine" PTFs listed in “Supported operating systems” on page 3 are not installed, an 0C1 abend is likely to occur. If the "Language Environment Automatic Binary Optimizer Runtime Engine" PTF is installed but is not the latest PTF listed in the Program Directory, a U4038 abend will occur and one of the following messages will be displayed:

```
IGZ0153S Program BOZSRC1 was compiled with a level of the compiler that requires service to be
installed on Language Environment.
IGZ0355S Program BOZSRC1 was optimized with a level of the Automatic Binary Optimizer that
requires service to be installed on Language Environment.
```

"Language Environment Automatic Binary Optimizer Runtime Engine" PTFs on z/OS 2.3 will cause the first message to be issued, and PTFs on z/OS 2.4 will cause the second message to be issued. The second message will be emitted on z/OS 2.5.

If instead of an abend, the GOAFTER1 step fails with a non-zero return code, then one or more of the following Language Environment APAR/PTFs need to be installed according to the return code from this failing step.

An entry of N/A in the table stands for ‘Not Applicable’ and means that this return code should not happen on the specified z/OS version.

If the GOAFTER1 step fails in one of these N/A cases, then contact IBM service to report the problem.

Table 5. GOAFTER1 step return code and corresponding missing LE PTFs			
Return code	**z/OS 2.3**	**z/OS 2.4**	**z/OS 2.5**
16 or other nonzero value	PH14705/UI64417	PH15921/UI65058 PH15921/UI65059 PH14705/UI64418	N/A
17	PI84561/UI49013	N/A	N/A

Install the required PTFs, and then run the BOZJIVP job again.

If the EXCEPT2 step was executed, you will see the following message in both the JOBLOG and in the console:

```
If any required Binder/LE PTFs is missing, then install it. Otherwise, contact IBM service to
report the problem.
```

If you see this message, then verify which Binder PTF is missing.

Table 6. Missing Binder PTFs for EXCEPT2 step		
z/OS 2.3	**z/OS 2.4**	**z/OS 2.5**
OA63070/UJ90044	OA63070/UJ90045	OA63070/UJ90046

An 0C1 abend will also occur if you attempt to run the ABO generated modules on a system that is not supported by ABO. See “Target hardware levels” on page 4 for the supported systems.

Chapter 5. Optimizing modules

To use the Automatic Binary Optimizer for z/OS, write your JCL for the optimization process.

Invoking ABO with the EXEC statement

Use the EXEC job control statement in your JCL to invoke ABO. The EXEC statement is as follows:

```
//OPT      EXEC PGM=BOZOPT
```

Required DD statements

The optimization process requires that you specify data sets for specific uses in the optimization process. You can define these data sets in DD statements with the required ddnames. The ddnames that are used by ABO and their characteristics are shown in Table 7 on page 17.

Specifying the optimizer directive BOPT or IEFOPZ

Use BOPT or IEFOPZ to direct ABO. You can include one or more of the BOPT or IEFOPZ directives in the SYSIN DD. For details, see “BOPT” on page 19 and “IEFOPZ” on page 20.

Required DD statements

The table shows the ddnames that are used by ABO.

Table 7. The ddnames used for binary optimization

ddname	Type	Required	Description
STEPLIB	Input	Yes	Specifies the name of the data set containing ABO and the dependent Language Environment runtime data sets.
SYSIN	Input	Yes	Specifies the location of the file that contains the optimizer directives BOPT and IEFOPZ and optimizer options. As a convenience, you can specify the in-stream file in the JCL using DD *.
OPTLOG	Output	Yes	Specifies that the optimization summary information (such as what is optimized and the location of the optimized binaries) is written to this DD. SCAN output is also written here.
SYSPRINT	Output	No, if the LIST option is specified Yes, if the LIST option is not specified	Specifies the default location for the listing transforms. See also “SYSPRINT DD and LIST option” on page 59.

Table 7. The ddnames used for binary optimization (continued)

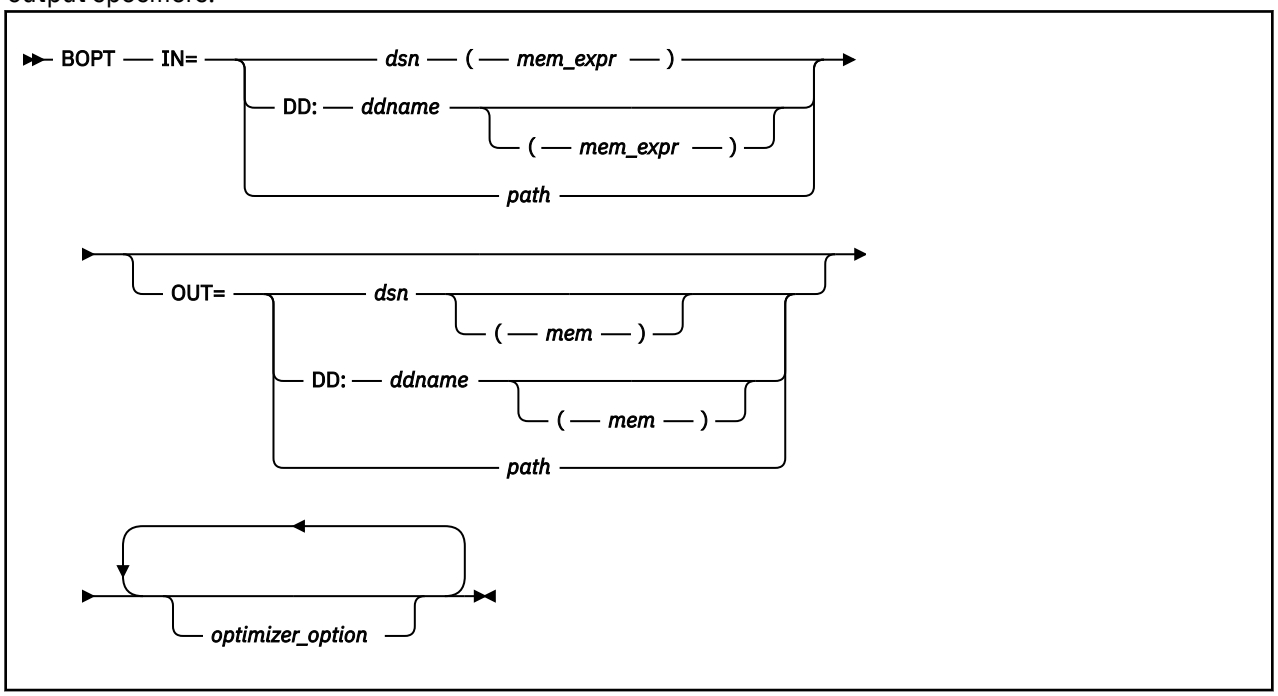
ddname	Type	Required	Description
OPTERR	Output	No	Specifies that the optimization diagnostic information is written to this DD in exceptional circumstances.
CEEDUMP	Output	No	Specifies that the optimization dump information is written to this DD in exceptional circumstances.
CEEOPTS	Input	No, if you want English messages. Yes, if you want Japanese messages.	Specifies the language to be used for messages. See also "Specifying the language to be used for ABO messages" on page 40.
SYSDEBUG	Input	No, if you want to use debug file name embedded in the module. Yes, if you want to use SYSDEBUG to locate the debug file.	Specifies the input debug file to be used for updating the DWARF debug information.
OPTDEBUG	Output	No	Specifies the output file for the updated DWARF debug information.
Note: SYSDEBUG and OPTDEBUG are relevant only when the module was compiled with TEST(SEPARATE) for COBOL 5 modules or later.			

Optimizer directives

Use BOPT or IEFOPZ to direct ABO.

BOPT

You can use the BOPT optimizer directive to produce optimized modules based on explicit input and output specifiers.



IN
Specifies one input module that you want to optimize or multiple input modules when wildcards are given in the *mem_expr* specifier.

OUT
Specifies one output module, or a PDS(E) for one or more output modules when the *mem* specifier is omitted.

DD:*ddname*
Specifies a ddname.

dsn
Is a data set name that must include the high-level qualifier.

mem
Is a data set member name.

mem_expr
Is a data set member name that might include an expression. Only the members whose name string match the expression will be processed. Matching is case insensitive.

A regular expression accepts the following symbols:

*****
Matches any string.

?
Matches any character.

:
Can be used as a separator for multiple expressions. With multiple expressions any expression matching the string counts as a match.

For example:

- To optimize only members PROGA and PROGB:

```
IN=DD:SYSBIN(PROGA:PROGB)
```

<>
Negates the entire expression that follows it.

For example:

- To skip a single member named MEMA:

```
IN=DD:SYSBIN(<>MEMA)
```

- To skip all members whose names begin with MEMB:

```
IN=DD:SYSBIN(<>MEMB*)
```

- To skip members named MEMA and MEMB:

```
IN=DD:SYSBIN(<>MEMA:MEMB)
```

path
Is a full HFS path that starts with a slash (/), for example, /home/user1/a.out.opt.

optimizer_option
Is an optimizer option. For a list of optimizer options that you can specify, see “Optimizer options” on page 22.

Notes:

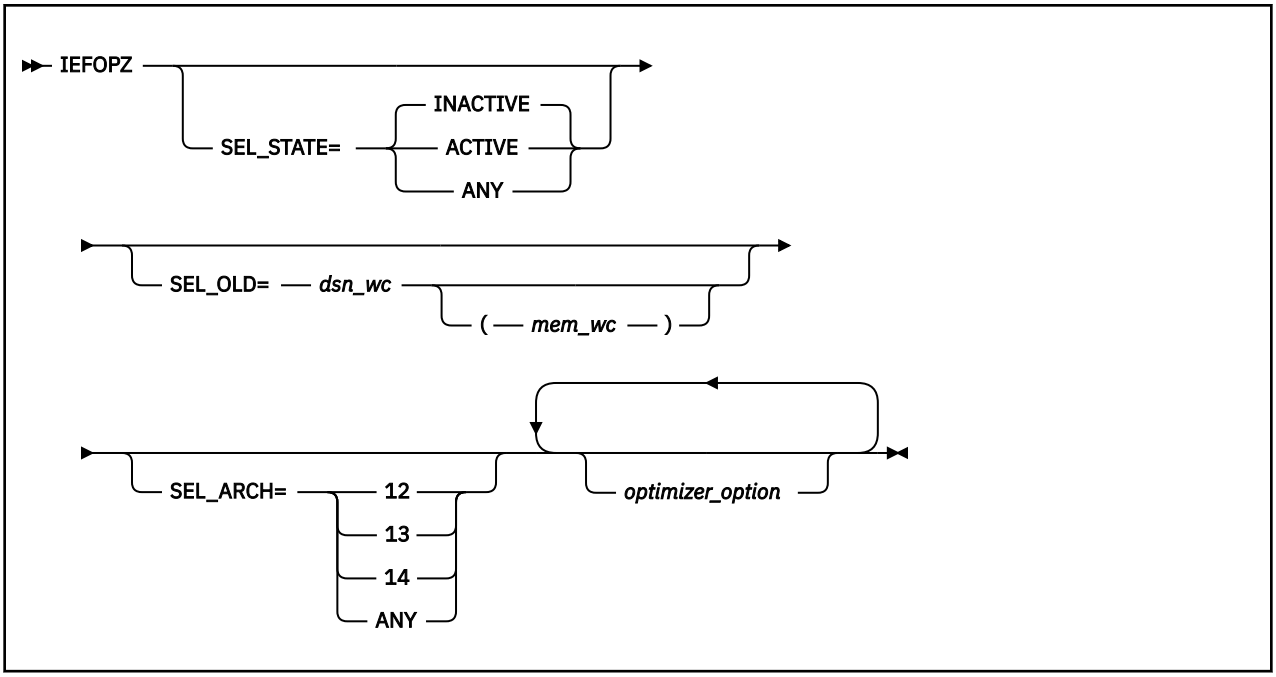
1. The OUT option on BOPT is optional when the SCAN optimizer option is set to Y.
2. When *mem_expr* is specified on the IN option, all members that match *mem_expr* are selected for optimization. Do not include a *mem* specifier on the OUT option when *mem_expr* is specified.
3. When there is no *mem* specifier on the OUT option, the member names for OUT are determined to be those that match the *mem* or *mem_expr* specifier on the IN option.
4. The IN specifier, the OUT specifier, and optimizer options can be in any order.

For examples of the BOPT directive see “JCL examples” on page 36. For sample scenarios of using the BOPT optimizer directive, see “Scenario 1: Optimization process with static deployment” on page 82 and “Scenario 3: Optimization process using a hybrid approach” on page 84.

IEFOPZ

You can use the IEFOPZ optimizer directive to produce optimized modules based on the IEFOPZ configuration.

For information about IEFOPZ configuration, see step 2 in “Scenario 2: Optimization process with dynamic deployment” on page 82.



SEL_STATE
Instructs the optimizer to optimize mappings that match the given state.

ANY
Instructs the optimizer to optimize mappings marked as ACTIVE or INACTIVE.

ACTIVE
Instructs the optimizer only to optimize mappings marked as ACTIVE.

INACTIVE
Instructs the optimizer only to optimize mappings marked as INACTIVE.

SEL_OLD
Restricts optimization to mappings with OLD data sets that match the given selector.

dsn_wc
Is a data set name that might include wildcards using the asterisk (*) symbol. For example, *IN*.LOAD.

mem_wc
Is a data set member name that might include wildcards using the asterisk (*) symbol. For example, M*.

SEL_ARCH
Instructs the optimizer to optimize mappings marked with the given architecture.

12
Instructs the optimizer only to optimize mappings marked as ARCH=12.

13
Instructs the optimizer only to optimize mappings marked as ARCH=13.

14
Instructs the optimizer only to optimize mappings marked as ARCH=14.

ANY
Instructs the optimizer to optimize mappings marked as ARCH=12, ARCH=13, or ARCH=14.

optimizer_option
Is an optimizer option. For a list of optimizer options that you can specify, see “Optimizer options” on page 22.

Notes:

1. Mapping refers to the association of OLD to NEW modules in the IEFOPZ configuration.

2. By default, all INACTIVE modules in OLD data sets of an IEFOPZ configuration are optimized at architecture levels as determined by NEW data sets and as determined by the IncludeMembers and ExcludeMembers configuration specifiers. The SEL_OLD= and SEL_ARCH= are selectors that can be used to restrict optimization or scanning to a subset of these modules.

   The SEL_STATE= is a selector that can be used to change optimization or scanning to ACTIVE modules or to modules of ANY state. While scanning of ACTIVE modules poses no risk, optimization of ACTIVE modules could cause problems for programs that use the ACTIVE modules. Care should be used with selectors SEL_STATE=ACTIVE or SEL_STATE=ANY and when optimization (as opposed to scanning) is performed.
3. The IN specifier, the OUT specifier, and optimizer options can be in any order.

For examples of the IEFOPZ directive see "JCL examples" on page 36. For a sample scenario of using the IEFOPZ optimizer directive, see "Scenario 2: Optimization process with dynamic deployment" on page 82.

Optimizer options

An optimizer option is an ABO option that is applicable to both the BOPT and IEFOPZ optimizer directives.

Optimizer options can be placed at the start of the SYSIN file on one or more lines, or on the BOPT or IEFOPZ optimizer directives.

A global option is an optimizer option that is specified on a line that does not include a BOPT or IEFOPZ optimizer directive. The value of a global option is referred to as the global setting for the option.

When an optimizer option is specified on a particular line that has an optimizer directive, the value applies to that optimization optimizer directive only and reverts back to the global setting for subsequent statements.

Table 8 on page 22 summarizes the optimizer options that apply to both BOPT and IEFOPZ.

Table 8. Optimizer options

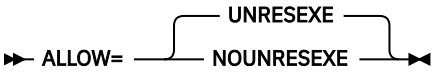
Option	Default	Description
"ALLOW" on page 23	ALLOW=UNRESEXE	Controls the type of program modules that ABO will accept.
"ARCH" on page 23	ARCH=12	Specifies the target hardware level.
"CSECT" on page 24	If you do not specify the CSECT option, ABO will process all eligible CSECTs.	Allows the user to limit processing to one or more CSECTs.
"LIST" on page 26	If you do not specify the LIST option, the listing transforms are placed in the location according to SYSPRINT DD.	Specifies the location of the generated listing transforms.
"LOG" on page 27	If you do not specify the LOG option, ABO will not generate member-level log files.	Specifies the location of the member-level log files to be additionally generated.
"REPLACE" on page 28	REPLACE=Y	Controls whether the output module is written or not.
"RTIBIND and the IBM Run Time Instrumentation Profiler" on page 28	RTIBIND=N | NO	Controls whether to rebind with the RTI program modules.
"SCAN" on page 33	SCAN=N	Controls whether to optimize or scan the program modules.

Table 8. Optimizer options (continued)		
Option	**Default**	**Description**
Note: The ARCH option can be specified at a global level and on the BOPT directive. It cannot be specified on the IEFOPZ directive. For IEFOPZ, use the SEL_ARCH option to select parts of an IEFOPZ configuration matching the SEL_ARCH value to optimize.		

ALLOW

Purpose

The ALLOW option controls the type of program modules that the Automatic Binary Optimizer for z/OS will accept.



Default

ALLOW=UNRESEXE

Usage

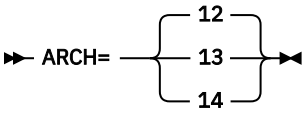
When ALLOW=UNRESEXE is specified, ABO accepts fully bound modules or partially bound modules. The only type of partially bound program modules accepted by the optimizer are those linked with the CALL=NO or NCAL binder option. If the input module is fully bound then the optimized output module will be fully bound. If the input module is partially bound then the optimized module will be partially bound.

When ALLOW=NOUNRESEXE is specified, ABO only accepts fully bound program modules (program object or load module) and always produces a fully bound program module. If partially bound modules are processed when ALLOW=NOUNRESEXE is specified then message BOZ1494S is issued.

ARCH

Purpose

The ARCH option specifies the target hardware level.



Default

ARCH=12

Usage

Use the ARCH option to specify the hardware level that the optimized modules produced by ABO will target.

Optimized modules produced using a lower ARCH setting will run on a higher ARCH system. However optimized modules from a higher ARCH setting will not work on a lower ARCH system.

ARCH setting	Can only be run on IBM zSystem
ARCH=12	z14, z14 ZR1, z15, z15 T02, z16
ARCH=13	z15, z15 T02, z16
ARCH=14	z16

If an invalid combination is attempted then the program is likely to terminate with the following runtime LE message:

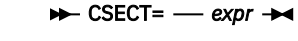
```
CEE3201S The system detected an operation exception (System Completion Code=0C1)
```

For details of these ARCH levels, see "Target hardware levels" on page 4.

CSECT

Purpose

The CSECT option allows you to limit processing to zero or more CSECTs.



Default

By default, if you do not specify the CSECT option, ABO processes all eligible CSECTs.

Parameter

expr
A regular expression for the CSECTs that you want to process. Only the CSECTs whose name string match the expression will be processed. Matching is case insensitive.

Usage

A regular expression accepts the following symbols:

*****
Matches any string.

?
Matches any character.

:
Can be used as a separator for multiple expressions. With multiple expressions any expression matching the string counts as a match.

For example:

- To optimize only CSECTs PROGA and PROGB:

```
CSECT=PROGA:PROGB
```

<>
Negates the entire expression that follows it.

For example:

- To skip a single CSECT named PROGA:

```
CSECT=<>PROGA
```

- To skip all CSECTs whose names begin with PROGB:

```
CSECT=<>PROGB*
```

- To skip CSECTs named PROGA and PROGB:

```
CSECT=<>PROGA:PROGB
```

Notes:

- Spaces and brackets are not allowed in a regular expression.

- An expression must match the entire string. A partial match does not count as a match. This means the expression M*2 matches the string MA2 but not the string MA2A.

Example 1

In the following example, the CSECTs that do not match the wildcard filter are not processed.

JCL COMMAND

```
BOPT IN=DD:SYSBIN(*) OUT=DD:SYSBOUT CSECT=SUB*1*
```

OUTPUT (in OPTLOG)

```
...
10:46:04   Processing CSECT filter expression 'SUB*1*' on member CALLLITT
10:46:04   CSECT CALLLIT was excluded by filter - skip
10:46:04   Processing CSECT SUB01L00, in member CALLLITT
10:46:04        Optimizing CSECT SUB01L00 for zEC12
10:46:04        Succeeded in optimizing SUB01L00
10:46:04        Generating listing transform into DD:SYSPRINT
10:46:04   CSECT SUB02L00 was excluded by filter - skip
10:46:04   CSECT SUB03L00 was excluded by filter - skip
10:46:04   CSECT SUB04L00 was excluded by filter - skip
10:46:04   CSECT SUB05L00 was excluded by filter - skip
10:46:04   CSECT SUB06L00 was excluded by filter - skip
10:46:04   CSECT SUB07L00 was excluded by filter - skip
10:46:04   CSECT SUB08L00 was excluded by filter - skip
10:46:04   CSECT SUB09L00 was excluded by filter - skip
10:46:04   Processing CSECT SUB10L00, in member CALLLITT
10:46:04        Optimizing CSECT SUB10L00 for zEC12
10:46:04        Succeeded in optimizing SUB10L00
10:46:04        Generating listing transform into DD:SYSPRINT
10:46:04   Finished processing, processed 2 of 11 CSECTs in member CALLLITT
```

Example 2

The following example shows how to specify multiple expressions for matching.

JCL COMMAND

```
BOPT IN=DD:SYSBIN(*) OUT=DD:SYSBOUT CSECT=SUB01L00:SUB02L00
```

OUTPUT

```
...
10:49:13   Processing CSECT filter expression 'SUB01L00:SUB02L00' on member CALLLITT
10:49:13   CSECT CALLLIT was excluded by filter - skip
10:49:13   Processing CSECT SUB01L00, in member CALLLITT
10:49:13        Optimizing CSECT SUB01L00 for zEC12
10:49:13        Succeeded in optimizing SUB01L00
10:49:13        Generating listing transform into DD:SYSPRINT
10:49:13   Processing CSECT SUB02L00, in member CALLLITT
10:49:13        Optimizing CSECT SUB02L00 for zEC12
10:49:13        Succeeded in optimizing SUB02L00
10:49:13        Generating listing transform into DD:SYSPRINT
10:49:13   CSECT SUB03L00 was excluded by filter - skip
10:49:13   CSECT SUB04L00 was excluded by filter - skip
10:49:13   CSECT SUB05L00 was excluded by filter - skip
10:49:13   CSECT SUB06L00 was excluded by filter - skip
10:49:13   CSECT SUB07L00 was excluded by filter - skip
10:49:13   CSECT SUB08L00 was excluded by filter - skip
10:49:13   CSECT SUB09L00 was excluded by filter - skip
10:49:13   CSECT SUB10L00 was excluded by filter - skip
10:49:13   Finished processing, processed 2 of 11 CSECTs in member CALLLITT
```

Example 3

In the following example, MEM1 in dataset HLQ.IN.LOAD has two CSECTs named A1 and B1. To limit ABO processing to only A1, use a CSECT=A* filter as follows:

JCL COMMAND

```
BOPT IN=HLQ.IN.LOAD(MEM1) OUT=HLQ.OUT.LOAD(MEM1) CSECT=A*
```

With this CSECT=A* filter in place the OPTLOG looks like the following:

OUTPUT

```
...
17:46:04   Processing CSECT filter expression 'A*' on member MEM1
17:46:04   Processing CSECT A1, in member MEM1
17:46:04           Optimizing CSECT A1 for zEC12
17:46:04           Succeeded in optimizing A1
17:46:04           Generating listing transform into DD:SYSPRINT
17:46:04   CSECT B1 was excluded by filter - skip
17:46:04    Finished processing, processed 1 of 2 CSECTs in member MEM1
...
```

Alternatively CSECT=A* can be specified as a global option so it applies to all subsequent BOPT and IEFOPZ directives unless overridden by a particular directive:

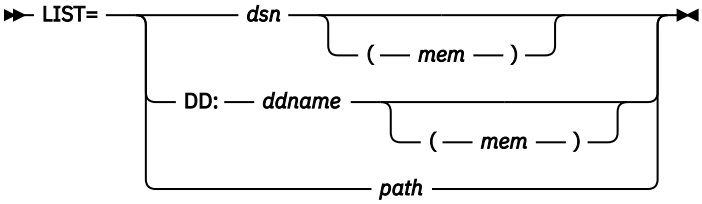
```
//SYSIN DD *
            CSECT=A*
            BOPT IN=HLQ.IN.LOAD(MEM1) OUT=HLQ.OUT.LOAD(MEM1A)
            BOPT IN=HLQ.IN.LOAD(MEM1) OUT=HLQ.OUT.LOAD(MEM1B) CSECT=B*
```

After processing, the member MEM1A will contain the optimized CSECT A1 and the original CSECT B1, and member MEM1B will contain the optimized CSECT B1 and the original CSECT A1.

LIST

Purpose

The LIST option specifies the location of the generated listing transforms.



Default

By default, if you do not specify the LIST option, the listing transforms are placed in the location according to SYSPRINT DD.

Parameters

dsn
Is a data set name that must include the high-level qualifier.

mem
Is a data set member name.

DD:*ddname*
Specifies a ddname.

path
Is a full HFS path that starts with a slash (/), for example, /home/user1/a.list.

Usage

The target of the LIST option can be one of the following items:

- A sequential data set or member of a PDSE (not PDS). If the sequential data set or member of a PDSE exists, it will be overwritten. If the data set specified by LIST=*dsn* is a sequential dataset that exists and is overwritten, or does not exist, it will be (re)allocated as a sequential dataset with the following

attributes: RECFM=FB, LRECL=133, BLKSIZE=0, SPACE=(CYL,(50,50)). The output of multiple CSECT optimizations are added to this single sequential data set.

- A PDS or PDSE. When a CSECT is optimized, the listing transform particular to that CSECT is placed in a member of the PDS or PDSE where the member name is based on the CSECT name (upper cased and truncated to 8 characters). The contents of the member, if any, are overwritten even if the former contents are produced by the optimizer in previous invocations.
- An HFS path. The output of multiple CSECT optimizations are added to this HFS file.

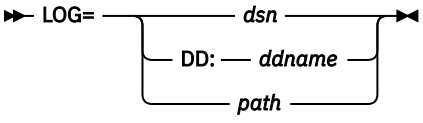
Related information

SYSPRINT DD

LOG

Purpose

The LOG option specifies the location of the member-level log files to be additionally generated.



Default

By default, if you do not specify the LOG option, ABO will not generate member-level log files. Note that regardless of the LOG option the output from the entire ABO invocation will always be generated in the location according to OPTLOG DD.

Parameters

dsn
Is a data set name that must include the high-level qualifier.

DD:*ddname*
Specifies a ddname.

path
Is a full HFS path that starts with a slash(/).

Usage

The target of the LOG option can be one of the following items:

- A PDS or PDSE. When a member is optimized, the log output particular to that member is placed in a member of the PDS or PDSE where the member name is based on the optimized member name (upper-cased and truncated to 8 characters). The contents of the member, if any, are overwritten even if the former contents were produced by ABO in previous invocations.
- An HFS path pointing to a directory. When a member is optimized, the log output particular to that member is placed in a file in the specified HFS directory where the file name is based on the optimized member name appended with `.log`. The contents of the member, if any, are overwritten even if the former contents were produced by ABO in previous invocations.

Notes:

1. The target of the LOG option must be either a PDS or PDSE with no member specified, or an HFS path that points to a directory. If the target of the LOG option is specified as a PDS or PDSE with a member specified, a sequential data set, or an HFS file, an error message will be issued.
2. The target data set of the LOG option should follow the recommended allocation parameters of the OPTLOG in Table 15 on page 91.

Example 1

```
BOPT IN=DD:SYSBIN(*) OUT=DD:SYSBOUT LOG=HLQ.LOG.OUT
```

Example 2

```
BOPT IN=DD:SYSBIN(*) OUT=DD:SYSBOUT LOG=DD:LOG
```

Example 3

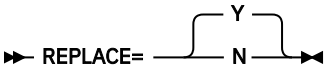
```
BOPT IN=DD:SYSBIN(*) OUT=DD:SYSBOUT LOG=/home/user1/logdir
```

For more information, see “Log files” on page 49.

REPLACE

Purpose

Controls whether the output module is written or not.



Default

REPLACE=Y

Usage

When REPLACE=Y and SCAN=N, the optimized module is written to the output module regardless of whether the output module exists.

- If the output module does not exist, the output module will be created and written.
- If the output module already exists, its content will be overwritten.

However, if there are no eligible COBOL CSECTs present that can be optimized, nothing is written to the output module.

When REPLACE=N is specified, optimization or scanning of the input module is bypassed if the output module already exists; nothing is written to the output module.

You can use REPLACE=N to bypass optimization for already optimized modules. For example, if after binary optimization, you add new members to the original data sets and you want to optimize only the new members, you can use a member wildcard with REPLACE=N as follows:

```
BOPT IN=HLQ.IN.LOAD(*) OUT=HLQ.OUT.LOAD REPLACE=N
```

In some cases, the optimizer prematurely terminates because of exceeded time or other abnormal conditions. To solve the problem, you can incrementally build the optimized modules in a sequence of jobs without spending time repeating optimizations done in an earlier job. Or you can use REPLACE=N in these cases.

RTIBIND and the IBM Run Time Instrumentation Profiler

This topic includes the following sections:

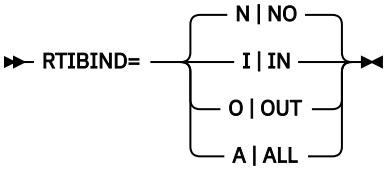
- “Overview” on page 28
- “Capturing profiling results during z/OS batch program execution” on page 29
- “Capturing profiling results during CICS application program execution” on page 30
- “Understanding the RTI Profiler Results” on page 32
- “RTI Profiler capabilities and restrictions” on page 32

Overview

The IBM Run Time Instrumentation (RTI) Profiler is a performance analysis tool to collect and report on the execution-time CPU performance characteristics of your batch z/OS and CICS applications.

The RTIBIND option controls whether to rebind your modules with the RTI program modules to enable the RTI performance profiling reports to be generated when the program runs.

All the Language Environment (LE) CSECTs in your application can be bound with the RTI program modules to enable profiling to get a complete picture of overall CPU performance. This includes programs compiled by the IBM COBOL, C/C++, and PL/I compilers as well as COBOL programs optimized by ABO. The time spent in any LE library routines is also collected and reported.



Default

RTIBIND=N | NO

Usage

When RTIBIND=N | NO is specified, ABO optimizes any eligible CSECTs as usual, and does not rebind any of the input program modules with the RTI program modules.

When RTIBIND=I | IN is specified, ABO does not perform any optimizations, but only rebinds all the input program modules with the RTI program modules (that is, rebind the modules specified by the BOPT IN= directive).

When RTIBIND=O | OUT is specified, ABO optimizes any eligible CSECTs as usual, and rebinds any input program modules that have at least one eligible CSECT optimized with the RTI program modules (that is, rebind the modules optimized and placed in the location specified by the BOPT OUT= directive).

When RTIBIND=A | ALL is specified, ABO optimizes any eligible CSECTs as usual, and rebinds these optimized modules plus any input modules that are not optimized, with the RTI program modules (that is, rebind the union of all the modules specified by BOPT IN= and OUT= directives).

Note: To enable the RTI profiling for CICS multi-program applications, it is recommended to rebind as many programs of this application as needed with RTI program modules. In the CICS environment, RTI controls concurrent and subsequent runs of multiple programs that are rebound with RTI by profiling one program at a time. If a particular RTI Profile-enabled program module executes at various times during the application run, then by default, only up to three profiling reports are generated for that program. However, the limit of profiling reports generated per program is adjustable, see “RTIS transaction” on page 128 for more information. You can recreate the profiling reports for the same program again which will overwrite an existing one after a one-minute RTI-silence time interval. The one-minute silence means no RTI rebound program execution for one minute.

Capturing profiling results during z/OS batch program execution

Before using the RTI Profiler-enabled modules, you must allocate a PDS or PDSE data set in order to hold the profiling results. The RTI Profiler generated report is stored into a member of this data set. In the following JCL examples, this data set is named hlq.SYSPROFD.

The following table shows the recommended allocation parameters for hlq.SYSPROFD.

Table 9. Recommended allocation parameters	
Data sets	**Recommended allocation parameters**
hlq.SYSPROFD (as a PDS)	Space units : CYLINDER Primary quantity : 10 Secondary quantity : 10 Directory blocks : 10 Record format : FB Record length : 80 Block size : 27920 Data set name type PDS
hlq.SYSPROFD (as a PDSE)	Space units : CYLINDER Primary quantity: 10 Secondary quantity : 10 Directory blocks : 10 Record format : FB Record length : 80 Block size : 27920 Data set name type LIBRARY

In addition, when running the application containing the RTI rebound modules there are two modifications required to your run step JCL in order to collect the profile.

In your existing JCL for executing your program:

- Add the ABO installation location, for example `hlqboz.SBOZMOD1`, to the existing STEPLIB. This change is not required if your ABO installation library is in the LNKLST.
- Add the DDNAME SYSPROFD as the location to receive the profiling results. If no SYSPROFD DD card is provided, then a new sequential data set will be allocated with the following name format: *userid*.BOZRTI.SYSPROFD.*pgmname*. If a data set with this name already exists, then it will be overwritten."

Below is a JCL example for this step:

```
//GO EXEC PGM=pgmname
//STEPLIB DD DSN=hlq.OUT.LOAD.RTI,DISP=SHR        <- the RTI rebound modules from ABO
//        DD DSN=hlqboz.SBOZMOD1,DISP=SHR         <- add ABO install location
//SYSPROFD DD DSN=hlq.SYSPROFD(pgmname),DISP=SHR  <- add SYSPROFD DD
```

When program execution completes, the profiling results are contained in `hlq.SYSPROFD(pgmname)`.

If SYSPROFD is not added in the execution step, the message BOZR001I SYSPROFD: *userid*.BOZRTI.SYSPROFD.*pgmname* is generated to the job log, but the execution step continues.

If the program is very short running, then the profiling results file specified by SYSPROFD may be incomplete, and will not be usable for performance investigations. In this case the following message will be displayed in the profiling results file:

```
No profiling data available due to short execution time
Choose programs for profiling that run for at least 5 CPU seconds to obtain valid profiling
results.
```

Capturing profiling results during CICS application program execution

Before using the RTI Profiler-enabled modules, the CICS startup JCL must be updated by placing the ABO load library in DFHRPL concatenation.

If your original application modules are in the data set concatenated under DFHRPL, then place the data set containing the RTI Profiler-enabled modules before the data set containing the original modules.

See the following example fragment of the CICS region startup JCL:

```
//DFHRPL           DD DSN=CICS.TS550.SEYULOAD,DISP=SHR
//                 DD DSN=CICS.TS550.SDFHLOAD,DISP=SHR
```

```
//                DD DSN=CICS.TS550.SDFHLIC,DISP=SHR
//                DD DSN=hlqboz.SBOZMOD1,DISP=SHR              <- add ABO install location
//                DD DSN=hlq.OUT.LOAD.RTI,DISP=SHR             <- add RTI Profiler-enabled
modules
//                DD DSN=hlq.ORIG.LOAD,DISP=SHR                <- original application modules
//SYSPROFD        DD DSN=hlq.SYSPROFD (pgmname)DISP=SHR        <- optional
```

If your original application modules are in the data set listed under the CICS library resource definition, as shown in the following example,

```
DEFINE LIBRARY(libname1)
  RANKING(55)
  STATUS(ENABLED)
  DSNAME01(hlq.ORIG.LOAD)     <- original application modules are in library libname1
  GROUP(LIBS)
```

you have two options to define your RTI Profiler-enabled data set using CICS library resource:

1. Place the name of data set that contains the RTI Profiler-enabled modules of the application into the CICS library resource definition with a lower ranking, as shown in the below example fragment of the CSD definitions:

   ```
   DEFINE LIBRARY(libname2)
     RANKING(45)
     STATUS(ENABLED)
     DSNAME01(hlq.OUT.LOAD.RTI)  <- add data set with RTI rebound modules in lower ranked
   library libname2
     GROUP(LIBS)
   DEFINE LIBRARY(libname1)
     RANKING(55)
     STATUS(ENABLED)
     DSNAME01(hlq.ORIG.LOAD)     <- original application modules are in library libname1
     GROUP(LIBS)
   ```

2. If Option 1 is unsuitable for you due to the number of libraries allowed on the same CICS region limitations or other library resource restrictions applicable in your installation, proceed with this option.
   Place the data set name containing the RTI Profiler-enabled modules of the application in the same CICS library resource definition but placed above the data set with the original modules, as shown in the below example fragment of the CSD definitions:

   ```
   DEFINE LIBRARY(libname1)
     RANKING(55)
     STATUS(ENABLED)
     DSNAME01(hlq.OUT.LOAD.RTI)  <- add data set with RTI rebound modules in front of original
     DSNAME02(hlq.ORIG.LOAD)     <- original application modules are in library libname1
     GROUP(LIBS)
   ```

Option 1 is recommended because it allows you to set the status of the library containing RTI Profiler-enabled modules to disable and stop the RTI Profiler activity at any time during your application run.

For example, you can do this by issuing the command from the CICS terminal:

```
CEMT SET LIBRARY(libname2) DISABLED
```

When using Option 2 configuration you cannot stop profiling using the above CICS terminal command.

You can optionally add DDNAME SYSPROFD in your CICS startup JCL, then follow the instructions for `hlq.SYSPROFD` data set allocation in “Capturing profiling results during z/OS batch program execution” on page 29. If no SYSPROFD DD card is provided, then a new sequential data set will be allocated with the following name format:

```
userid.BOZRTI.SYSPROFD.pgmname.LOG#nnn
```

If a data set with this name already exists, then it will be overwritten.

The nnn corresponds with the report number generated for the program named *pgmname*. For the first run of RTI Profiler-enabled *pgmname* program, the suffix of the report will be `LOG#001`. For each subsequent run of the same program, the corresponding report name suffix will be increased by 1. The

default maximum number of reports per program is 3, but this limit is adjustable. For more information, see "RTIS transaction" on page 128. For each dynamically allocated RTI report file, the corresponding job log message is as follows:

```
BOZR001I SYSPROFD: userid.BOZRTI.SYSPROFD.pgmname.LOG#nnn
```

To support RTI in your CICS region, you must add the following CSD definitions:

```
DEFINE PROGRAM(BOZRIDT) GROUP(BOZRTI1) LANGUAGE(ASSEMBLER)
DEFINE PROGRAM(BOZTRUE) GROUP(BOZRTI1) LANGUAGE(ASSEMBLER) RELOAD(NO)
DEFINE PROGRAM(BOZGLUE) GROUP(BOZRTI1) LANGUAGE(ASSEMBLER) RELOAD(NO)
DEFINE TRANSACTION(RTI) GROUP(BOZRTI1) PROGRAM(BOZRIDT)
DEFINE PROGRAM(BOZSETRT) GROUP(BOZRTI1) LANGUAGE(ASSEMBLER)
DEFINE TRANSACTION(RTIS) GROUP(BOZRTI1) PROGRAM(BOZSETRT)
DEFINE TSMODEL(BOZRTI1) GROUP(BOZRTI1) PREFIX(BOZRTIQ.) EXPIRYINTMIN(9)
DEFINE TSMODEL(BOZRTI2) GROUP(BOZRTI1) PREFIX(BOZRTIQ@BOZRTIQ.) SECURITY(NO)
ADD GROUP(BOZRTI1) LIST(listname)  <- put your list name here
```

Notes:

1. The following keywords in the above definitions: BOZRIDT, BOZTRUE, BOZGLUE, RTI, BOZSETRT, BOZRTIQ@, and BOZRTIQ. are reserved. You can optionally change the name of group BOZRTI1, the TS models BOZRTI1/BOZRTI2, and transaction RTIS, if needed.
2. An RTI program bound into your CICS COBOL modules owns and maintains the CICS TS queues named BOZRTIQ@BOZRTIQ.. The first queue permanently allocates 10 bytes of temporary storage. The second type of queue has * to mask the individual program name and allocates 1 byte of storage per RTI Profiler-enabled program. RTI automatically deletes all queues of that second type after the 1-minute RTI-silence time interval is due.
3. To secure RTI profiling and RTIS transaction usage by CICS users, consider setting SECURITY （YES） with TS model BOZRTI2 and maintaining the TS queue BOZRTIQ@BOZRTIQ. resource security profile as needed.

Understanding the RTI Profiler Results

The output of the RTI Profiler is a text file containing details per compiled or optimized module and CSECT where the program is spending its time while running. There is a high-level summary provided as well as a breakdown of timing samples down to the offsets within each CSECT. For more detailed information, see Run Time Instrumentation report.

In conjunction with the corresponding compiler or ABO listing files for the programs being profiled, the RTI Profiler output can help determine specific parts of your program, down to the machine instruction, where there can be opportunities for improving application CPU performance.

An IBM support or development representative can also request the RTIBIND option be used and an RTI Profile be collected, and its output sent to IBM along with other artifacts (e.g. listing files) from the original compilation or optimization of your programs to aid in performance investigations.

RTI Profiler capabilities and restrictions

The RTI Profiler is best suited for in-depth CPU profiling of LE enabled batch and CICS applications, but does not provide other system-level performance metrics such as wait times, I/O performance, or DASD usage.

The RTI Profiler can be used on all LE enabled batch applications including IMS batch and batch programs that interact with Db2. Running non-LE programs that have been rebound with the RTI program modules is not supported and the result is unpredictable.

The RTI Profiler is not supported for any application that is part of a non-batch IMS environment.

The RTI Profiler can be used on all LE enabled CICS applications.

The RTI Profiler will not work if your z/OS operating system is running on z/VM®. If run on z/VM guest, the message RISTART: AUTH REQUEST FAILED is output to the job log.

The RTI Profiler does not provide CSECT level details for modules in LPA or that are managed by LLA. Profiling sample information will only be reported at the module level for these cases.

If you require more complete and full featured application profiling, we recommend using a profiling tool such as Application Performance Analyzer for z/OS in ADFz.

Notes:

1. SCAN=Y takes precedence over the RTIBIND option settings. Specifying SCAN=Y disables rebinding with the RTI program modules and optimization of any eligible CSECTs regardless of the RTIBIND option.
2. When processing a module originally linked with the AMODE 24 or RMODE 24 option and then processed by ABO with the RTIBIND=I | 0 | A option, the warning message BOZ1490W is expected. The BOZ1490W message reflects an AMODE/RMODE conflict due to rebinding with the AMODE 31 RTI modules. The message has no impact on the original module execution. The program will still run correctly in the original AMODE 24 or RMODE 24 mode.
3. The RTIBIND option is the most convenient way to enable modules for RTI profiling. However, if you require more low-level control of the rebinding process, see Appendix E, “Manual RTI rebinding instructions,” on page 131.
4. If the program to be profiled was compiled using VS COBOL II or an ABO optimized VS COBOL II compiled program, an extra step might be needed to rebind the module to replace the bootstrap routine IGZEBST with the current version from LE. If you do not see any profiling output at the SYSPROFD location after using RTIBIND and attempting to profile VS COBOL II or an ABO optimized VS COBOL II compiled program, follow the steps below to replace the IGZEBST routine before using RTIBIND (or before following the manual rebinding steps) to enable the module for RTI profiling.

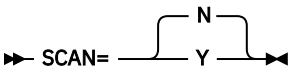
```
//LKED EXEC PGM=IEWL,PARM='options' <- original link options
//SYSLIB DD DSN=hlqcee.SCEELKED,DISP=SHR
// DD DSN=hlqcee.SCEELKEX,DISP=SHR
//LOAD DD DISP=SHR,DSN=&LOAD   <- original module that will be linked with new IGZEBST
//SYSLMOD DD DISP=SHR,DSN=&SYSLMOD(pgmname)  <- output module location
//SYSPRINT DD SYSOUT=*
//SYSLIN DD *
  INCLUDE SYSLIB(IGZEBST)      <- new bootstrap to include from CEE
  INCLUDE LOAD(pgmname)        <- original member from LOAD to link new parts into
  REPLACE -IMMED,IGZEBST       <- bootstrap member to replace
  ENTRY pgmname
  NAME pgmname(R)
//*
```

5. If the program to be profiled is called by an assembler module that is linked with AMODE 24, the assembler module must be rebound by using RTIBIND=I|IN or A|ALL.

SCAN

Purpose

The SCAN option instructs ABO whether to optimize or to scan the program modules



Default

SCAN=N

Usage

When SCAN=N is in effect, ABO performs optimization on the input program modules.

When SCAN=Y is in effect, ABO scans the input program modules instead of performing optimization. No output modules are produced.

The REPLACE=N option can affect whether scanning is performed:

- If REPLACE=N is specified on the BOPT directive and the output module on the OUT option already exists, scanning of the module on the IN option is bypassed.
- If REPLACE=N is specified on the IEFOPZ directive, scanning of the member of the OLD data set is bypassed if the member of the NEW data set already exists.

If you exclude the OUT option on the BOPT directive, scanning is always performed regardless of the value of the REPLACE option.

In the scanning mode, the optimizer checks the input and output program modules, and lists the CSECTs in the modules. Scanning output is written to the OPTLOG DD.

You can use SCAN=Y to test the SYSIN setup or to see what modules are present and their contents. If the program is ineligible for optimization due to the original compiler used, this is also indicated in the scanning output.

For more information, see "The log file for scanning" on page 50.

Comments

A comment is specified by starting with the (#) character.

When using the (#) character, the following rules apply:

- If the first non-blank character on a line of the SYSIN file is the (#) character, the rest of the line is ignored.
- If ABO sees a (#) character, which is preceded by a blank, on an input line of SYSIN file, the rest of the line is ignored.

Example 1

```
#BOPT IN=DD:SYSBIN OUT=SYSBOUT
  #BOPT IN=SYSBIN OUT=SYSBOUT
```

In example 1, both of the BOPT directives are commented out. On the first line, the # character is in column 1 and the rest of the line is ignored. On the second line, the first non-blank character is the # character and the rest of the line is ignored.

Example 2

```
# Optimizing all members of library
  # Note: we don't compile a member optimized earlier and found in the OUT dataset
BOPT IN=SYSBIN(*) OUT=SYSBOUT REPLACE=N
```

Example 2 shows adding two full lines of informational comments to the SYSIN file.

Example 3

```
BOPT IN=SYSBIN(*) OUT=SYSBOUT  #
```

In example 3, the # character at the end of a line is ignored.

Example 4

```
BOPT IN=SYSBIN(*) OUT=SYSBOUT  #optimizing library files
```

Example 4 shows adding a comment after a directive.

Line continuation

Line continuation in a SYSIN file is specified by the (+) or (-) continuation character, which indicates the next line should be read as if it were a part of the previous line. Line continuation is not required but can be used to break up long lines to simplify editing or to ease consumption by other tooling that may have line length restrictions.

The (+) or (-) continuation character must be the last non-blank and non-comment character on a line. When using the continuation characters, the following rules apply:

- The (-) character can only be used to continue a line following complete options, directives, or specifiers. Line continuation begins at column 1 on the next line. The (-) character must be preceded by one or more blanks.
- The (+) character can be used to continue a line following complete options, directives, or specifiers, or in the middle of options, directives, or specifiers. Line continuation begins with the first non-blank character on the next line. When continuing complete options, directives, or specifiers, the (+) character must be preceded by one or more blanks.
- Blanks that precede the line continuation character will be included in the concatenated line.
- The comment character (#) takes precedence over the line continuation character. If a line continuation character is part of a comment, it will be ignored as part of that comment and not continue the comment.

An error message might be issued if the continuation character is in an unexpected position.

These examples show the JCL using the BOPT optimizer directive. The examples are not full examples. They are intended to reflect what the user should specify in the SYSIN file. For basic JCL configuration, see Appendix A, “JCL sample,” on page 91.

Example 1

```
//SYSIN  DD *
ARCH=13 -
ALLOW=UNRESEXE
BOPT IN=HLQ.LOAD.APPXYZ1.ORIG(*) -
OUT=HLQ.LOAD.APPXYZ1.ABO
```

This example is interpreted by ABO as the following:

```
//SYSIN  DD *
ARCH=13 ALLOW=UNRESEXE
BOPT IN=HLQ.LOAD.APPXYZ1.ORIG(*) OUT=HLQ.LOAD.APPXYZ1.ABO
```

Note that blanks preceding the continuation character are included in the concatenated string in all cases.

Example 2

```
//SYSIN DD *
                                       AR+
                                       CH=12 +
                                                  ALLOW=UNRESEXE
BOPT IN=HLQ.LOAD.APPXYZ1.OR+
IG(*) -
OUT=HLQ.LOAD.APPXYZ1.ABO
```

This example shows how the (+) character allows continuing wherever desired on the next line (at the first non-blank) instead of only at column 1, and how the (+) character can be used to continue an option, directive, or specifier that is not complete. This example is interpreted by ABO as the following:

```
//SYSIN  DD *
ARCH=12 ALLOW=UNRESEXE
BOPT IN=HLQ.LOAD.APPXYZ1.ORIG(*) OUT=HLQ.LOAD.APPXYZ1.ABO
```

Example 3

```
//SYSOPTF  DD *
       # some comment +                                                  (1)
                       ARCH=12    +   #other comment -                  (2)
                                 ALLOW=UNRESEXE                        (3)
BOPT IN=HLQ.LOAD.APPXYZ1.ORIG(*) OUT=HLQ.LOAD.APPXYZ1.ABO               (4)
```

This example is processed by ABO as the following:

```
//SYSOPTF  DD *
ARCH=12 ALLOW=UNRESEXE
BOPT IN=HLQ.LOAD.APPXYZ1.ORIG(*) OUT=HLQ.LOAD.APPXYZ1.ABO
```

The (+) character on line 1 and the (-) character on line 2 are ignored because they are part of comments. The (+) character on line 2 is treated as line continuation character because it is the last non-comment and non-blank character on the line.

JCL examples

You can use the job control language (JCL) statements that are shown in these examples to process compiled COBOL modules with ABO.

Specifying optimization with BOPT

These examples show the JCL using the BOPT optimizer directive.

The examples in this section are not full examples. They are intended to reflect what the user should specify in the SYSIN file. For basic JCL configuration, see Appendix A, “JCL sample,” on page 91.

Example 1. Specifying I/O modules using data set names

In this example, input and output modules are determined by the BOPT line. Input and output data set names are given precisely rather than specified in ddnames.

```
...
//SYSIN DD *
  ARCH=12
  BOPT IN=HLQ.IN.LOAD(MEM1) OUT=HLQ.OUT.LOAD(MEM1)
```

Example 2. Specifying I/O modules using ddnames

In this example, input and output modules are specified using ddnames.

```
...
//SYSBIN DD DSN=HLQ.IN.LOAD,DISP=SHR
//SYSBOUT DD DSN=HLQ.OUT.LOAD,DISP=SHR
//SYSIN DD *
  ARCH=12
  BOPT IN=DD:SYSBIN(MEM1) OUT=DD:SYSBOUT(MEM1)
```

Example 3. Specifying I/O modules using HFS paths

In this example, the input and output modules are specified using HFS paths. You must use fully qualified HFS paths that start with a slash (/).

```
...
//SYSIN DD *
  ARCH=12
  BOPT IN=/home/user1/a.out OUT=/home/user1/a.out.opt
```

Example 4. Specifying an input module and omitting the output member specifier

In this example, an input module is specified and the output member specifier is omitted. The member name in the output PDS(E) for the optimized module will be the same name as the specified member of the input PDS(E).

```
...
//SYSBIN DD DSN=HLQ.IN.LOAD,DISP=SHR
//SYSBOUT DD DSN=HLQ.OUT.LOAD,DISP=SHR
//SYSIN DD *
  ARCH=12
  BOPT IN=DD:SYSBIN(MEM1) OUT=DD:SYSBOUT
```

Example 5. Specifying multiple input modules using an expression

In this example, multiple input modules are specified using an expression. The member name in the output PDS(E) for an optimized module will be the same name as the corresponding member of the input PDS(E).

To include modules with names that start with MEM:

```
...
//SYSBIN DD DSN=HLQ.IN.LOAD,DISP=SHR
//SYSBOUT DD DSN=HLQ.OUT.LOAD,DISP=SHR
//SYSIN DD *
  ARCH=12
  BOPT IN=DD:SYSBIN(MEM*) OUT=DD:SYSBOUT
```

To exclude a single module named MEMA:

```
...
//SYSBIN DD DSN=HLQ.IN.LOAD,DISP=SHR
//SYSBOUT DD DSN=HLQ.OUT.LOAD,DISP=SHR
//SYSIN DD *
  ARCH=12
  BOPT IN=DD:SYSBIN(<>MEMA) OUT=DD:SYSBOUT
```

To exclude all members whose names begin with MEMB:

```
...
//SYSBIN DD DSN=HLQ.IN.LOAD,DISP=SHR
//SYSBOUT DD DSN=HLQ.OUT.LOAD,DISP=SHR
//SYSIN DD *
  ARCH=12
  BOPT IN=DD:SYSBIN(<>MEMB*) OUT=DD:SYSBOUT
```

To exclude members named SUB1 and SUB2:

```
...
//SYSBIN DD DSN=HLQ.IN.LOAD,DISP=SHR
//SYSBOUT DD DSN=HLQ.OUT.LOAD,DISP=SHR
//SYSIN DD *
  ARCH=12
  BOPT IN=DD:SYSBIN(<>SUB1:SUB2) OUT=DD:SYSBOUT
```

Example 6. Specifying multiple input modules by using multiple BOPT optimizer directives

In this example, multiple input modules are specified using multiple BOPT optimizer directives:

```
...
//SYSIN DD *
  ARCH=12
  BOPT IN=HLQ.IN.LOAD(MEM1) OUT=HLQ.OUT.LOAD(MEM1)
  BOPT IN=HLQ.IN.LOAD(MEM5) OUT=HLQ.OUT.LOAD(MEM5)
```

Example 7. Bypassing optimizations or scans with the REPLACE option

In this example, the REPLACE option is used. REPLACE=N bypasses optimization or scanning if the associated output modules already exist. In this example, the optimizer performs optimization in the first and third BOPT directives. The optimizer bypasses optimization in the second BOPT directive and scanning in the last BOPT directive REPLACE=N is specified, and the output modules were created in the first and third BOPT directives.

```
...
//SYSIN DD *
  ARCH=12
  BOPT IN=HLQ.IN.LOAD(MEM1) OUT=HLQ.OUT.LOAD(MEM1) REPLACE=Y
  BOPT IN=HLQ.IN.LOAD(MEM1) OUT=HLQ.OUT.LOAD(MEM1) REPLACE=N
  BOPT IN=HLQ.IN.LOAD(*) OUT=HLQ.OUT.LOAD REPLACE=Y
  BOPT IN=HLQ.IN.LOAD(*) OUT=HLQ.OUT.LOAD SCAN=Y REPLACE=N
```

Example 8. Specifying global and local optimizer options

In the example, global and local options are used. ARCH=12 and REPLACE=N apply to the first BOPT directive; ARCH=13 and REPLACE=Y apply to the second BOPT directive.

```
...
//SYSBOUT DD DSN=HLQ.OUT.LOAD,DISP=SHR
//SYSIN DD *
  ARCH=12 REPLACE=N
  BOPT IN=HLQ.IN.LOAD(MEM1) OUT=DD:SYSBOUT(MEM1) ARCH=13
  BOPT IN=HLQ.IN.LOAD(MEM1) OUT=DD:SYSBOUT(MEM1) REPLACE=Y
```

Example 9. Switching to scan mode by using global SCAN

The following example changes the mode from optimizing input modules to scanning input modules using the global option SCAN. In this scan mode, output modules are not written. The OUT option is processed, for example for correct syntax, but otherwise ignored. For details about the SCAN option, see “SCAN” on page 33.

The optimizer reverts back to the optimization mode for the second BOPT directive.

```
...
//SYSBOUT DD DSN=HLQ.OUT.LOAD,DISP=SHR
//SYSIN DD *
  ARCH=12 SCAN=Y
  BOPT IN=HLQ.IN.LOAD(MEM1) OUT=DD:SYSBOUT(MEM1)
  BOPT IN=HLQ.IN.LOAD(MEM1) OUT=DD:SYSBOUT(MEM1) SCAN=N
```

Example 10. Specifying ALLOW options

This example shows a global setting of ALLOW=NOUNRESEXE being specified with a local override for the second BOPT directive.

The first (MEM1) and third (MEM3) BOPT directives are done using ALLOW=NOUNRESEXE meaning that only fully bound program modules will be accepted. If a partially bound program module is encountered then the BOZ1494S message will be issued.

The second (MEM2) BOPT directive is done using ALLOW=UNRESEXE so both fully and partially bound program modules are accepted as input.

```
...
//SYSBOUT DD DSN=HLQ.OUT.LOAD,DISP=SHR
//SYSIN DD *
  ARCH=12 ALLOW=NOUNRESEXE
  BOPT IN=HLQ.IN.LOAD(MEM1) OUT=DD:SYSBOUT(MEM1)
  BOPT IN=HLQ.IN.LOAD(MEM2) OUT=DD:SYSBOUT(MEM2) ALLOW=UNRESEXE
  BOPT IN=HLQ.IN.LOAD(MEM3) OUT=DD:SYSBOUT(MEM3)
```

Specifying optimization with IEFOPZ

These examples show the JCL using the IEFOPZ optimizer directive.

The examples in this section are not full examples. They are intended to reflect what the user should specify in the SYSIN file. For basic JCL configuration, see Appendix A, “JCL sample,” on page 91.

Example 1. Single ARCH configuration

This example shows the minimal JCL for running ABO with the IEFOPZ directive.

You can use this JCL when your IEFOPZ configuration includes only one ARCH level.

```
...
//SYSIN DD *
  IEFOPZ
```

Example 2. Multiple ARCH configuration

If your IEFOPZ configuration includes more than one ARCH level, specify separate IEFOPZ directives lines to avoid listing file name collisions.

```
...
//SYSIN DD *
  IEFOPZ SEL_ARCH=13 LIST=HLQ.OUT1.ARCH11.LIST
  IEFOPZ SEL_ARCH=12 LIST=HLQ.OUT1.ARCH12.LIST
```

Example 3. Restricting optimization using the SEL_STATE and SEL_ARCH selectors

The following example produces optimized modules for those OLDNEW mappings that are marked as INACTIVE. As with example 2, if mappings have more than one ARCH level, the LIST option is used to avoid listing file name collisions.

```
...
//SYSIN DD *
  IEFOPZ SEL_STATE=INACTIVE SEL_ARCH=12 LIST=HLQ.OUT.ARCH12.LIST
  IEFOPZ SEL_STATE=INACTIVE SEL_ARCH=13 LIST=HLQ.OUT.ARCH13.LIST
```

Example 4. Restricting optimization using the SEL_OLD selector

The following example produces optimized modules for members of OLD data sets that match given patterns.

In the first line, optimized modules will be produced for all members of HLQ.IN.LOAD.

In the second line, optimized modules will be produced for data set members that match M* in OLD data sets matching HLQ.IN.*. REPLACE=N is specified on the second line to avoid re-optimizing modules from the first line.

```
...
//SYSIN DD *
  IEFOPZ SEL_OLD=HLQ.IN.LOAD
  IEFOPZ SEL_OLD=HLQ.IN*.*(M*) REPLACE=N
```

Recommended settings for the z/OS JCL REGION and JCL MEMLIMIT parameters

ABO optimization techniques

To generate high performing optimized modules, beyond those possible from Enterprise COBOL V4 and earlier, ABO performs advanced analysis and uses code optimization techniques that require substantial machine resources.

In addition, since ABO is easily invoked on many compiled programs at once, the overall resources can be high due to the amount of processing requested. An entire data set, potentially composed of many modules, where each module can itself contain many compiled programs (CSECTs), can be optimized in bulk using ABO. Therefore, the total resources required by ABO for optimizing hundreds or thousands of compiled programs at once will be higher compared to a compilation process that is operating on a single source file at a time.

The time and memory required to optimize a module is based on the following factors:

- The number of CSECTs in the module.
- The complexity of each CSECT. Complexity is impacted by both the size of the compiled PROCEDURE DIVISION statements and also by the size of the input program's DATA DIVISION.

Setting the z/OS JCL REGION and JCL MEMLIMIT parameters appropriately

As shown in the Appendix A, “JCL sample,” on page 91, the JCL REGION parameter should be set to 0M to allow ABO the memory it requires to operate.

ABO will use storage above the 2 GB BAR to optimize large CSECTs. This means that the z/OS MEMLIMIT parameter should be set to a high enough value in order to allow ABO processing to complete successfully.

The recommended REGION setting of 0M will set MEMLIMIT to NOLIMIT. However, this NOLIMIT value may have been overriden to a lower value by a MEMLIMIT setting on JOB or EXEC statements or by the exit routine IEFUSI. The amount of storage required by ABO will depend on the number and size of CSECTs being optimized.

If MEMLIMIT=NOLIMIT has been overridden to a lower value and this MEMLIMIT setting is not high enough, you may get one of these ABO messages:

```
BOZ1145U Insufficient memory in the compiler to continue compilation.
BOZ1428U Insufficient memory encountered during binder API "&1": return code=&2
        reason code=&3. Terminating optimizer.
BOZ1449U Unhandled out of memory exception
```

A MEMLIMIT setting of 10GB or more may be required for optimizing very large CSECTs, or a high number of smaller CSECTs. If any of the ABO BOZ1145U, BOZ1428U or BOZ1449U messages are encountered then increase the MEMLIMIT setting to a higher value.

For more information on JCL REGION and JCL MEMLIMIT parameters, see the z/OS MVS Initialization and Tuning Reference and the z/OS MVS Initialization and Tuning Guide.

Specifying the language to be used for ABO messages

The CEEOPTS DD is used to specify the language for ABO produced messages.

By default, messages are in English. To specify that you want Japanese messages, add the following code to your JCL:

```
//CEEOPTS DD *
          NATLANG(JPN)
/*
```

Invoking ABO from TSO, REXX and assembler code

This section describes how to invoke ABO from TSO, REXX and the assembler.

Optimizing under TSO

Under TSO, you can use TSO commands, command lists (CLISTs), REXX execs, or ISPF to optimize COBOL programs using traditional MVS data sets. You can use TSO commands or REXX execs to optimize programs using z/OS UNIX files.

With each method, you need to allocate the data sets and request the optimization following these steps:

1. Use the ALLOCATE command to allocate data sets. You can allocate data sets in any order. However, you must allocate all needed data sets before you start to optimize.
2. Provide the optimizer parameters within your SYSIN data set.
3. Use the CALL command at the READY prompt to request optimization:
   `CALL 'hlqboz.SBOZMOD1(BOZOPT)'`

You can specify the ALLOCATE and CALL commands on the TSO command line, or, if you are not using z/OS UNIX files, you can include them in a CLIST.

You can allocate z/OS UNIX files for all the optimizer data sets if they are not PDS or PDSE libraries. For example, if ABO parameters are stored in the UNIX file `/u/myu/abo.parms`, then the ALLOCATE statements have the following form:

```
  Allocate File(SYSIN) Path('/u/myu/abo.parms') Pathopts(ORDONLY) Filedata(TEXT)
```

ALLOCATE and CALL for optimizing under TSO

The following example shows how to specify the ALLOCATE and CALL commands when you are optimizing under TSO. Notice that all the files can be either new or existing ones, except the input files SYSIN and SYSBIN that must exist before optimization starts.

```
 [READY]
FREE F(SYSPRINT SYSIN OPTLOG OPTERR CEEDUMP SYSBIN
SYSBOUT)
 [READY]
ALLOC F(SYSPRINT) DA(ABO.LISTING) SPACE(10,10) CYL NEW CATALOG RECFM(V B) LRECL(512)
BLKSIZE(27998) DSORG(PS)
 [READY]
ALLOC FI(OPTLOG) DA(ABO.OPTLOG) SPACE(10,10) CYL NEW CATALOG RECFM(V B) LRECL(512)
BLKSIZE(27998) DSORG(PS)
 [READY]
ALLOC FI(OPTERR) DA(ABO.OPTERR) SPACE(10,10) CYL NEW CATALOG RECFM(V B) LRECL(512)
BLKSIZE(27998) DSORG(PS)
 [READY]
ALLOC FI(CEEDUMP) DA(ABO.CEEDUMP) SPACE(50,10) CYL NEW CATALOG RECFM(F B) LRECL(133)
BLKSIZE(27930) DSORG(PS)
 [READY]
ALLOC FI(SYSIN) DA(ABO.SYSIN) SHR          /* supply ABO parameters within SYSIN file
*/
[READY]
ALLOC FI(SYSBIN) DA(IN.LOAD) SHR           /* supply COBOL load library to be optimized */
 [READY]
ALLOC FI(SYSBOUT) DA(OUT.LOAD) SHR         /* supply load library for optimized load modules */
 [READY]
CALL 'hlqboz.SBOZMOD1(BOZOPT)'
 [READY]
FREE F(SYSPRINT SYSIN OPTLOG OPTERR CEEDUMP SYSBIN SYSBOUT)
```

CLIST for optimizing under TSO

The following example shows a CLIST for optimizing under TSO:

```
PROC 1 HLQBOZ
FREE F(SYSPRINT SYSIN OPTLOG OPTERR CEEDUMP SYSBIN
SYSBOUT)
ALLOC FI(SYSPRINT) DA(ABO.LISTING) SHR
ALLOC FI(SYSIN) UNIT(SYSDA) SPACE(1,0) TRACKS NEW +
RECFM(F B) LRECL(80) BLKSIZE(800) DSORG(PS)
OPENFILE SYSIN OUTPUT
SET SYSIN = &STR(BOPT IN=DD:SYSBIN(MEMBER) OUT=DD:SYSBOUT(MEMBER))
PUTFILE SYSIN
CLOSFILE SYSIN
ALLOC FI(OPTLOG) DA(ABO.OPTLOG) SHR
ALLOC FI(OPTERR) DA(ABO.OPTERR) SHR
ALLOC FI(CEEDUMP) DA(ABO.CEEDUMP) SHR
ALLOC FI(SYSBIN) DA(IN.LOAD) SHR
ALLOC FI(SYSBOUT) DA(OUT.LOAD) SHR
CALL '&HLQBOZ..SBOZMOD1(BOZOPT)'
FREE F(SYSIN SYSPRINT OPTLOG OPTERR CEEDUMP SYSBIN SYSBOUT)
```

REXX for optimizing under TSO

The following example shows a REXX for optimizing under TSO:

```
/*=========================>> REXX <<================================*/
  Parse Arg hlqboz .                              /* get argument  */
   IF HLQBOZ = '' THEN DO
   SAY 'HLQBOZ ARGUMENT MISSING'
   EXIT
   END
   ADDRESS TSO
   msgstat = MSG("OFF")
```

```
   "FREE FILE (SYSIN SYSPRINT)"
   "ALLOC FI(SYSPRINT) DA(ABO.LISTING) SHR"
   "ALLOC FI(OPTLOG) DA(ABO.OPTLOG) SHR"
   "ALLOC FI(OPTERR) DA(ABO.OPTERR) SHR"
   "ALLOC FI(CEEDUMP) DA(ABO.CEEDUMP) SHR"
   "ALLOC FI(SYSBIN) DA(IN.LOAD) SHR"
   "ALLOC FI(SYSBOUT) DA(OUT.LOAD) SHR"
   "ALLOC FI(SYSIN) NEW CYL SPACE(1,1) RECFM(F B)",
   " LRECL(80) BLKSIZE(800) DSORG(PS)"
   line.1 = 'ARCH=12'
   line.2 = 'BOPT IN=DD:SYSBIN(MEMBER) OUT=DD:SYSBOUT(MEMBER)'
   "EXECIO  2  DISKW  SYSIN  (STEM line. FINIS"
   "CALL '"hlqboz".SBOZMOD1(BOZOPT)'"
   "FREE FI(OPTLOG OPTERR CEEDUMP SYSPRINT SYSBIN SYSBOUT SYSIN)"
   msgstat = MSG("ON")
```

REXX for optimizing under TSO batch, directing OPTLOG and SYSPRINT output to the library members.

Sometimes you may need to optimize every member of the entire load library or even more than one load library without having to type multiple BOPT statements.

The following example shows a TSO batch job used to run ABOMEMBS REXX located in the data set referenced by the SYSEXEC DD name. ABOMEMBS goes through the SYSBIN data sets concatenation and individually optimizes every data set member found in that concatenation.

```
//OPTMEMBS EXEC PGM=IKJEFT01,PARM='ABOMEMBS',
REGION=0M
//STEPLIB  DD  DISP=SHR,DSN= hlqboz.SBOZMOD1
//SYSEXEC  DD  DISP=SHR,DSN=hlq.CLIST         /* supply ABOMEMBS REXX member within SYSEXEC
library */
//SYSTSPRT DD  SYSOUT=*,DCB=(LRECL=132,RECFM=FBA,BLKSIZE=1320)
//SYSTSIN  DD   DUMMY
//OPTERR   DD   SYSOUT=*
//CEEDUMP  DD   SYSOUT=*
//SYSBIN   DD  DISP=SHR,DSN=hlq.INLOAD1       /* supply one or more COBOL load libraries to be
optimized */
//         DD  DISP=SHR,DSN=hlq.INLOAD2
//         DD  DISP=SHR,DSN=hlq.INLOAD3
//SYSBOUT  DD  DISP=SHR,DSN=hlq.OUTLOAD
//OPTLOG   DD  DISP=(,CATLG),DSN=hlq.OPTLOG,UNIT=3390,    /* supply new or existing PDS/PDSE or
SEQ file */
//         SPACE=(CYL,(5,5,20)),DSNTYPE=LIBRARY,
//                DCB=(RECFM=VB,LRECL=512,BLKSIZE=0)
//SYSPRINT DD  DISP=(,CATLG),DSN=hlq.SYSPRINT,UNIT=3390,  /* supply new or existing PDS/PDSE or
SEQ file */
//                SPACE=(CYL,(5,5,20)),DSNTYPE=LIBRARY,
//                DCB=(RECFM=VB,LRECL=512,BLKSIZE=0)
//SYSIN    DD *
  ARCH=12
  SCAN=N
# optionally put BOPTs below this line for specific library members optimization
  BOPT IN=hlq.INLOAD7(A*) OUT=hlq.OUTLOAD LOG=hlq.OPTLOG LIST=hlq.SYSPRINT
/*
```

For every member found in the above SYSBIN DD concatenation, ABOMEMBS invokes ABO with SYSIN, OPTLOG, SYSBIN and SYSPRINT files individually allocated for that member.

For example, for member MEM1 located in the hlq.INLOAD1 data set, it will allocate the SYSBIN file hlq.INLOAD1, reallocate the OPTLOG file hlq.OPTLOG(MEM1) and the SYSPRINT file hlq.SYSPRINT(MEM1), if the OPTLOG and SYSPRINT files allocated in the above JCL are PDS or PDSE data sets. Otherwise, it will keep using JCL allocation of these files.

To construct the SYSIN file for MEM1, it will use all the ABO parameters listed in the above JCL SYSIN file down to the first BOPT statement, and then append it with internally generated BOPT statement. Here is an example of the SYSIN file for MEM1:

```
==================================================
ARCH=12
SCAN=N
BOPT IN=DD:SYSBIN(MEM1) OUT=DD:SYSBOUT(MEM1)
==================================================
```

After all the individual SYSIN, OPTLOG, SYSPRINT and SYSBIN files are allocated, ABO is invoked to optimize MEM1.

ABOMEMBS exec repeats this process for every member in the SYSBIN concatenation. If the BOPT statement is present in the above JCL SYSIN file, after all the members in the SYSBIN concatenation are optimized, ABOMEMBS reallocates the OPTLOG and SYSPRINT files to SYSOUT, reallocates the SYSIN file containing all the same statement as the original JCL SYSIN file, and invokes ABO one more time to proceed with the parameters explicitly specified in the SYSIN file.

This approach can be useful to optimize all the library members just by including it into the SYSBIN concatenation, without having to specify any additional BOPT explicitly, and optionally, including an additional BOPT statement when you need to optimize some members of the library only. The JCL cards in the above JCL can be in any order.

ABOMEMBS generates the following SYSTSPRT output:

```
============== Processing hlq.INLOAD1 data set ========================
Member ADD10 processed, rc=4
Member CALLEE processed, rc=0
Member CALLEE1 processed, rc=0
…
============== Processing hlq.INLOAD2 data set ========================
Member COBPGM processed, rc=0
Member DJSIEV85 processed, rc=0
Member IAMASM processed, rc=4
…
============== Processing hlq.INLOAD3 data set ========================
Member TEMPNAM6 processed, rc=12
Member TEMPNAM7 processed, rc=0
Member TEMPNAM8 processed, rc=0
…
============== Processing SYSIN statements ============================
ARCH=12
SCAN=N
BOPT IN=hlq.INLOAD7(A*) OUT=hlq.OUTLOAD LOG=hlq.OPTLOG LIST=hlq.SYSPRINT
=======================================================================
READY
END
```

The following is an example of the ABOMEMBS REXX source code:

```
 BROWSE    hlq.CLIST(ABOMEMBS)
 Command ===>
****************************************************** Top of Data
/*============================>> REXX <<================================*/
  Parse Arg

   X = LISTDSI(STEPLIB FILE)
   If x > 0 Then Do
      Say 'Check STEPLIB allocation' ; Exit
   End
   optimizer = "'"||SYSDSNAME||'(BOZOPT)'||"'"
   flaglog = 0  /* assume OPTLOG is sequential */
   X = LISTDSI(OPTLOG FILE)
   IF x = 0 & SYSDSORG = 'PO' Then Do
      optlogds = SYSDSNAME ; flaglog = 1
   End
   flaglist = 0 /* assume SYSPRINT is sequential */
   X = LISTDSI(SYSPRINT FILE)
   IF x = 0 & SYSDSORG = 'PO' Then Do
      sysprtds = SYSDSNAME ; flaglist = 1
   End
   "EXECIO  *  DISKR  SYSIN  (STEM line. FINIS" /* read in SYSIN parameters into
line. array */
   n = line.0 + 1     /* assume no explicit BOPT specified. n is per-member BOPT
line number */
   Do l = 1 To
line.0
      PARSE UPPER VALUE line.l WITH
line.l
      If POS('BOPT ',line.l) > 0 Then
Do
```

```
          n = l ; flagbopt = 1 ; leave  /* leave when first BOPT found
*/

End

      linea.l = line.l /* copy all lines before first BOPT into linea. array
*/

End

   X = LISTDSI(SYSBIN
FILE)
   If x > 0 Then
Do
       Say 'Check SYSBIN allocation' ;
Exit

End

   x =
outtrap(concatl.)
   lista
status
   x =
outtrap(off)
   /* i2 is a line with last dsn in SYSBIN concatenation
*/
   i2 = concatl.0 - 1          /* assume that SYSBIN concatenation is a last in JCL
*/
   Do i = 1 To
concatl.0
      If substr(concatl.i,3,6) = 'SYSBIN' Then
Do
         i1 = i - 1 ; leave    /* i1 line with first dsn in concatenation
*/

End

   End
   k = i + 2
   If k < concatl.0 Then Do
      Do i = k To concatl.0 by 2
         If substr(concatl.i,3,1) <> ' ' Then Do
            i2 = i - 3 ; leave /* i2 line with last dsn in concatenation */
         End
      End
   End
   Else i2 = i1         /* when SYSBIN DD with a single data set is a last JCL card
*/

 msgstat = MSG("OFF")

   Do i = i1 To i2 by 2   /* loop trough SYSBIN concatenation */
      sysbinds = strip(concatl.i)
      Say '============== Processing 'sysbinds' data set =============='
      x=outtrap('row.')
         Address TSO "LISTDS '"sysbinds"' members"
      x=outtrap('off')
      If row.0 < 6 Then Do
         Say 'Check SYSBIN file' ; Exit
      End
     DO J = 7 TO row.0    /* loop trough member list */
        PARSE VALUE row.J WITH memn alias
        "FREE FI(SYSIN)"
        "ALLOC FI(SYSIN) NEW CYL SPACE(1,1) RECFM(F B)",
        " LRECL(80) BLKSIZE(800) DSORG(PS)"
        linea.n = 'BOPT IN=DD:SYSBIN('||memn||') ',
        'OUT=DD:SYSBOUT('||memn||')'
        "EXECIO  *  DISKW  SYSIN  (STEM linea. FINIS"
        If flaglog = 1 Then Do
           "FREE FI(OPTLOG)"
           optlogm = "'"||optlogds||'('||memn||')'||"'"
```

```
            "ALLOC FI(OPTLOG) DA("optlogm") SHR"
         End
         If flaglist = 1 Then Do
            "FREE FI(SYSPRINT)"
            sysprtm = "'"||sysprtds||'('||memn||')'||"'"
            "ALLOC FI(SYSPRINT) DA("sysprtm") SHR"
         End
         "FREE FI(SYSBIN)"
         "ALLOC FI(SYSBIN) DA('"sysbinds"') SHR"
         "CALL "optimizer   /* optimize member */
         Say 'Member 'memn' processed, rc='rc
      END   /* end of member list loop */
   End     /* end of concatenation list loop */
   If flagbopt = 1 Then Do
       "FREE FI(SYSIN)"
       "ALLOC FI(SYSIN) NEW CYL SPACE(1,1) RECFM(F B)",
       " LRECL(80) BLKSIZE(800) DSORG(PS)"
       "EXECIO  *  DISKW  SYSIN  (STEM line. FINIS"
       "FREE FI(OPTLOG)"
       "ALLOC FI(OPTLOG) SYSOUT"
       "FREE FI(SYSPRINT)"
       "ALLOC FI(SYSPRINT) SYSOUT"
       "CALL "optimizer      /* optimize with BOPTs provided in SYSIN */
       Say '============== Processing SYSIN statements =============='
       Do i = 1 To line.0
          Say line.i
       End
       Say '============================================================'
   End
   msgstat = MSG("ON")
EXIT
```

Starting the optimizer from an assembler program

You can invoke ABO programmatically from within an HLASM program.

Before you start to optimize, complete these steps:

1. Allocate all the needed data sets, either by using dynamic allocation within your assembler program or specifying the DD cards in the job JCL used for your assembler program invocation. The following DD names must be allocated: SYSPRINT, SYSBIN, SYSBOUT, SYSIN, OPTLOG, OPTERR, CEEDUMP.
2. Provide the ABO parameters within your SYSIN data set.

You can start ABO from within an assembler program by using the LINKX or ATTACHX macro because those two are 64 bit mode compatible and ABO is running in AMODE 64.

The following is an example of the LINKX macro in list form:

```
  symbol {LINKX} EP=BOZOPT,AMODE64OK=YES,PLIST8=YES,SF=L
```

The following is an example of the LINKX macro in execute form:

```
  LINKX EP=BOZOPT,AMODE64OK=YES,PARAM=(addr),PLIST8=YES,       *
        MF=(E,#LINKX),SF=(E,#LINK2)
```

where *#* is used as a prefix symbol.

EP
Specifies the symbolic name of ABO.

PARAM
Specifies the address parameters list to be passed from the assembler program to ABO. In the example, *addr* can be any value because it's ignored by BOZOPT program which directly reads parameters supplied in the SYSIN file.

PLIST8=YES
Defines the size of the parameter list entries for a parameter list to be built by LINKX based on the PARAM keyword, as an 8-bytes-per-entry parameter list.

AMODE64OK=YES
Indicates that the system is to accept an attempt to link to an AMODE 64 target routine from an AMODE 24 or AMODE 31 routine.

SF=L
Specifies the list form of the LINKX macro.

When ABO completes processing, it puts a return code in register 15.

Assembler program starting the optimizer

The following example shows the ABO invocation from an assembler program:

```
//JOBCARD JOB
//*
//ASMHCL  PROC MAC='SYS1.MACLIB',MAC1='SYS1.MODGEN',U=3390,
//             MAC2='SYS1.MACLIB'
//*-------------------------------------------------------------------***
//*            A S M H C L     H-ASSEMBLER                            ***
//*            IBM-PROCEDURE: COMPILE + LINK             ***
//*-------------------------------------------------------------------***
//ASM      EXEC PGM=ASMA90,PARM=OBJECT,REGION=0M
//SYSLIB   DD   DSN=&MAC,DISP=SHR
//         DD   DSN=&MAC1,DISP=SHR
//         DD   DSN=&MAC2,DISP=SHR
//SYSUT1   DD   DSN=&&SYSUT1,UNIT=&U,SPACE=(TRK,(60,45))
//SYSPRINT DD   SYSOUT=*,DCB=BLKSIZE=1089
//SYSPUNCH DD   DUMMY
//SYSLIN   DD   DSN=&&OBJSET,UNIT=&U,SPACE=(80,(2000,500)),
//         DISP=(MOD,PASS)
//LKED     EXEC PGM=HEWLH096,
//         PARM='XREF,LET,LIST,AC=0,FILL=NONE',
//         COND=(8,LT,ASM)
//SYSLIN   DD   DSN=&&OBJSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
//SYSLMOD  DD   DSN=hlq.CALLABO.LOAD,DISP=SHR
//SYSUT1   DD   DSN=&&SYSUT1,UNIT=(&U,SEP=(SYSLIN,SYSLMOD)),
//         SPACE=(1024,(50,20))
//SYSPRINT DD   SYSOUT=*
//         PEND
//*
//         EXEC ASMHCL
//ASM.SYSIN  DD
*
         YREGS
CALLABO  CSECT
CALLABO  AMODE 31
CALLABO  RMODE ANY
         STM   R14,R12,12(R13)
         USING CALLABO,R12
         LR    R12,R15      LOAD BASE REGISTER
         LA    R9,SAVE      POINT TO CURRENT SAVEAREA
         ST    R9,8(,R13)   A(CURRENT_SA) IN OLD_SA
         ST    R13,4(,R9)   A(OLD_SA) IN CURRENT_SA
         LR    R13,R9       R13=A(CURRENT_SAVEAREA)
*-----------------------------------------------------------------*
         LLGTR R12,R12             allow 64-bit
addressability
         BAL   R14,LINKOPT       invoke BOZOPT
*-----------------------------------------------------------------*
DONE     DS    0H
         L     R13,4(,R13)
         ST    R15,16(,R13)
         LM    R14,R12,12(R13)
         BR    R14
*-----------------------------------------------------------------*
LINKOPT  DS    0H
         XR    R15,R15              nullify parm addr, BOZOPT ignores
it
         LLGTR R15,R15
         SAM64
         LINKX EP=BOZOPT,AMODE64OK=YES,PARAM=((R15)),PLIST8=YES,       *
               MF=(E,#LINKX),SF=(E,#LINK2)
         SAM31
```

```
         B     DONE
*-------------------------------------------------------------------*
         DS    0F
SAVE     DC    XL72'00'
#LINKX   LINKX EP=BOZOPT,AMODE64OK=YES,PLIST8=YES,SF=L
#LINK2   LINKX EP=BOZOPT,AMODE64OK=YES,PLIST8=YES,SF=L
         LTORG
         END
//LKED.SYSIN DD *
   PAGE CALLABO
   NAME CALLABO(R)
//
```

The following is a sample JCL to run the CALLABO program:

```
//OPT      EXEC PGM=CALLABO,REGION=0M
//STEPLIB  DD DSN=hlqboz.SBOZMOD1,DISP=SHR
//         DD DSN=hlq.CALLABO.LOAD,DISP=SHR
//OPTLOG   DD SYSOUT=*
//OPTERR   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSBIN   DD DSN=hlq.IN.LOAD,DISP=SHR
//SYSBOUT  DD DSN=hlq.OUT.LOAD,DISP=SHR
//SYSIN    DD *
# supply ABO directives down this line
  BOPT IN=DD:SYSBIN(*) OUT=DD:SYSBOUT
```

Chapter 6. Understanding output from the optimization process

Log files

IBM Automatic Binary Optimizer for z/OS generates a log file that helps you identify and resolve problems.

You can diagnose problems at optimize time by inspecting the log file that is generated unconditionally by ABO into the OPTLOG DD (see Table 7 on page 17). The log file contains diagnostic information about optimization and scanning.

To generate additional member-level log files, use the LOG option.

The log file for optimization

When optimization is performed on a module (SCAN=N optimizer option is in effect), the log file includes the following information:

- File name information of the input module being processed
- Names of the CSECTs being optimized
- File name information of the listing transform for each optimized CSECT
- File name information of the output optimized module
- A time stamp for each line of the OPTLOG and a header for the date
- Other diagnostic information including error messages

Example 1:

The following log file shows COBOL CSECTs from members MEMA and MEMB of data set *HLQ*.IN1.LOAD are being optimized. Module MEMA has two COBOL CSECTs named SUB1 and SUB2 that are optimized. Module MEMB has one COBOL CSECT named PROGB that is optimized. Listing transforms are all placed in the default SYSPRINT DD. Optimized modules are written to the *HLQ*.OUT1.LOAD data set.

```
5697-AB2 IBM Automatic Binary Optimizer for z/OS 2.2.0

======== Apr 22 2022 ========
10:53:16 Optimizer build level: tr_v22_binopt_20220422_1409_VWTre8JnEeyIPYltUEoX1g
(Apr 22 2022 20:05:19)
10:53:16 Processing HLQ.IN1.LOAD, member MEMA
10:53:16   Processing CSECT SUB1, in member MEMA
10:53:16        Optimizing CSECT SUB1 for z16
10:53:16        Succeeded in optimizing SUB1
10:53:16        Generating listing transform into DD:SYSPRINT
10:53:16   Processing CSECT SUB2, in member MEMA
10:53:16        Optimizing CSECT SUB2 for z16
10:53:16        Succeeded in optimizing SUB2
10:53:16        Generating listing transform into DD:SYSPRINT
10:53:16   Finished processing, processed 2 of 2 CSECTs in member MEMA
10:53:16   Save HLQ.OUT1.LOAD (MEMA) succeeded
10:53:16 Processing HLQ.IN1.LOAD, member MEMB
10:53:16   Processing CSECT PROGB, in member MEMB
10:53:16        Optimizing CSECT PROGB for z16
10:53:16        Succeeded in optimizing PROGB
10:53:16        Generating listing transform into DD:SYSPRINT
10:53:16   Finished processing, processed 1 of 1 CSECTs in member MEMB
10:53:16   Save HLQ.OUT1.LOAD (MEMB) succeeded
10:53:16 Exiting with return code: 0
```

Example 2:

The following log files show the output for the member-level log files produced by the same *HLQ*.IN1.LOAD as the previous example with LOG=*HLQ*.LOG.OUT specified.

The contents of *HLQ*.LOG.OUT(MEMA):

```
5697-AB2 IBM Automatic Binary Optimizer for z/OS 2.2.0
======== Apr 22 2022 ========
10:53:16 Optimizer build level: tr_v22_binopt_20220422_1409_VWTre8JnEeyIPYltUEoX1g (Apr 22 2022
20:05:19)
10:53:16 Processing HLQ.IN1.LOAD, member MEMA
10:53:16   Processing CSECT SUB1, in member MEMA
10:53:16        Optimizing CSECT SUB1 for z16
10:53:16        Succeeded in optimizing SUB1
10:53:16        Generating listing transform into DD:SYSPRINT
10:53:16   Processing CSECT SUB2, in member MEMA
10:53:16        Optimizing CSECT SUB2 for z16
10:53:16        Succeeded in optimizing SUB2
10:53:16        Generating listing transform into DD:SYSPRINT
10:53:16   Finished processing, processed 2 of 2 CSECTs in member MEMA
10:53:16   Save HLQ.OUT1.LOAD (MEMA) succeeded
10:53:16 Exiting with return code: 0
```

The contents of *HLQ*.LOG.OUT(MEMB):

```
5697-AB2 IBM Automatic Binary Optimizer for z/OS 2.2.0
======== Apr 22 2022 ========
10:53:16 Optimizer build level: tr_v22_binopt_20220422_1409_VWTre8JnEeyIPYltUEoX1g (Apr 22 2022
20:05:19)
10:53:16 Processing HLQ.IN1.LOAD, member MEMB
10:53:16   Processing CSECT PROGB, in member MEMB
10:53:16        Optimizing CSECT PROGB for z16
10:53:16        Succeeded in optimizing PROGB
10:53:16        Generating listing transform into DD:SYSPRINT
10:53:16   Finished processing, processed 1 of 1 CSECTs in member MEMB
10:53:16   Save HLQ.OUT1.LOAD (MEMB) succeeded
10:53:16 Exiting with return code: 0
```

The log file for scanning

When scanning is performed (SCAN=Y optimizer option is in effect), the log file shows:

- File name information of the input module being scanned
- Names of the CSECT of the module being scanned
- Other diagnostic messages

Example:

The following log file shows modules MEMA and MEMB of data set *HLQ*.MEM1.LOAD are being scanned. Scanning output shows each of the CSECTs in the modules. The COBOL compiler version used for the compilation and the "Signature information bytes" extracted from the CSECT are displayed for the COBOL CSECTs SUB, SUB2 and PROGB. "Signature information bytes" are documented in the *COBOL Programming Guide* and provide information about the compiled program.

```
5697-AB2 IBM Automatic Binary Optimizer for z/OS 2.2.0

======== Apr 22 2022 ========
10:58:23 Optimizer build level: tr_v22_binopt_20190730_1658_t54Rw7MMEem46oTHworqTQ
(Jul 30 2019 20:05:19)
10:58:23 Processing HLQ.IN1.LOAD, member MEMA
      Language ID Records:
            id 5688187 v21 m00 2015281 resident EDCOEXTS
            id 5655S7100 v42 m00 2015281 resident SUB
           Enterprise COBOL V4: start=0x10, length=4.19 (kBytes)
                        Signature information bytes:
                  a0487d4c 20000000 00880100 00000040
                  08000000 000000 00008004 1400
            id 5655S7100 v42 m00 2015281 resident SUB2
           Enterprise COBOL V4: start=0x10d8, length=4.20 (kBytes)
               Signature information bytes:
```

```
                    a0487d4c 20000000 00880100 00000040
                    08000000 000000 00008004 1400
             id 569623400 v01 m06 2013071 resident CEESG005
             id 569623400 v01 m06 2013072 resident CEEBETBL
             id 569623400 v01 m06 2013072 resident CEESTART
10:58:23 Processing HLQ.IN1.LOAD, member MEMB
      Language ID Records:
             id 5688187 v21 m00 2015281 resident EDCOEXTS
             id 5655S7100 v42 m00 2015281 resident PROGB
           Enterprise COBOL V4: start=0x10, length=8.19 (kBytes)
                        Signature information bytes:
                        a0487d4c 20000000 00880100 00000040
                          08000000 000000 00008004 1400
             id 569623400 v01 m06 2013072 resident CEESTART
10:58:23 Exiting with return code: 0
```

Listing transform

The listing transform describes the changes made by the Automatic Binary Optimizer for z/OS to your already compiled program module. It shows how the instructions in the input binary map to the optimized instructions. A listing transform is produced for every CSECT that is optimized. The listing transform is intended to complement the compiler listing generated when the input binary was originally compiled from source. While the listing transform does not depend on the compiler listing, using the two together will allow you to better understand how ABO transformed your binary.

A listing transform is generated unconditionally for every optimized CSECT; no special options or flags need to be specified. By default, listing transforms are generated into the SYSPRINT DD.

The listing transform is also used by “Application Delivery Foundation for z/OS” on page 89.

Listing transform contents

The listing transform is provided to help in diagnosing problems encountered during the execution of the optimized program. The listing transform is primarily intended for use by debugging tools such as IBM Debug for z/OS.

A listing transform contains the following information:

- A summary of the optimization options
- The optimized instructions interspersed with the input instructions
- A literal pool containing any new literals created by ABO
- A PPA4 section containing information about the optimized CSECT
- An automatic map, also known as a dynamic storage area (DSA) map, of any new stack symbols that were created by ABO
- An input instructions section containing the complete list of instructions and compiler options for the CSECT being optimized

Summary of optimization parameters

This section contains the name of the architecture level for which the program is optimized, and the date and time stamp of both the input binary, along with the compiler used to produce it. The date and time stamp of the output binary is also shown.

Example:

```
Invocation Parameters:
  Architecture Level: z16

Input IDRL Record: 5655S7100 v42 m00 2013122
  Name: Enterprise COBOL V4
  Version: 42
```

```
  Mod Level: 00
  Compiled Date (YYYYDDD): 2013122

Output IDRL Record: 5697-AB2 v22 m00 2022158
  Name: IBM Automatic Binary Optimizer for z/OS
  Version: 22
  Mod Level: 00
  Optimized Date (YYYYDDD): 2019256
```

Optimized instructions

This section makes up the majority of the data in the listings transform. It is similar to the assembler code section of the listings generated by various IBM COBOL compilers, such as IBM Enterprise COBOL V5 and V6. Each CSECT that was optimized begins with the PR0C psuedo opcode and the name of the CSECT as its operand.

Example:

```
     (1)       (2)              (3)              (4)      (5)

    000258                     000000           PROC     PROGA
    000258  183F               000000           LR       R3,R15
    00025A  5800 3008          000000           L        R0, 8(,R3)
    00025E  1E01               000000           ALR      R0,R1
    000260  5500 C00C          000000           CL       R0, 12(,R12)
    000264  0DF0               000000           BASR     R15,R0
    000266  47D0 F00C          000000           BC       R13, 12(,R15)
    00026A  58F0 C300          000000           L        R15, 768(,R12)
    00026E  0DEF               000000           BASR     R14,R15
    000270  181F               000000           LR       R1,R15
    000272  50D0 1004          000000           ST       R13, 4(,R1)
    000276  5000 104C          000000           ST       R0, 76(,R1)
    00027A  D203 1000 3058     000000           MVC      0(4,R1), 88(R3)
    000280  D703 1084 1084     000000           XC       132(4,R1), 132(R1)
    000286  5090 105C          000000           ST       R9, 92(,R1)
    00028A  18D1               000000           LR       R13,R1
    00028C  41A0 D120          000000           LA       R10, 288(,R13)
```

The preceding example shows the optimized instructions produced for a CSECT named PR0GA. The five sections of an optimized instruction are described as follows:

1. The hexadecimal offset in the CSECT of the optimized instruction
2. The hexadecimal representation of the instruction bytes
3. The hexadecimal CSECT offset of the "source" instructions for which these optimized instructions were generated
4. Instruction opcode
5. Instruction operands

Interspersed with the optimized instructions are the "source" instructions for which the optimized instructions are generated. Lines that begin in column 5 are the optimized instructions, and the lines that begin in column 1 are the "source" instructions. In the following example, the first two lines, starting at column 1, are the PACK and OI source instructions at hex offsets 00042C and 000432 respectively. The third line, starting in column 5, is the ABO generated instruction 'CDZT'. Notice that the "source" hex offset of the CDZT is 00042c, which shows that it was generated for the PACK instruction in the input module.

Example:

```
00042C   PACK    272(4,13),0(7,8)
000432   OI      276(13),15
    0004C4  ED07 4000 00AA     00042C           CDZT     FP0,_WSA[0x12c] 0(8,R4),0x0
000436   PACK    280(4,13),8(7,8)
00043C   OI      284(13),15
    0004CA  ED07 4008 10AA     000436           CDZT     FP1,_WSA[0x12c] 8(8,R4),0x0
000440   AP      272(4,13),280(4,13)
000446   UNPK    16(7,8),272(4,13)
    0004D0  B3D2 1000          000440           ADTR     FP0,FP0,FP1
00044C   OI      23(8),240
```

```
000450   L       2,248(0,,13)
    0004D4  ED07 4010 00A8     00044C            CZDT    FP0, 16(8,R4),0x0
```

The literal pool

ABO places any new literals it creates at the end of the code section. These sections are named *Constant Data Snippets* in the listing transform. There may be zero or more Constant Data Snippets in the listing and their contents are very similar to the literal pool that is created by the original compiler. The original literal pool remains intact and continues to be used, just as in the input binary.

Example:

```
    (1)         (2)                   (3)              (4)

                                      L0032:            #  Constant Data Snippet
    000550  4040 4040 4040 4040                 DC      X'4040404040404040'
    000558  4040 4040 4040 4040                 DC      X'4040404040404040'
    000560  4040 4040 4040 4040                 DC      X'4040404040404040'
    000568  4040 4040 4040 0000                 DC      X'4040404040400000'
    000570  8000 0000 0000 0000                 DC      X'8000000000000000'
    000578  0000 0000 0000 0000                 DC      X'0000000000000000'
    000580  0000 0001 0000 0000                 DC      X'0000000100000000'
    000588  C9C7 E9E2 D9E3 C3C4                 DC      X'C9C7E9E2D9E3C3C4'
    000590  E2E8 E2D6 E4E3 4040                 DC      X'E2E8E2D6E4E34040'
    000598  0E00 0000 0000 0000                 DC      X'0E00000000000000'
```

A description of the four sections of data is as follows:

1. The hexadecimal offset from the start of the CSECT to these bytes in the literal pool
2. The hexadecimal representation of the bytes in the literal pool
3. The label denoting the start of the literal pool
4. Assembler syntax of these bytes

Additional DSA and TGT Bytes Allocated section

This section displays in hexadecimal number of bytes any additional DSA or TGT bytes allocated by the optimizer

Example:

```
DSA WILL BE ALLOCATED FOR AN ADDITIONAL 000001A8 BYTES
TGT WILL BE ALLOCATED FOR AN ADDITIONAL 00000000 BYTES
```

The PPA4 section

The PPA4 section contains information about the optimization of the program module. For example, it contains the time and date of optimization, the length of the code section and other information.

Example:

```
    (1)      (2)                                    (3)             (4)


                    PPA4:  Entry Point Constants
    046668  00000000                               =X'00000000'  Flags 1
    04666C  00000300                               =X'00000300'  Flags 2
    046670  F2F0F1F6                               =C'2019'      Compiled Year
    046674  F0F8F2F1                               =C'0913'      Compiled Date MMDD
    046678  F1F2F3F0F2F3                           =C'123023'    Compiled Time HHMMSS
    04667E  F0F1F0F2F0F0                           =C'010200'    Compiler Version
    046684  0004706A                               =F'290922'    Code Length
    046688  0B020000                               =X'0B020000'  Options
    04668C  00000028                               =X'00000028'  A(PPA4-ListName)
                    PPA4  End
```

A description of the four sections of PPA4 follows:

1. The offset in hexadecimal in the CSECT of PPA4 section entry
2. The hexadecimal representation of the bytes in the PPA4 section
3. Assembler syntax of the bytes in the PPA4
4. Description of the data in the PPA4

The automatic map

The automatic map contains the offsets and sizes (in hexadecimal) of symbols that are created by ABO. These offsets are relative to a base established at the end of the original DSA. The automatic map does not show automatics in the original program or temporaries created by the original compiler. ABO will establish a general purpose register (GPR) to contain the start offset of the "new" DSA. All newly created automatics will be referenced with this new register as the base.

Example:

```
    (1)              (2)        (3)

                   * * * * *   A U T O M A T I C     M A P   * * * * *

 OFFSET (HEX)   LENGTH (HEX)   NAME

           0              4       _GPR0
           4              4       _GPR1
           8              4       _GPR2
           C              4       _GPR3
          10              4       _GPR4
          14              4       _GPR5
          18              4       _GPR6
          1C              4       _GPR7
          20              4       _GPR8
          24              4       _GPR9
          28              4       _GPR10
          2C              4       _GPR11
          30              4       _GPR12
          34              4       _GPR13
```

A description of the three sections of the automatic map follows:

1. Hexadecimal offset of the stack symbol, relative to the start of the new stack
2. Hexadecimal length of the symbol in bytes
3. Name of the symbol

Input instructions

This section contains the complete list of instructions from the input module. There is one section for each CSECT that was optimized. These input instructions are the same as those already shown in the Optimized instructions section. In that section, the input instructions are shown interspersed with the corresponding optimized instructions, and as such are not a complete and ordered list. Each CSECT has its own prolog section and is displayed after "Compiler Options in Effect" finishes. This prolog section is displayed for CSECTs compiled by compiler releases from COBOL/370 1.1 to Enterprise COBOL 6. CSECTs compiled by VS COBOL II 1.3 and 1.4 do not display the prolog section.

The input instructions section begins with the COBOL compiler version used for the compilation and the "Signature information bytes" extracted from the CSECT. "Signature information bytes" are documented in the *COBOL Programming Guide* and provide information about the compiled program. These information bytes are decoded and the corresponding compiler options that were in effect are printed. Note that the decoded compiler options may not exactly match in content and formatting those displayed in the original compiler listing. This is because ABO decodes the options only according to the signature information bytes present in the input CSECT instead of the full original source and options specified during the original compilation process.

In the following example, the COBOL CSECT named PROGA was optimized.

```
 (1)      (2)       (3)

                * * * * *  I N P U T  I N S T R U C T I O N S  * * * * *

id 5655S7100  v42 m00 2020072 resident PROGA
    Enterprise COBOL V4: start=0x0, length=1.39 (kBytes)
        Signature information bytes:
          a4487d4c 20000000 00080000 00000000
          08000000 00000000 00808014 00
        Compiler Options in effect:
            ADV
            ARITH(COMPAT)
          NOAWO
          NOBLOCK0
            Compilation unit is a program.
          NOCICS
            CODEPAGE(1140)
          NOCURRENCY
            Default DDNAME for OUTDD will be used
            DATA(31)
          NODATEPROC
            DBCS
          NODECK
          NODLL
          NODUMP
            DYNAM
          NOEXPORTALL
          NOFASTSRT
            INTDATE(ANSI)
          NOLIB
            LIST
          NOMAP
          NOMDECK
          NONAME
          NONUM
            NUMCLS(PRIM)
            NUMPROC(NOPFD)
            OBJ
          NOOFFSET
          NOOPTIMIZE
            PGMNAME(LONGUPPER)
            QUOTE
            RENT
            RMODE(ANY)
            SEQUENCE
            SIZE(MAX)
            SOURCE
          NOSQL
            SQLCCSID
          NOSSRANGE
            TERM
          NOTEST
          NOTHREAD
            TRUNC(STD)
          NOVBREF
          NOWORD
            XMLPARSE(XMLSS)
            XREF
            YEARWINDOW(1900)
            ZWB


                * * * * *  P R O L O G U E    I N F O  * * * * *

000000     47F0F028            BYPASS CONSTANTS. BRANCH TO @STM
000004     00                  ZERO NAME LENGTH FOR DUMPS
000005     C3C5C5              CEE EYE CATCHER
000008     00000110            STACK FRAME SIZE
00000C     00000014            OFFSET TO PPA1 FROM PRIMARY ENTRY
000010     47F0F001            RESERVED

CSECT: PROGA     PPA1
000014     98                  OFFSET TO LENGTH OF PROGRAM NAME FROM PPA1
000015     CE                  CEL SIGNATURE
000016     AC                  CEL FLAGS
000017     00                  MEMBER FLAGS FOR COBOL
000018     000000B6            ADDRESS OF PPA2
00001C     00000000            OFFSET TO THE BDI
000020     00000000            ADDRESS OF ENTRY POINT DESCRIPTORS
```

```
   000024     0000                RESERVED
   000026     00                  DSA FPR 8-15 SAVE AREA OFFSET/16
   000027     00                  DSA FPR 8-15 SAVE AREA BIT MASK
   000028     90ECD00C            STM STARTS HERE: SAVE CALLER'S REGISTERS
   00002C     4110F038            GET ADDRESS OF PARMLIST INTO R1
   000030     98EFF04C            LOAD ADDRESSES FROM @BRVAL
   000034     07FF                DO ANY NECESSARY INITIALIZATION
   000036     0000                AVAILABLE HALF-WORD
   000038     00000000            1) PRIMARY ENTRY POINT ADDRESS
   00003C     00000000            2) AVAILABLE
   000040     000003A0            3) DAB ADDRESS
   000044     000000AE            4) ENTRY POINT NAME ADDRESS
   000048     00000000            5) CURRENT ENTRY POINT ADDRESS
   00004C     00000262            6) PROCEDURE CODE ADDRESS
   000050     000018C0            7) INITIALIZATION ROUTINE
   000054     000000CA            8) ADDRESS OF PARM LIST FOR CEEINT
   000058     00104001            DSA WORD 0 CONSTANT
   00005C     00000008            AVAILABLE WORD
   000060     D7D9D6C7            AVAILABLE WORD
   000064     C1404040            AVAILABLE WORD
   000068     F2F0F2F0            YEAR OF COMPILATION
   00006C     F0F3F1F2            MONTH/DAY OF COMPILATION
   000070     F1F3F3F8            HOURS/MINUTES OF COMPILATION
   000074     F2F1                SECONDS FOR COMPILATION DATE
   000076     F0F4F0F2F0F0        VERSION/RELEASE/MOD LEVEL OF PROD
   00007C     0474                UNSIGNED BINARY CODE PAGE CCSID VALUE
   00007E     0000                AVAILABLE HALF-WORD
   000080     1400                INFO. BYTES 28-29
   000082     076C                SIGNED BINARY YEARWINDOW OPTION VALUE
   000084     A4487D4C2000        INFO. BYTES 1-6
   00008A     000000080000        INFO. BYTES 7-12
   000090     000000000800        INFO. BYTES 13-18
   000096     0000000000          INFO. BYTES 19-23
   00009B     00                  COBOL SIGNATURE LEVEL
   00009C     00000003            # DATA DIVISION STATEMENTS
   0000A0     00000002            # PROCEDURE DIVISION STATEMENTS
   0000A4     000080              INFO. BYTES 24-26
   0000A7     80                  INFO. BYTE 27
   0000A8     40404040            USER LEVEL INFO (LVLINFO)
   0000AC     0005                LENGTH OF PROGRAM NAME
   0000AE     D7D9D6C7C1404040    PROGRAM NAME


   CSECT: PROGA     PPA2
   0000B6     05                  CEL MEMBER IDENTIFIER
   0000B7     00                  CEL MEMBER SUB-IDENTIFIER
   0000B8     00                  CEL MEMBER DEFINED BYTE
   0000B9     01                  CONTROL LEVEL OF PROLOG
   0000BA     00001810            VCON FOR LOAD MODULE
   0000BE     00000000            OFFSET TO THE CDI
   0000C2     FFFFFFB2            OFFSET TO TIMESTAMP/VERSION INFO
   0000C6     00000000            ADDRESS OF CU PRIMARY ENTRY POINT
   0000CA     00000038            PARM LIST FOR CEEINT: POINTER TO PRIMARY ENTRY PT ADDR
   0000CE     00000008            OFFSET TO PARAMETERS FOR CEEINT
   0000D2     00000006            PARAMETERS FOR CEEINT 1) NUMBER OF ENTRIES IN PARM LIST
   0000D6     00000038            2) POINTER TO PRIMARY ENTRY PT ADDR
   0000DA     00001810            3) ADDRESS OF CEESTART
   0000DE     000017E8            4) ADDRESS OF CEEBETBL
   0000E2     00000005            5) CEL MEMBER IDENTIFIER
   0000E6     00000000            6) FOR CEL MEMBER USE
   0000EA     00000000            AVAILABLE WORD
   0000EE     00000000            AVAILABLE WORD
   0000F2     00000000            AVAILABLE WORD
   0000F6     00000000            AVAILABLE WORD
   0000FA     0000                AVAILABLE HALF-WORD


   CSECT: PROGA    PGT SECTION

   0000FC   (LIT+0)    00000001 40404040 40404040 40404040 40404040 40404040 40404040
40404040     |....                                |
   00011C   (LIT+32)    40400000 2A05D7C0 2A05D890 2A05D8E4
2A05D8FA                                  |  ....P{..Q...QU..Q.|

   CSECT: PROGA    CGT SECTION

   000130   (LIT+0)    FFFFFFFC 00001000 00000001 00000000 899540D7 D9D6C7C1 E2E8E2D6
E4E34040     |.............in PROGASYSOUT  |
   000150   (LIT+32)    C9C7E9E2 D9E3C3C4 00000000 0000012C 00000001 00000130 00000001
00000000     |IGZSRTCD.....................|
   000170   (LIT+64)    00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000     |............................|
```

```
   000190   (LIT+96)    80000000 40000000 00000000 00000000 F0F0F0F9 F7F9F1F6 F4F0F3F6
F0400000     |.... ...........0009791640360 ..|
   0001B0   (LIT+128)    00000000 00000000 00000000 00400000 000025C0 0001C000 05080000
24001B01     |............. .....{..{.........|
   0001D0   (LIT+160)    40000008 08000024 0018FF40 00000000 40C00001 40000508 00002400
1B02C000     | .......... .... {.. .........{.|
   0001F0   (LIT+244)    05080000
24001B00                                                                |........|


                  * * * * *  E N D  O F  P R O L O G U E    I N F O  * * * * *

    0001F8  5820 9128          L       2,296(0,,9)
    0001FC  D201 2008 A00C     MVC     8(2,2),12(10)
    000202  D201 2010 A00C     MVC     16(2,2),12(10)
    000208  D207 2018 A028     MVC     24(8,2),40(10)
    00020E  D203 2020 A014     MVC     32(4,2),20(10)
    000214  D203 2028 A014     MVC     40(4,2),20(10)
    00021A  D203 2030 A014     MVC     48(4,2),20(10)
    000220  D207 2038 A020     MVC     56(8,2),32(10)
    000226  D203 2040 A014     MVC     64(4,2),20(10)
    00022C  920E 2048          MVI     72(2),14
    000230  920F 2050          MVI     80(2),15
    000234  D203 2058 A014     MVC     88(4,2),20(10)
    00023A  D203 2080 A014     MVC     128(4,2),20(10)
    000240  D21D 2060 C004     MVC     96(30,2),4(12)
    000246  5830 D0F0          L       3,240(0,,13)
    00024A  07F3               BCR     -1,3
    00024C  5820 912C          L       2,300(0,,9)
    000250  D208 2010 A078     MVC     16(9,2),120(10)
    000256  D203 2020 A081     MVC     32(4,2),129(10)
    00025C  5830 D0F4          L       3,244(0,,13)
    000260  07F3               BCR     -1,3
    000262  183F               LR      3,15
    000264  4100 1110          LA      0,272(0,,1)
    000268  5500 C00C          CL      0,12(0,,12)
    00026C  0DF0               BASR    15,0
    00026E  47D0 F00C          BC      13,12(0,15)
    000272  58F0 C300          L       15,768(0,,12)
    000276  0DEF               BASR    14,15
    000278  181F               LR      1,15
    00027A  50D0 1004          ST      13,4(0,,1)
    00027E  5000 104C          ST      0,76(0,,1)
    000282  D203 1000 3058     MVC     0(4,1),88(3)
    000288  D703 1084 1084     XC      132(4,1),132(1)
    00028E  5090 105C          ST      9,92(0,,1)
    000292  18D1               LR      13,1
    000294  58C0 90E8          L       12,232(0,,9)
    000298  1812               LR      1,2
    00029A  50D0 D058          ST      13,88(0,,13)
    00029E  58A0 C024          L       10,36(0,,12)
    0002A2  D203 D088 A010     MVC     136(4,13),16(10)
    0002A8  BF2F 9144          ICM     2,15,324(9)
    0002AC  58B0 C028          L       11,40(0,,12)
    0002B0  4780 B0CA          BC      8,202(0,11)
    0002B4  5830 905C          L       3,92(0,,9)
    0002B8  58F0 30F4          L       15,244(0,,3)
    0002BC  4110 A0B3          LA      1,179(0,,10)
    0002C0  0DEF               BASR    14,15
    0002C2  5A20 C000          A       2,0(0,,12)
    0002C6  5020 9144          ST      2,324(0,,9)
    0002CA  9140 9134          TM      308(9),64
    0002CE  4710 B0F2          BC      1,242(0,11)
    0002D2  D203 D0F8 D0F4     MVC     248(4,13),244(13)
    0002D8  4120 B0EC          LA      2,236(0,,11)
    0002DC  5020 D0F4          ST      2,244(0,,13)
    0002E0  47F0 B054          BC      15,84(0,11)
    0002E4  D203 D0F4 D0F8     MVC     244(4,13),248(13)
    0002EA  9140 9134          TM      308(9),64
    0002EE  58B0 C028          L       11,40(0,,12)
    0002F2  4710 B116          BC      1,278(0,11)
    0002F6  D203 D0FC D0F0     MVC     252(4,13),240(13)
    0002FC  4120 B110          LA      2,272(0,,11)
    000300  5020 D0F0          ST      2,240(0,,13)
    000304  47F0 B000          BC      15,0(0,11)
    000308  D203 D0F0 D0FC     MVC     240(4,13),252(13)
    00030E  9140 9057          TM      87(9),64
    000312  58B0 C028          L       11,40(0,,12)
    000316  4710 B136          BC      1,310(0,11)
    00031A  9120 9054          TM      84(9),32
    00031E  47E0 B12E          BC      14,302(0,11)
    000322  9620 D084          OI      132(13),32
```

```
    000326  9640 9057          OI      87(9),64
    00032A  47F0 B136          BC      15,310(0,11)
    00032E  9640 9134          OI      308(9),64
    000332  9601 D084          OI      132(13),1
    000336  5820 905C          L       2,92(0,,9)
    00033A  58F0 202C          L       15,44(0,,2)
    00033E  4110 A0A7          LA      1,167(0,,10)
    000342  0DEF               BASR    14,15
    000344  47F0 B162          BC      15,354(0,11)
    000348  9120 D084          TM      132(13),32
    00034C  47E0 B162          BC      14,354(0,11)
    000350  58F0 20F4          L       15,244(0,,2)
    000354  4110 A095          LA      1,149(0,,10)
    000358  0DEF               BASR    14,15
    00035A  5830 9144          L       3,324(0,,9)
    00035E  5B30 C000          S       3,0(0,,12)
    000362  5030 9144          ST      3,324(0,,9)
    000366  9128 9054          TM      84(9),40
    00036A  4770 B18C          BC      7,396(0,11)
    00036E  5830 9128          L       3,296(0,,9)
    000372  48F0 3008          LH      15,8(0,,3)
    000376  58D0 D004          L       13,4(0,,13)
    00037A  58E0 D00C          L       14,12(0,,13)
    00037E  980C D014          LM      0,12,20(13)
    000382  07FE               BCR     -1,14
    000384  D20B D100 A06C     MVC     256(12,13),108(10)
    00038A  5840 9128          L       4,296(0,,9)
    00038E  4830 4008          LH      3,8(0,,4)
    000392  5030 D10C          ST      3,268(0,,13)


                    * * * * *  E N D  O F  I N P U T  I N S T R U C T I O N S  * * * * *
```

A description of the three sections of the input instructions section follows:

1. The hexadecimal offset in the input CSECT of the original instruction
2. Instruction mnemonic
3. Instruction operands

How to detect ABO optimized modules and ABO specific PPA4

This section introduces a new way to detect if a program was optimized by ABO, and a new way to reach the ABO specific PPA4 block.

Detect the ABO optimized modules

For Enterprise COBOL 5.1 and later compiled modules that were optimized by ABO, bit12 is set in 'Compilation flags' in the PPA2 (PPA2 Control Level = 3, PPA2+0x14):

```
'........ ....0... ........ ........'B Indicates that program is not optimized by ABO
'........ ....1... ........ ........'B Indicates that program is optimized by ABO
```

To determine if a given compiled program (a CSECT) was optimized by ABO, follow these steps:

- If the PPA4 version (bits 16-23 in PPA4 offset X’04’) is 3, then this is an Enterprise COBOL 4.2 or earlier compiled program that was optimized by ABO.
- If the ‘Member Identifier’ (bits 0-7 in PPA2 offset X’00) is 4 and the bit12 in ‘Compilation flags’(PPA2 offset X’14’) is 1, then this is an Enterprise COBOL 5.1 or later compiled program that was optimized by ABO.
- Otherwise, the program was not optimized by ABO.

Find the ABO specific PPA4 block

ABO preserves the existing layout of PPA1, PPA2, and PPA3 blocks in the Enterprise COBOL 5.1 and later compiled programs. ABO also preserves almost the same PPA4 layout as Enterprise COBOL ('COBOL V5+

32-bit PPA4 layout (with minor-version)') except that the minor version is increased from 0 to 1. See also COBOL V5+ 32-bit PPA4 layout (with minor-version).

This new PPA4 minor version 1 of the Enterprise COBOL 5.1 and later PPA4 introduces a new 2-byte field at PPA4 offset X'2C' :A(PPA4ABO-PPA4). This field contains the signed offset from the start of the Enterprise COBOL 5.1 and later PPA4 to a second PPA4 block, with a layout specific to ABO optimized modules.

This new field is used to locate the ABO specific PPA4 block when the compiled COBOL program optimized by ABO was produced by Enterprise COBOL 5.1 or later. For ABO optimized programs produced from Enterprise COBOL 4.2 and earlier compiled programs, the algorithm is unchanged from ABO 1.1 to 2.1 and the entire PPA4 containing only the ABO specific information is found in the standard way from the PPA4 offset in PPA2.

You can find the ABO specific PPA4 block regardless of the compiled language type or Enterprise COBOL compiler version. Note that the conditional checks are the same as in the earlier algorithm to determine if the module was optimized by ABO or not.

- If the PPA4 version (bits 16-23 in PPA4 offset X'04') is 3, then this entire PPA4 is the ABO specific version.
- If the Member Identifier (bits 0-7 in PPA2 offset X'00) is 4 and the bit12 in 'Compilation flags' (PPA2 offset X'14') is 1, then PPA4+X'2C' field 'A(PPA4ABO-PPA4)' has the signed offset to the ABO specific PPA4.
- Otherwise, the program was not optimized by ABO so there is no ABO specific PPA4.

Enterprise COBOL 5.1 and later compiled programs not optimized by ABO will have the PPA4+X'2C' field A(PPA4ABO-PPA4) set to 0.

SYSPRINT DD and LIST option

Use the SYSPRINT DD or LIST options to specify the locations of the generated listing transforms.

The target of SYSPRINT or LIST can be one of the following items:

- A sequential data set or member of a PDSE (not PDS). The output of multiple CSECT optimizations are added to this sequential data set in optimization order.
- A PDS or PDSE. When a CSECT is optimized, the listing transform particular to that CSECT is placed in a member of the PDS or PDSE where the member name is based on the CSECT name (upper cased and truncated to 8 characters). The contents of the member, if any, are overwritten even if the former contents are produced by ABO in previous invocations.
- An HFS path. The output of multiple CSECT optimizations are added to this HFS file.

The LIST option takes precedence over the SYSPRINT DD. If you specify the LIST option, it will override the SYSPRINT DD. When the LIST option is specified, you can omit the SYSPRINT DDname.

Note:

The ABO listing contains detailed transformation information and can therefore become very large. Specifying SYSPRINT DD target as SYSOUT might cause the JES2 spools to reach a system specified line limit. When the spool line limit is reached, the JES2 passes control to an installation exit routine but the ABO job may or may not be terminated. Although the spool line limit can be increased using the JOBPARM L option, the maximum L setting of 999999 might still not be large enough for the ABO listing.

To avoid this problem, it is recommended that SYSPRINT specify a PDS, PDSE or HFS location as documented in this section.

Example

The following JCL example uses a PDSE in the SYSPRINT DD so that listing transforms are written to the members of the PDSE.

```
//SYSIN DD *
  BOPT IN=HLQ.IN.LOAD(MOD*) OUT=HLQ.OUT.LOAD
  …
//SYSPRINT DD DSN=HLQ.LIST.PDSE,DISP=SHR
```

In this example, the input program modules are specified as *HLQ*.IN.LOAD(MOD*), which means, optimize all eligible members in *HLQ*.IN.LOAD with names beginning with "MOD".

There are two members in the input data set, MOD1 and MOD2. Within these two program modules, are various CSECTs:

Table 10. Input modules and their containing CSECTs

***HLQ*.IN.LOAD**	**CSECTs**
MOD1	PROG1A
	PROG1B
	PROG1C
MOD2	PROG2A
	PROG2B

ABO will optimize each of these CSECTs, one at a time, and produce two outputs for each CSECT:

1. The optimized CSECT
2. The listing transform for the CSECT

The optimized CSECT has the same name as the input CSECT, and the optimized CSECT will be placed in a program module that has the same member name as the input program module. However, the new program modules will be placed into a new PDSE called '*HLQ*.OUT.LOAD'

Table 11. Output 1: Optimized modules and their CSECTs

***HLQ*.OUT.LOAD**	**CSECTs**
MOD1	PROG1A
	PROG1B
	PROG1C
MOD2	PROG2A
	PROG2B

The listings, generated for each of the optimized CSECT, are placed into the PDSE '*HLQ*.LIST.PDSE', as separate members. Each such PDSE member will have the same name as the input CSECT name. The results is that *HLQ*.LIST.PDSE will have 5 members, PROG1A, PROG1B, PROG1C, PROG2A and PROG2B.

Table 12. Output 2: Listing transforms

***HLQ*.LIST.PDSE**
PROG1A
PROG1B
PROG1C
PROG2A

Table 12. Output 2: Listing transforms (continued)
***HLQ*.LIST.PDSE**
PROG2B

Chapter 7. Using the ABO Assistant

Overview of the ABO Assistant

The IBM Automatic Binary Optimizer (ABO) Assistant is a suite of tools to automate the main parts of finding and optimizing your top CPU consuming COBOL applications.

Batch applications

Starting from your provided SMF data, the Batch SMF Analyzer component of the ABO Assistant will produce a prioritized report on the jobs and programs that consume the most CPU across all the jobs running on your system. Optionally, the JCL locations to run these programs will also be displayed.

For programs known to contain COBOL, these JCL locations can then be used as input to the second component of the ABO Assistant called the Program Analyzer and Optimizer. This component will automate all the individual steps required to efficiently optimize the top CPU consuming individual COBOL batch applications using ABO and to clearly report the CPU time savings from using ABO.

After inputting the JCL used to run each COBOL batch application, one at a time, to the Program Analyzer and Optimizer, it automatically does the following:

- Rebinds the original program to enable an RTI Profile to be collected
- Runs the rebound original program to report the CPU time taken by the program and to collect the RTI profile
- Optimizes the top CPU consuming COBOL CSECTs as found in the RTI profile with ABO
- Runs the ABO optimized program
- Reports the CPU time of the optimized program, proportion of time spent in COBOL, and the percentage of the CPU reduction from using ABO

The JCL for the next top CPU consuming COBOL batch application can then be passed to the Program Analyzer and Optimizer component of the ABO Assistant to repeat the above process.

Note: Each target COBOL application must be re-runnable sequentially without cleanup or other steps needed that are not already present in the original JCL. As the application will be run twice, it is highly recommended that the Program Analyzer and Optimizer component of the ABO Assistant only be used on a test or development system. For more information on this and other restrictions, see “Limitations and requirements on Program Analyzer and Optimizer” on page 73.

CICS applications

Starting from your provided CICS SMF data, gathered according to the instructions at “Gathering SMF 110 data” on page 76, the CICS SMF Analyzer component of the ABO Assistant will produce a *prioritized report* on the programs and transactions that consume the most CPU across all the programs running in your CICS region.

For the top CPU consuming programs known to contain COBOL, ABO can then be used to optimize these compiled programs to reduce the CPU time, reduce transaction time and improve response time.

To demonstrate these ABO improvements a second report, using SMF data gathered when using the ABO optimized modules, can be produced from the CICS SMF Analyzer.

Finally, the two reports from original and ABO optimized runs can be compared to produce a *comparison report* that shows the total CPU reduction from using ABO as well as the change in CPU time from each individual program.

Throughout this document:

- Prioritized report denotes the report the CICS SMF Analyzer produces from the SMF data

- Comparison report denotes the report the CICS SMF Analyzer produces from two already generated prioritized reports

Components of the ABO Assistant

The ABO Assistant is comprised of these components:

- Batch SMF Analyzer
- Program Analyzer and Optimizer
- CICS SMF Analyzer

ABO Assistant – *Batch SMF Analyzer*

The Batch SMF Analyzer component of the ABO Assistant consists of 3 files:

- BOZSMFJ - Sample JCL for invoking the ABO Assistant's Batch SMF Analyzer programs. There are 4 input parameters that must be specified in this JCL:
  - SMFDUMP - Data set name of SMF DUMP to analyze containing SMF 30(4) records
  - ABO - Data set name of the ABO installation location (location of the BOZSMF program)
  - EXE - Data set name where the BOZSMFR REXX program is located
  - THRSHOLD - Minimum CPU time required for program to be included in the report
- BOZSMF - Program that is invoked by the sample job BOZSMFJ described above
- BOZSMFR - REXX program that is invoked by the sample job BOZSMFJ described above

Note: The SMFDUMP data must consist of SMF Type 30 subtype 4 records for the time interval you wish to analyze. See "SMF DUMP generation" on page 75 for information on how to generate this SMF dump.

After setting the 4 input parameters in BOZSMFJ and submitting the JCL, the Batch SMF Analyzer will produce a report at userid.BOZSMF.OUTPUT detailing the top CPU consuming programs, sorted high to low based on the CPU time.

Optionally, a list of data set locations where your COBOL run JCL files are located can be specified and the Batch SMF Analyzer tool will produce an additional report at userid.BOZSMF.OUTPUT.JCLLIST to also show the specific JCL location for each program found in the SMF data.

These JCL files can be used as input to the second component of the ABO Assistant, described next, to automatically optimize and produce a performance report on the CPU time savings from using ABO.

ABO Assistant – *Program Analyzer and Optimizer*

The Program Analyzer and Optimizer component of the ABO Assistant consists of 2 files:

- BOZPAJ - Sample JCL for invoking the ABO Assistant REXX program BOZPA. The following input parameters must be specified in this JCL:
  - JCLNAME - Name of the sequential file or PDS(E) member that contains the JCL to run the original COBOL program
  - PGMNAME - Name of the original COBOL program
  - ABO - Data set name of the ABO installation location
  - EXE - Data set name where the BOZPA REXX program is located
  - PROCNAME - Name of the sequential file or library member that contains the cataloged procedure that executes the pgmname program. If there is a chain of nested procedures, the one to specify is the cataloged procedure that contains the 'EXEC PGM=pgmname' card specified explicitly. A setting of N is the default when there is no cataloged procedure involvement in the pgmname program execution.
  - OPTLOAD - New or existing data set name where the optimized modules will be stored or TEMP to indicate a temporary data set should be used. TEMP is the default.

  - DYNSCAN - Y | N. A setting of Y will cause the modules that make up the application to be scanned for compilation details and no performance report will be produced. A setting of N is the default and will cause the performance report to be generated.
- BOZPA - REXX program that is invoked by the sample job BOZPAJ described above.

After setting the input parameters in BOZPAJ and submitting the JCL, the Program Analyzer and Optimizer actions will depend on the DYNSCAN setting.

Regardless of the DYNSCAN setting, the BOZPAJ job step will automatically build JCL and submit two jobs with the following job name format: userid1 and userid2.

For example, if userid is ABOEVAL, then the generated job names are ABOEVAL1 (job #1) and ABOEVAL2 (job #2).

- Job #1 is a modified version of your original JCL to enable profiling.
- Job #2 is a modified version of your original JCL to automatically optimize the top CPU consuming modules using ABO and to run the application using these optimized modules.

When the PROCNAME parameter is specified and it points to the cataloged procedure location, then the procedure is copied into the job #1 and job #2 JCL as an in-stream procedure.

When DYNSCAN=N is in effect (the default setting), a performance report is produced at userid.BOZPA.OUTPUT.pgmname. This report shows the time spent in the COBOL application and LE as well as the actual performance improvements from using ABO.

When DYNSCAN=Y is in effect, a report containing compilation details for all programs that dynamically make up the application is produced in the job #2 OPTLOG. This report contains the module and CSECT names, compilation dates, COBOL compiler versions or translator ID records and more for every module that contributes to the applications CPU time.

The report contents are the same as those described in "The log file for scanning" on page 50, except instead of statically scanning modules as with the ABO option SCAN=Y, the DYNSCAN=Y report provides the more targeted information of only those modules/programs actually used by the application.

The report produced by DYNSCAN=Y allows deep insight into the dynamic make up of your COBOL applications.

Notes:

1. When DYNSCAN=Y is in effect, no ABO optimization is performed, and no performance report is produced at userid.BOZPA.OUTPUT.pgmname. The OPTLOAD parameter is also ignored in this case.
2. ABO is automatically and transparently used by the Program Analyzer and Optimizer, so no user action is required to directly invoke ABO.
3. If no report is produced at userid.BOZPA.OUTPUT.pgmname (for DYNSCAN=N), see the SYSTSPRT file of job #1 or job #2 for error messages. See "BOZPAJ parameter error messages" on page 73 for a list of possible messages.

ABO Assistant – *CICS SMF Analyzer*

The CICS SMF Analyzer component of the ABO Assistant has two capabilities:

- Processing SMF 110 data to produce a prioritized report
- Comparison of two prioritized reports to produce a comparison report

Additionally, the CICS SMF Analyzer, contains a JCL sample to enable both parts above to be run together and produce two prioritized reports and one comparison report from two sets of SMF data.

Processing SMF 110 data to produce a prioritized report

The CICS SMF Analyzer contains two files, JCL sample BOZCICSJ and REXX program BOZCICSR, to produce a prioritized report from your SMF 110 data.

1. **BOZCICSJ** - Sample JCL for invoking the CICS DFH$MOLS sample utility program and the REXX program BOZCICSR to process your SMF 110 data.

The input parameters below must be specified in this JCL:

Parameter	Description
SMFDUMP	Data set name of SMF DUMP to analyze containing SMF 110(1) records
CICS	Data set name of CICS SDFHLOAD library location
MCT	Data set name of DFHMCTPA module location
EXE	Data set name where the BOZMCTPA and BOZSMFR REXX program is located
DUMPSTAT	ORIGINAL or OPTIMIZED, specifies whether the dump was taken for original or optimized programs run
COBAPPL	Name of tested COBOL CICS application
APPLID	CICS region application ID
SYSID	Four characters system ID of a system where SMF dump was taken
DATEJ	Date in YYYYDDD format when the SMF dump was taken
THRSHOLD	Minimum cumulative CPU time required for program to be included in the report

2. **BOZCICSR** - REXX program that is invoked by the sample job BOZCICSJ described above

   Note: The SMFDUMP data must consist of SMF Type 110 subtype 1 records for the time interval you wish to analyze. See Gathering SMF 110 Data for information on how to generate this SMF dump.

   After setting the above input parameters in BOZCICSJ and submitting the JCL, the CICS SMF Analyzer will produce a report at **userid.BOZCICS.cobappl.dumpstat** detailing the top CPU consuming programs, sorted high to low based on each program's cumulative CPU time.

Comparison of two prioritized reports

The CICS SMF Analyzer contains two files, JCL sample BOZCICPJ and REXX program BOZCICPR, to produce a comparison report to show the performance improvements between two sets of SMF 110 data (for example, to compare a run using the original modules to a run using the ABO optimized modules).

1. **BOZCICPJ** - Sample JCL for invoking the REXX program BOZCICPR.

   There are 4 input parameters that must be specified in this JCL:

Parameter	Description
ORIGINAL	Name of the first CICS SMF Analyzer prioritized report to compare. For example, from the run of the CICS application using the original modules (before ABO optimization).
OPTIMIZD	Name of the second CICS SMF Analyzer prioritized report to compare. For example, from the run of the CICS application using the ABO optimized modules.
EXE	Data set name where the BOZCICPR REXX program is located.
COBAPPL	Name of tested COBOL CICS application.

2. **BOZCICPR** - REXX program that is invoked by the sample job BOZCICPJ described above.

After setting the above input parameters in BOZCICPJ and submitting the JCL, a comparison report will be produced at **userid.BOZCICS.cobappl.GAIN**. This sorted comparison report details the CPU time changes overall, and per program, between the ORIGINAL and OPTMIZD prioritized reports. This report enables you to see how much ABO improved performance of your COBOL CICS applications.

Processing and comparing SMF data in one step

The CICS SMF Analyzer contains an additional sample JCL BOZCIAPJ. This JCL combines the processing and comparison parts described above. Using this combined processing requires that both SMF dumps generated for the original and optimized programs run are already available. If so, then this JCL can be used to generate the two prioritized reports for the original and optimized runs and also the comparison report for these two runs.

There are several input parameters that must be specified in the sample BOZCIAPJ JCL:

Parameter	Description
SMFDUMP1	Data set name of first SMFDUMP containing SMF110 records to process and compare. For example, from the run of the CICS application using the original modules (before ABO optimization).
SMFDUMP2	Data set name of second SMFDUMP containing SMF110 records to process and compare. For example, from the run of the CICS application using the ABO optimized modules.
CICS	Data set name of CICS SDFHLOAD library location
MCT	Data set name of DFHMCTPA module location
EXE	Data set name where the BOZMCTPA, BOZSMFR and BOZCICPR REXX program is located
COBAPPL	Name of tested COBOL CICS application
APPLID1	CICS region application ID matching SMFDUMP1 (e.g. for original programs run)
APPLID2	CICS region application ID matching SMFDUMP2 (e.g. for optimized programs run)
SYSID1	Four characters system ID of a system where SMFDUMP1 was taken
SYSID2	Four characters system ID of a system where SMFDUMP2 was taken
DATEJ1	Date in YYYYDDD format when the SMFDUMP1 was taken
DATEJ2	Date in YYYYDDD format when the SMFDUMP2 was taken
THRSHOLD	Minimum accumulative CPU time required for program to be included in the report

After setting the above input parameters in BOZCIAPJ and submitting the JCL, three reports are produced:

- A prioritized report for SMFDUMP1, located at userid.BOZCICS.cobappl.ORIGINAL, showing the top CPU consuming programs and transactions.
- A prioritized report for SMFDUMP2, located at userid.BOZCICS.cobappl.OPTIMIZD, showing the top CPU consuming programs and transactions.

- A comparison report, located at userid. BOZCICS.cobappl.GAIN, that details the CPU time changes overall, and per program, between the prioritized reports produced for SMFDUMP1 and SMFDUMP2.

How to use the ABO Assistant

This section contains instructions on how to use the two components of the ABO Assistant.

How to use the Batch SMF Analyzer

Four parameters must be set before submitting BOZSMFJ to run the Batch SMF Analyzer tool:

```
//  SET SMFDUMP=hlq.SMF30.DUMP          <- SMF TYPE 30 subtype 4 dump location
//  SET ABO=hlq.abo.loadlib             <- Program BOZSMF location
//  SET EXE=hlq.bozpa.location          <- REXX program BOZSMFR location
//  SET THRSHOLD=0                      <- Minimum CPU Time threshold value
```

Optionally, an additional report can be produced to display the JCL locations to run each program found in the SMF data. To enable this function, provide a list of existing JCL library locations at the PGMHOME in-stream JCLLIST DD file at the bottom of BOZSMFJ:

```
//PGMHOME  EXEC PGM=IKJEFT01,REGION=0M,PARM='BOZSMFR 2'
...
//JCLLIST  DD *
example.jcl.library1    <- Put your COBOL related JCL library here
example.jcl.library2    <- Put your COBOL related JCL library here
example.jcl.library3    <- Put your COBOL related JCL library here
```

When BOZSMFJ is submitted with these settings, the following actions take place:

The ABO Assistant REXX program **BOZSMFR** in **hlq.bozpa.location** and program BOZSMF in **hlq.abo.loadlib** will analyze your SMF data in **hlq.SMF30.DUMP** and produce a report at **userid.BOZSMF.OUTPUT** detailing the top CPU consuming programs, sorted high to low based on the CPU time.

If the JCLLIST function is enabled, then the provided JCL data set locations in the final BOZSMFJ step PGMHOME is used to find the JCL locations to run each program. These JCL file locations are displayed next to the sorted CPU data in an additional report at **userid.BOZSMF.OUTPUT.JCLLIST**.

If the JCLLIST function is not enabled or not successful, either by leaving the example data set locations unchanged or specifying an empty JCLLIST in-stream file, or if one or more data sets specified cannot be found, then the PGMHOME step completes with a return code of 4 and the additional report is not generated.

In most cases, only the final reports at BOZSMF.OUTPUT or BOZSMF.OUTPUT.JCLLIST will be required, but two earlier intermediate reports are also generated:

- userid.BOZSMF.OUTTEMP1 - All the performance related fields for all the programs from the SMF30 records
- userid.BOZSMF.OUTTEMP2 - All the performance related fields for non-excluded programs from the SMF30 records. The excluded programs are the records generated by STC/TSU and those in the default exclusion list. BOZSMFJ contains this default exclusion list.

See also “Example report from the Batch SMF Analyzer” on page 71 for the example reports.

How to use the Program Analyzer and Optimizer

The following parameters must be set before submitting BOZPAJ to run the Program Analyzer and Optimizer component of the ABO Assistant:

```
//  SET JCLNAME=hlq.JCL.LIB(COBJCL)     <- original COBOL program JCL location
//  SET PGMNAME=COBPGM                  <- COBOL program name
```

```
//  SET ABO=hlq.abo.loadlib              <- ABO install location
//  SET EXE=hlq.bozpa.location           <- REXX program BOZPA location
//  SET PROCNAME=N                       <- put hlq.lib(procname)
//  SET OPTLOAD=TEMP                     <- put data set name or TEMP
//  SET DYNSCAN=N                        <- for dynamic scan set to Y
```

The JCLNAME parameter can be obtained from the SMF Analyzer final report that can provide the JCL locations for the top CPU consuming programs.

When BOZPAJ is submitted with these settings, including DYNSCAN=N, the following actions take place:

When the PROCNAME parameter is specified and it points to the cataloged procedure location, then the procedure is copied into the job #1 and job #2 JCL as an in-stream procedure.

The ABO Assistant REXX program BOZPA in hlq.bozpa.location will submit job #1 to automatically profile and then run the program COBPGM in the original application JCL at hlq.JCL.LIB(COBJCL). Then job #2 will be submitted to optimize the ABO eligible COBOL CSECTs using the ABO at hlq.abo.loadlib and then run the application using the optimized modules. The performance results from analyzing and comparing the runs of the original and ABO optimized versions of the application will be generated in the ABO Assistant report. See "Example report from the Program Analyzer and Optimizer" on page 71.

Reminder: The names of job #1 and job #2 are based on the current userid. For example, if userid is ABOEVAL, then the generated job names are ABOEVAL1 (job #1) and ABOEVAL2 (job #2).

The ABO Assistant report is available at userid.BOZPA.OUTPUT.pgmname. For example, if the TSO ID of the user who submitted BOZPAJ is ABOEVAL and PGMNAME=COBPGM, then the ABO Assistant report is available at ABOEVAL.BOZPA.OUTPUT.COBPGM. If no report is produced at userid.BOZPA.OUTPUT.pgmname (when DYNSCAN=N is specified), see the SYSTSPRT file of job #1 or job #2 for error messages. See BOZPAJ Parameter Error Messages for a list of possible messages.

The OPTLOAD parameter specifies where the optimized modules will be stored. If left at the default value of TEMP, then the ABO Assitant will use a dynamically allocated temporary data set for the optimized modules. This temporary data set will be removed after the performance report is produced. If the OPTLOAD parameter specifies an existing data set name, then the optimized modules will be stored at this location. If the OPTLOAD parameter specifies the name of a data set that does not exist, then this data set will be dynamically allocated at hlq.OPTLOAD to contain the optimized modules. Specifying an existing or new data set name for OPTLOAD allows you to retain the optimized modules after the Program Analyzer and Optimizer completes to be used for further evaluation or deployment.

If DYNSCAN=Y is in effect, then a report detailing compilation information for the modules that dynamically make up the application is produced in the job #2 OPTLOG file. In this case, no performance report is generated to userid.BOZPA.OUTPUT.pgmname and the OPTLOAD parameter is also ignored. If there is no job #2 OPTLOG file produced, then see the SYSTSPRT file of job #1 or job #2 for error messages. See BOZPAJ Parameter Error Messages for a list of possible messages.

How to use the CICS SMF Analyzer

This section contains instructions on how to use the CICS SMF Analyzer component of the ABO Assistant.

How to use the CICS SMF Analyzer to produce a prioritized report

The following parameters must be set before submitting BOZCICSJ to produce a prioritized report from your SMF 110 data

```
// SET SMFDUMP=hlq.smf110.dump     <- SMF DUMP
// SET CICS=CICS.TS520.SDFHLOAD    <- CICS SDFHLOAD location
// SET MCT=hlq.personal.SDFHLOAD   <- DFHMCTPA location
// SET EXE=hlq.bozcicsr.location   <- REXX program BOZCICSR location
// SET DUMPSTAT=ORIGINAL           <- ORIGINAL or AFTER
// SET COBAPPL=cobappl             <- COBOL CICS application name
// SET APPLID=applid               <- CICS region APPLID
// SET DATEJ=yyyyddd               <- date of SMF DUMP
// SET SYSID=sysid                 <- SYSID of SMF DUMP
// SET THRSHOLD=0.0                <- CPU threshold value
```

When BOZCICSJ is submitted with these settings, the following actions take place:

- The DFHMNDUP CICS utility program generates a performance dictionary record
- The DFH$MOLS CICS utility program printing CICS monitoring data into the **userid.BOZCICS.DFH$MOLS.dumpstat** data set
- The CICS SMF Analyzer REXX program BOZCICSR in **hlq.bozcicsr.location** will analyze the CICS monitoring data provided within **userid.BOZCICS.DFH$MOLS.dumpstat** data set and produce a report at **userid.BOZCICS.cobappl.dumpstat** detailing the top CPU consuming programs, sorted high to low based on the program cumulative CPU time.

How to use the CICS SMF Analyzer to produce a comparison report

The following parameters must be set before submitting BOZCICPJ to produce a comparison report from two already generated prioritized reports:

```
// SET ORIGINAL=hlq.report.original  <- first prioritized report to compare
// SET OPTIMIZD=hlq.report.optimizd  <- second prioritized report to compare
// SET EXE=hlq.bozcicpr.location     <- BOZCICPR location
// SET COBAPPL=cobappl               <- COBOL CICS application name
```

When BOZCICPJ is submitted with these settings, the following actions take place:

The CICS SMF Analyzer REXX program BOZCICPR in **hlq.bozcicpr.location** will read the two prioritized reports and produce a comparison report at **userid.BOZCICS.cobappl.GAIN** to detail the CPU time changes overall, and per program, between the ORIGINAL and OPTMIZD prioritized reports.

How to use the CICS SMF Analyzer to produce prioritized reports and a comparison report in a single job

The following parameters must be set before submitting BOZCIAPJ to run the CICS SMF Analyzer to produce two prioritized reports from two SMF dumps and then also produce a comparison report of this data in a single job submission:

```
// SET SMFDUMP1=hlq.smf110.original <- first SMF DUMP (e.g. from original run)
// SET SMFDUMP2=hlq.smf110.optimizd <- second SMF DUMP (e.g. from optimized run)
// SET CICS=CICS.TS520.SDFHLOAD     <- CICS SDFHLOAD location
// SET MCT=hlq.personal.SDFHLOAD    <- DFHMCTPA location
// SET EXE=hlq.bozcicsr.location    <- BOZCICSR, BOZCICPR REXX location
// SET COBAPPL=cobappl              <- COBOL CICS application name
// SET APPLID1=applid               <- CICS APPLID for original run
// SET DATEJ1=yyyyddd               <- date of SMFDUMP1
// SET SYSID1=sysid                 <- SYSID of SMFDUMP1
// SET APPLID2=applid               <- CICS APPLID for optimized run
// SET DATEJ2=yyyyddd               <- date of SMFDUMP2
// SET SYSID2=sysid                 <- SYSID of SMFDUMP2
// SET THRSHOLD=0.0                 <- CPU threshold value
```

When BOZCIAPJ is submitted with these settings, the following actions take place:

- The CICS SMF Analyzer will produce a prioritized report at **userid.BOZCICS.cobappl.ORIGINAL** of the top CPU consuming programs from SMFDUMP1
- The CICS SMF Analyzer will then produce a prioritized report at **userid.BOZCICS.cobappl.OPTIMIZD** of the top CPU consuming programs from SMFDUMP2
- Finally, a comparison report between the prioritized reports generated for SMFDUMP1 and SMFDUMP2 will be produced at **userid.BOZCICS.cobappl.GAIN**. This comparison report details the CPU time changes overall, and per program, between the ORIGINAL and OPTMIZD prioritized reports produced in the first two steps. This comparison report is the same in format and content as that produced if the CICS SMF Analyzer steps described in the previous sections were individually run.

Example reports

This section contains example reports produced by the two components of the ABO Assistant.

Example report from the Batch SMF Analyzer

Below are example reports from the Batch SMF Analyzer.

These reports provide a sorted list, from high to low CPU time, of the application jobs and programs for the time range of SMF data that was collected.

This report enables you to quickly identify, at a high level, the applications consuming the most CPU time on your system. These applications would be the best candidates for ABO optimization.

Due to limitations in the SMF data, only the main or driver program name is listed in this report so the second component of the ABO Assistant, the Program Analyzer and Optimizer, should be used to identify and optimize all the programs that make up each application.

The following is an example report of userid.BOZSMF.OUTPUT:

```
    IBM Automatic Binary Optimizer for z/OS Assistant
    SMF Analyzer, 28 Jan 2021 12:35:13
    System: ZTFP     z/OS 2.3
    Range: JAN 21, 2021 01:00:01 - JAN 21, 2021 09:00:02
=======================================================
    Top CPU consuming programs
    * Listed by job step location
    * Sorted high to low by CPU Time

    JobName  ProgName StepName CPU sssss.hh
    G00960D  ATMCF000 DMS           8433.67
    G73745C  IKJEFT01 BATCHTSO      7791.43
    G77121F  TUYS200B SB330         6781.52
    G52682H  IKJEFT01 SB332         3384.34
    ...
    G46530T  PK8141SQ SB501           62.98
    G21044N  UT8080HG SB423           61.33
    Total CPU:                     83145.23
```

The following is an example report of userid.BOZSMF.OUTPUT.JCLLIST:

```
    IBM Automatic Binary Optimizer for z/OS Assistant
    Batch SMF Analyzer, 28 Jan 2021 12:35:13
    System: ZTFP     z/OS 2.3
    Range: JAN 21, 2021 01:00:01 - JAN 21, 2021 09:00:02
=======================================================
    Top CPU consuming programs
    * Listed by job step location
    * Sorted high to low by CPU Time

    JobName  ProgName StepName CPU sssss.hh Location
    G00960D  ATMCF000 DMS           8433.67 hlq1.PROD.LOAD1.JCL(TYDFD1)
    G73745C  IKJEFT01 BATCHTSO      7791.43 hlq1.PROD.LOAD2.JCL(GSDFD2)
    G77121F  TUYS200B SB330         6781.52 hlq2.TEST.LOAD.JCL(RDTR5)
    G52682H  IKJEFT01 SB332         3384.34 hlq2.TEST.LOAD.JCL(RDTR6)
    ...
    G46530T  PK8141SQ SB501           62.98 hlq1.PROD.LOAD1.JCL(UBBJ4)
    G21044N  UT8080HG SB423           61.33 NOT FOUND
    Total CPU:                     83145.23
```

In these examples, program ATMCF000 is identified as the top CPU consuming program. The JCL to run this application (hlq1.PROD.LOAD1.JCL(TYDFD1)) is then the input for the example Program Analyzer and Optimizer report shown next to determine the actual CPU reduction from ABO.

Example report from the Program Analyzer and Optimizer

Below is an example report from the Program Analyzer and Optimizer.

The report indicates the original CPU time when running the application, the CPU time after automatically optimizing the top CPU consuming CSECTs with ABO, and finally the actual CPU reduction from using ABO on this application.

```
================================================================================================
    IBM Automatic Binary Optimizer for z/OS Assistant
    Program Analyzer and Optimizer, 26 Feb 2021 14:06:04

    Program: ATMCF000
    z/OS version: 02 . 03
    LPAR name: SYSZEF1
    IBM Z system: z14 (3906-7F4)

=======   ORIGINAL RUN DETAILS   ================================================================

    Original CPU Time for ATMCF000: 8.90 seconds

    CPU consuming COBOL CSECTs:

    Module    CSECT     Product                 Status     %TOTAL     DSNAME

    ATMSTAC0  ATMSTAC0  Enterprise COBOL V4        *        35.95%    USER.ATM06.COBV42.OPTFULL.LOAD
    ATMSTCU0  ATMSTCU0  Enterprise COBOL V4        *        15.44%    USER.ATM06.COBV42.OPTFULL.LOAD
    IGZCPAC   IGZCXDI   COBOL LE                            14.56%    SYS2.CEEZ240.SCEERUN
              IGZCFCC   COBOL LE                             5.37%
              IGZCONVX  COBOL LE                             5.10%
              IGZCONV   COBOL LE                             2.05%
    ATMCF000  ATMCF000  Enterprise COBOL V4        *         1.52%    USER.ATM06.COBV42.OPTFULL.LOAD
    ATMM0000  ATMM0000  Enterprise COBOL V4        -         1.42%    USER.ATM06.COBV42.OPTFULL.LOAD
    ATMVITF0  ATMVITF0  Enterprise COBOL V5                  0.37%    USER.ATM06.COBV42.OPTFULL.LOAD
    ATMSTBR0  ATMSTBR0  Enterprise COBOL V4        *         0.31%    USER.ATM06.COBV42.OPTFULL.LOAD
    ATMURND0  ATMURND0  Enterprise COBOL V6                  0.07%    USER.ATM06.COBV42.OPTFULL.LOAD
    ATMCFEE0  ATMCFEE0  Enterprise COBOL V4        *         0.05%    USER.ATM06.COBV42.OPTFULL.LOAD
              ATMOFEE0  Enterprise COBOL V4        *         0.03%
              ATMRTALG  Enterprise COBOL V4        *         0.00%
    ATMRTCLG  ATMRTCLG  Enterprise COBOL V4        *         0.00%    USER.ATM06.COBV42.OPTFULL.LOAD

    Program ATMCF000 spends 82.24% of time in COBOL application & LE

=======   OPTIMIZED RUN DETAILS   ===============================================================

    Set ARCH=12 for z14

  * Optimize 8 COBOL CSECTs from 6 modules using ABO, store the modules in
USER.BOZPA.TEST.OPTLOAD

    ABO optimized CPU Time for ATMCF000: 03.84 seconds

=======   RESULTS   =============================================================================

    CPU time reduction using ABO: 56.85%

================================================================================================
```

Note: The character in column 4 (following the language or product name) indicates the CSECT's eligibility for optimization by ABO:

- An asterisk `*` indicates the CSECT is eligible for ABO optimization
- A space ` ` indicates the CSECT is not eligible for ABO optimization, because for example, it is part of Language Environment (LE), or is from a language not supported by ABO (e.g. PL/I) or that it is compiled by a version of the COBOL compiler not currently eligible for ABO optimization
- A dash `-` indicates the CSECT has already been optimized by ABO so is not eligible to be optimized again. Optimization by ABO can only happen from original CSECT

Example reports from the CICS SMF Analyzer

Below is a snippet from an example prioritized report at userid.BOZCICS.cobappl.dumpstat. The top CPU consuming programs are listed from high (program DSWFORVV) to low (program DSWTX1VV).

The total CPU time from all programs is listed at the bottom.

```
    IBM Automatic Binary Optimizer for z/OS Assistant
    CICS SMF Analyzer – Prioritized Report, 29 Apr 2021 14:32:42
    ApplID: SSJCICP1, System: ZTFJ z/OS 2.2
    Range: 2021/04/06 11:45:54 - 2021/04/06 11:58:22
============================================================
    Top CPU consuming programs
    * Sorted high to low by accumulated CPU Time

ProgName    TranName Count    CPU sssss.mmmmmm
DSWFORVV    /FOR     42776           0.886330
DSWPS3VV    PS3      2861            0.436512
DSWPX3VV    PX3      2747            0.423158
```

```
DSWIX8VV    IX8      2467            0.376860
...
DSWTX1VV    TX1      1242            0.031920
Total CPU:                           7.287035
```

Below is a snippet from an example comparison report at userid.BOZCICS.cobappl.GAIN. The top CPU consuming programs are listed high(program DSWFORVV) to low (program DSWTX1VV). The total CPU across all programs is listed at the bottom.

```
    IBM Automatic Binary Optimizer for z/OS Assistant
    CICS SMF Analyzer – Comparison Report, 30 Apr 2021 10:25:04
    ApplID: SSJCICP1, System: ZTFJ z/OS 2.2
================================================================================
    Top CPU consuming original programs
    * Sorted high to low by Reduction of Total
    * Compared with optimized programs CPU Time

Program  Transact  CPU Original  CPU Optimized  %Reduction  %Reduction
Name     Name      sssss.mmmmmm  sssss.mmmmmm                of Total
--------------------------------------------------------------------------------
Total CPU:             7.287035      6.600909      9.42%     100.00%
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DSWFORVV /FOR          0.886330      0.727202     17.95%      23.19%
DSWDE2VV DE20+         0.299734      0.257184     14.20%       6.20%
DSWDX2VV DX20+         0.296540      0.255306     13.91%       6.01%
DSWOE2VV OE2           0.134240      0.100301     25.28%       4.95%
…
DSWSC6VV SC6           0.257253      0.255454      0.70%       0.26%
DSWIT8VV IT8           0.370085      0.390281     -5.46%      -2.94%
DSWIX8VV IX8           0.376860      0.399796     -6.09%      -3.34%
-   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
The 5th column shows the % reduction in CPU in each row
%Reduct = 100 - (CPUOptim * 100 / CPUOrig)
The 6th column shows the % contribution of each row to the total CPU reduction
%ReductOfTotal = 100 * (CPUOrig - CPUOptim) / (TotalCPUOrig - TotalCPUOptim)
```

BOZPAJ parameter error messages

Any errors in the BOZPAJ parameters will cause a BOZPAJ job failure with return code 8 and the **SYSTSPRT** file will have one of the following messages:

```
*** Error: no JOB card provided in jclname
*** Error: no EXEC card provided in jclname
*** Error: no step with PGM=pgmname located in jclname
*** Error: no step with PGM=pgmname located in jclname and procname
*** Error: invalid data set jclname
*** Error: data set jclname could not be found
*** Error: data set ABO could not be found
*** Error: invalid OPTLOAD optload
*** Error: invalid DYNSCAN dynscan
*** Error: allocating data set optload
```

Limitations and requirements on Program Analyzer and Optimizer

The Program Analyzer and Optimizer component of the ABO Assistant beta has some limitations and requirements on the types and format of JCL it can process and the application environments where it can operate.

Unless otherwise indicated, all the error messages are placed in the SYSTSPRT file from job #2.

1. The ABO Assistant can be used on LE enabled batch applications. The ABO Assistant does not work for CICS or IMS applications. The ABO Assistant will not work if your z/OS operating system is running on z/VM.
2. The target COBOL application must be re-runnable sequentially without cleanup or other steps needed that are not already present in the original JCL. This is required as two runs of the application (original and ABO optimized) are required to form the performance comparison. The errors generated in case of

some resource missing or a skipped cleanup step will vary depending on the missing action during the second run of the application.

User response:

Modify the original JCL so all required cleanup or other steps are included to enable a sequential rerun of the application

3. The COBOL program specified by the pgmname variable, as well as, any other COBOL program implicitly called must be in the STEPLIB data set concatenation for it to be available for ABO optimization.

   If a module is not found in the STEPLIB, but still found in the LINKLIST or any other system library, then the ABO Assistant will continue, but the corresponding module will not be optimized, and will be marked as 'not found in STEPLIB' in the ABO Assistant report.

   If the module is not found at all then ABEND 806 will be generated.

   User response:

   Add the missing data set(s) containing all modules required for the application to the STEPLIB concatenation

4. As the ABO Assistant uses ABO it has the same system and hardware requirements as ABO does. If the system used for any of generated jobs #1 or #2 is not supported, the BOZPAJ job will fail with return code 8 and one of the following error messages will be generated.

   Messages:

   ```
   z/OS version: xx.xx is not a supported z/OS version to run ABO.
   ```

   IBM z server: (xxxx) is not a supported hardware level to run ABO optimized modules.

   User Response:

   Ensure the system being used for the ABO Assistant meets the minimum requirements for ABO.

   Note: The ABO Assistant automatically detects the IBM Z system where it is being run and sets the ARCH level for ABO optimization to match the IBM Z system.

5. The target COBOL program name provided in the original JCL with 'PGM=' parameter cannot be a variable. For example, the following are not allowed:

   ```
   //STEP1 EXEC PGM=&pgm
   ```

   or

   ```
   PGM=%pgm
   ```

   If variable name is used in the 'PGM=' field, or there is no step with the specified COBOL program name pgmname found in the original JCL, the BOZPAJ job fails with return code 8 and an error message is generated.

   This error can occur if your JCL contains cataloged JCL procedures call but parameter PROCNAME=N was used in your BOZPAJ configuration. This error can also happen if the PROCNAME=hlq.lib(procname) specified in your BOZPAJ configuration points to a cataloged procedure that has no 'EXEC PGM=pgmname' card specified explicitly but has a nested cataloged procedure call instead.

   Message:

   ```
   *** Error: no step with PGM=pgmname located in jclname [ and procname]
   ```

   User response:

   Correct the JCL so a variable is not used for the program name. If your JCL contains cataloged JCL procedures call, then configure the BOZPAJ parameter PROCNAME to point to the sequential file or library member that contains the cataloged procedure that executes the pgmname program. If there

is a chain of nested procedures, the one to specify is the cataloged procedure that contains the 'EXEC PGM=pgmname' card specified explicitly.

6. If the original job JCL has system affinity or routing directives, such as CLASS, SYSAFF, ROUTE, SCHENV, these may cause the job to be routed for execution on a system that satisfies the requirements.

   Depending on routing directives there may be a different system selected each time. This may cause the automatically generated jobs #1 and #2 to be run on different systems

   In case the IBM mainframe level of the jobs #1 and #2 runs are not the same, then the job #2 completion code will be 4 and a warning message will be generated at the end of the ABO Assistant report.

   Message:

   ```
   *** Warning: the system ARCH level for POC JOB # 1 and # 2 is nn nn
   respectively

   The evaluation results may not be accurate.
   ```

   User response:

   Adjust the original JCL routing directives to ensure the same single system selection for each job

7. The TSO user invoking the ABO Assistant must be authorized to use the TSO SUBMIT command. If not, then the tool's use of this command will fail, and the BOZPAJ job will fail with return code 8 and an error message generated.

   Message:

   ```
   *** Error: TSO SUBMIT command failure
   ```

   User response:

   Obtain TSO user authorization for using TSO SUBMIT

8. The STEPLIB JCL card must be present in the target step (step with EXEC PGM=pgmname) of the original JCL. If no STEPLIB card is found, then the BOZPAJ job fails with return code 8 and an error message is generated.

   Message:

   ```
   *** Error: no STEPLIB provided for stepname step in jclname
   ```

   User response:

   Add the required STEPLIB card to the JCL

Additional usage notes

The following are some additional usage notes:

- You can use the SDSF SJ command in front of jobs #1 and #2 to see and possibly reuse the JCL internally generated for those jobs by the ABO Assistant.
- The complete RTI profiles generated by jobs #1 and #2 for your original and ABO optimized COBOL programs can be found in the following locations for program `pgmname`:
  - Original RTI Profile: `userid.SYSPROFD.ORIG(pgmname)`
  - ABO optimized RTI profile: `userid.SYSPROFD.OPTIM(pgmname)`

SMF DUMP generation

The SMF dump used as the input to the SMF Analyzer component of the ABO Assistant can be produced by either the IFASMFDP or IFASMFDL program depending on your system installation and your system programmer recommendations.

Sample 1:

```
//IFASMFDP  JOB   accounting information
//STEP      EXEC  PGM=IFASMFDP
//INDD1     DD    DSN=SYS1.MANB,DISP=SHR
//OUTDD1    DD    DSN=&SYSUID..SMFDUMP,DISP=(,CATLG,DELETE),UNIT=SYSDA,
//     SPACE=(CYL,(10,10),RLSE),DCB=(LRECL=32760,RECFM=VBS,BLKSIZE=0)
  //SYSPRINT  DD    SYSOUT=*
//SYSIN     DD    *
    INDD(INDD1,OPTIONS(DUMP))
    OUTDD(OUTDD1,TYPE(30(4)))
    DATE(2020151,2020151)
    START(1200)
    END(1600)
/*
```

The above job produces &SYSUID..SMFDUMP that contains the SMF 30 subtype 4 records collected by system in the data set SYS1.MANB on day 151 of year 2020, from 12:00 to 16:00.

Note: Consult your system programmer to determine the correct data set name for the INDD1 statement.

Sample 2:

```
//IFASMFDL  JOB   accounting information
//STEP      EXEC  PGM=IFASMFDL
//OUTDD1    DD    DSN=&SYSUID..SMFDUMP,DISP=(,CATLG,DELETE),UNIT=SYSDA,
//     SPACE=(CYL,(10,10),RLSE),DCB=(LRECL=32760,RECFM=VBS,BLKSIZE=0)
  //SYSPRINT  DD    SYSOUT=*
//SYSIN     DD    *
    LSNAME(IFASMF.SMFTO30.SYSA,OPTIONS(DUMP))
    OUTDD(OUTDD1,TYPE(30(4)))
    DATE(2020151,2020151)
    START(1200)
    END(1600)
/*
```

The above job produces &SYSUID..SMFDUMP that contains the SMF 30 subtype 4 records collected by system in the log stream IFASMF.SMFTO30.SYSA on day 151 of year 2020 from 12:00 to 16:00.

Note: Consult your system programmer to determine the right log stream name for the LSNAME statement. In the above sample, the appropriate log stream name on LPAR SYSA is IFASMF.SMFTO30.SYSA.

Gathering SMF 110 data

Below are the instructions to gather the correct records in the SMF data for use in the CICS SMF Analyzer.

STEP 1

To enable the generation of SMF record type 110 data add SMF type 110 subtype 1 into your system SYS1.PARMLIB(SMFPRMxx) member.

For example SYS1.PARMLIB(SMFPRM11):

```
ACTIVE
DSNAME(SYS1.MAN1.&NAMESYS,
SYS1.MAN2.&NAMESYS,
SYS1.MAN3.&NAMESYS)
NOPROMPT
...
SID(&SYSNAME(1:4))
LISTDSN                    /* LIST
SYS(TYPE(30,110(1)),       <- introduce it here
     EXITS(IEFACTRT,IEFU29,IEFUSI),
     NOINTERVAL,NODETAIL)
...
```

Note that adding SMF 110(1) into your parmlib will not automatically activate those records generation. See STEP 3 for activation details.

STEP 2

Before starting the monitoring of SMF110 performance records generate a MCT (Monitoring Control Table) to reduce the amount of performance data generated to the fields required by the CICS SMF Analyzer. This step is required so STEP 3 and STEP 4 that follow generate properly formatted input for the CICS SMF Analyzer.

Use job BOZAUPLE sample to generate the DFHMCTPA MCT table using the BOZMCTPA member supplied included in the ABO sample library.

- Customize job BOZAUPLE by setting the following variables:

```
// SET EXE=hlq.lib(BOZMCTPA)          <- put BOZMCTPA location here
// SET INDEX=hlq.CICS.TS520.CICS      <- put CICS lib prefix here
// SET INDEX2=hlq.personal.loadlib    <- put your lib prefix here
```

  The full data set name will be generated by adding '.SDFHLOAD' suffix to the prefix provided in variable INDEX and INDEX2.
- Submit the BOZAUPLE job. It will generate DFHMCTPA load module in your personal load library (pointed by SYSLMOD DD of the LNKEDT step of that job)

STEP 3

To activate the CICS region SMF records generation define the following CICS initialization parameters in the SYSIN data set of the CICS startup job stream:

- MCT to specify suffix of your MCT table
- MN to activate monitoring
- MNPER to activate performance records monitoring

Note: In the scope of this procedure do not introduce in SYSIN any other monitoring related parameters besides the listed above MCT, MN and MNPER. Also, do not leave in SYSIN any existing related to monitoring parameters other than 3 listed above. If, for example, your current SYSIN already has MNRES, MNDIN or MNEXC=ON statement it should be removed for the purpose of this procedure.

For example:

```
//SYSIN    DD *
...  <- leave not related to monitoring existing parameters
...  <- and add the 3 new ones displayed below
MCT=PA
MN=ON
MNPER=ON
/*
```

Concatenate your personal load library containing DFHMCTPA into DFHRPL DD of the CICS region JCL on top of your existing SDFHLOAD library.

For example, if your library name containing DFHMCTPA is userid.CICS.TS520.CICS.SDFHLOAD and your existing SDFHLOAD library is hlq.CICS.TS520.CICS.SDFHLOAD:

```
...
//DFHRPL   DD DISP=SHR,DSN=...SEYULOAD
//         DD DISP=SHR,DSN=userid.CICS.TS520.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=hlq.CICS.TS520.CICS.SDFHLOAD
```

STEP 4

- a) Start your CICS region using the updated JCL.
- b) Run your CICS COBOL application.

  While your application is running issue from SDSF the following command: /D SMF. It will display the status of allocated by system 'MAN' data sets. For example:

```
RESPONSE=SP0A
 IEE974I 09.53.29 SMF DATA SETS 644
```

```
           NAME                VOLSER SIZE(BLKS) %FULL  STATUS
         P-SYS1.MAN1.SYSTEMA   WRKA0C    360000     0  ALTERNATE
         S-SYS1.MAN2.SYSTEMA   WRKA04    360000    30  ACTIVE
         S-SYS1.MAN3.SYSTEMA   WRKA06    360000     0  ALTERNATE
```

Note: As per the above display the system is now actively writing SMF records into SYS1.MAN2.SYSTEMA as it is 30% full and has status ‘ACTIVE’. Therefore, the character ‘x’ in the data set name SYS1.MANx.SYSTEMA in the below JCL example must be updated to ‘2’ at this step.

- c) Wait for your CICS COBOL application completion.

  Then issue from SDSF the following command: /I SMF. This command will force system to stop writing to the currently active MANx data set (MAN2 in our example) and start writing to the next available one. The /D SMF command issued right after the /I SMF will reflect the new MAN2 data set status as ‘DUMP REQUIRED’.

  For example:

```
RESPONSE=SP0A
 IEE974I 10.50.20 SMF DATA SETS 172
           NAME                VOLSER SIZE(BLKS) %FULL  STATUS
         P-SYS1.MAN1.SYSTEMA   WRKA0C    360000     1  ACTIVE
         S-SYS1.MAN2.SYSTEMA   WRKA04    360000    34  DUMP REQUIRED
         S-SYS1.MAN3.SYSTEMA   WRKA06    360000     0  ALTERNATE
```

- d) Right after the last /I SMF command issued, as fast as you can, submit the job to capture the SMF dump. If this action is not taken quickly enough then the system will take over control of the required input data set SYS1.MANx and then the job will fail with code 8 trying to open that data set.

  For example:

```
// SET DUMPSTAT=ORIGINAL  <- use ORIGINAL or OPTIMIZD qualifier
//SMFUNLD  EXEC PGM=IFASMFDP
//INDD1    DD
DSN=SYS1.MANx.SYSTEMA,DISP=SHR,AMP=('BUFSP=65536')

//OUTDD1 DD DSN=&SYSUID.SMF110.&DUMPSTAT,
//  DISP=(,CATLG,DELETE),UNIT=SYSDA,
//  SPACE=(CYL,(200,50),RLSE),
//  DCB=(LRECL=32760,RECFM=VBS,BLKSIZE=0)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
        INDD(INDD1,OPTIONS(DUMP))
        OUTDD(OUTDD1,TYPE(110(1)))
/*
```

  As per the above sample, the dump file userid.SMF110.ORIGINAL is generated.

  Notes:

  1. It’s recommended to use ORIGINAL or OPTIMIZD qualifier in the dump name for your convenience to clarify whether the dump was taken for the original or optimized programs run.
  2. The above job example may not be applicable for your system in case it is using log stream instead of MANx data sets for SMF recording. Then contact your system programmer to get the name of the right log stream for the SMF 110 records. The above a, b, c and d steps are still applicable without the expanded description in step b and c related to /D SMF & /I SMF technique, because all SMF activity would be available in a single log stream at this time. So, only the *yourlogstreamname* substitution will be required in the job capturing the dump.

  For example:

```
// SET DUMPSTAT=ORIGINAL  <- use ORIGINAL or OPTIMIZD qualifier
//SMFLG     EXEC  PGM=IFASMFDL
//OUTDD1 DD DSN=&SYSUID.SMF110.&DUMPSTAT,
//  DISP=(,CATLG,DELETE),UNIT=SYSDA,
//  SPACE=(CYL,(200,50),RLSE),
//  DCB=(LRECL=32760,RECFM=VBS,BLKSIZE=0)
//SYSPRINT  DD    SYSOUT=*
//SYSIN     DD    *
  LSNAME(yourlogstreamname,OPTIONS(DUMP))
```

```
  OUTDD(OUTDD1,TYPE(110(1)))
  DATE(2021056,2021056)
  START(1100)
  END(1200)
// *
```

Chapter 8. Managing optimization and optimized module deployment process

Taking an iterative and staged approach when using ABO

An iterative and staged approach to use ABO is a recommended approach to balance the cost of the optimization process to the benefits from running the optimized programs generated by ABO.

For example, first optimize the modules containing the top x% contributors to CPU time. Measure the impact (for example, the reduction in CPU time) using these ABO generated modules, and then repeat for the next top x% CPU contributors until your performance goals are met.

Included with ABO is the Chapter 7, “Using the ABO Assistant,” on page 63. The ABO Assistant automates all the individual steps required to efficiently optimize your COBOL batch application using ABO and to clearly report on the CPU time savings from using ABO. The ABO Assistant can be used when evaluating ABO to determine potential CPU savings and also to prioritize deployment of applications and determine the top CPU consuming modules and CSECTs.

There are also stand-alone performance measurement and reporting tools that can be used, such as IBM Application Performance Analyzer (APA) for z/OS, to help determine the top CPU contributors. Alternatively, the RTI Profiler that comes with ABO can be used for COBOL batch applications to help determine which modules and CSECTs are most frequently executed when the application is running.

Characteristics of programs that benefit most from ABO

Some compiled programs will benefit more from ABO than others. Knowing some key characteristics of these programs can also help in staging use of ABO on your compiled COBOL programs.

ABO can only improve performance of the original compiler generated code and some select Language Environment (LE) routines, but ABO does not have the means to improve performance when time is spent in other subsystems such as CICS, Db2, and IMS.

Key characteristics of programs that might benefit more from optimization with ABO are:

- A significant portion of the application’s execution time is spent in the COBOL code instead of in other subsystems such as CICS, Db2, and IMS.
- The COBOL code is performing a significant amount of computations. For example, a program where the COBOL code itself is doing the actual, real work, and is not simply acting as a "driver" program for other programs or subsystems.
  - At the source level, some statements most likely to benefit include, but are not limited to: COMPUTE, IF, MOVE, ADD, SUBTRACT, MULTIPLY, DIVIDE, and REMAINDER.
  - In addition, some select Language Environment (LE) routines can also be optimized by ABO. These routines perform a variety of conversion, move, and arithmetic operations and include IGZCSH2, IGZCFPC, IGZCONV, IGZCVMO, IGZCXPR, IGZCXMU, and IGZCXDI. ABO optimizes these routines by more efficiently performing the work of these routines directly in the optimized code or by calling a more efficient LE routine.

    Note: Looking at the COBOL source alone does not take into account where the time in the application is actually spent, so this should be done in combination with a performance report from an analysis tool such as APA.
- Most COBOL modules within the application are eligible for optimization by ABO. This means that the modules were complied with an eligible COBOL compiler and contain language features that are supported by ABO.

Optimization and deployment usage scenarios

This section contains three typical usage scenarios for IBM Automatic Binary Optimizer for z/OS. These scenarios describe possible approaches to using the IBM Automatic Binary Optimizer for z/OS to improve the performance of already compiled IBM COBOL programs. Each scenario provides step-by-step instructions to enable you to optimize your compiled IBM COBOL programs.

Scenario 1: Optimization process with static deployment

In this usage scenario, specify input modules to the optimizer in your JCL using BOPT directives, and for deployment, update all existing JCL that identifies data sets containing the original modules.

To perform the optimization with static deployment, complete the following steps:

1. Create new data sets. For example, the following data sets are created for this scenario where *HLQ* is the high-level qualifier that you define.
   - *HLQ*.OUT.LOAD.Z14. This data set will be populated with optimized binaries targeting the z14 machine.
   - *HLQ*.OUT.LOAD.Z15. This data set will be populated with optimized binaries targeting the z15 machine.
2. Run ABO to populate the new data sets. To run the optimizer, create new JCL. In the in-stream line that starts with SYSIN, use the BOPT optimizer directive. Select your compiled COBOL programs to optimize with the IN option. For example, the following JCL instructs the optimizer to optimize all the members with names beginning with M from *HLQ*.IN.LOAD. The optimized binaries targeting z14 and z15 are placed in *HLQ*.OUT.LOAD.Z14 and *HLQ*.OUT.LOAD.Z15 respectively.

   ```
   ...
   //SYSIN    DD *
     BOPT IN=HLQ.IN.LOAD(M*) OUT=HLQ.OUT.LOAD.Z14 LIST=HLQ.OUT.LIST.Z14 ARCH=12
     BOPT IN=HLQ.IN.LOAD(M*) OUT=HLQ.OUT.LOAD.Z15 LIST=HLQ.OUT.LIST.Z15 ARCH=13
   ```

   The example is intended to reflect what the user should specify in the SYSIN file. For basic JCL configuration, see Appendix A, “JCL sample,” on page 91. For more sample JCL that you can use in the static deployment scenario, see “Specifying optimization with BOPT” on page 36.
3. To run the optimized programs, modify the JCL that is used to run the original programs. That JCL identifies data sets that contain the original modules. In the STEPLIB setting, you must place the data sets of the optimized modules ahead of the data sets of the original modules for each targeted architecture. The following snippets show the modified parts in JCL that points to the optimized and original program binaries.

   Here's the modified part in the JCL that is used to run the original program on z14:

   ```
   //STEPLIB DD DSN=HLQ.OUT.LOAD.Z14,DISP=SHR
   //        DD DSN=HLQ.IN.LOAD,DISP=SHR
   ...
   ```

   Here's the modified part in the JCL that is used to run the original program on z15:

   ```
   //STEPLIB DD DSN=HLQ.OUT.LOAD.Z15,DISP=SHR
   //        DD DSN=HLQ.IN.LOAD,DISP=SHR
   ...
   ```

Scenario 2: Optimization process with dynamic deployment

In this usage scenario, map input to output modules in IEFOPZxx SYS1.PARMLIB's members and then use the IEFOPZ optimizer directive to specify optimization of input to output modules. After the binary optimization completes, run the optimized programs with no changes to the existing JCL that was used to run the original programs.

IEFOPZxx contains statements that define the data set optimization configuration which provide a list of pairings of an old COBOL library and the intended new libraries (one for each desired architecture

level) and specifies which members are to be processed (optimized). For more information, see *z/OS MVS Initialization and Tuning Reference*.

To perform the binary optimization with dynamic deployment, complete the following steps:

1. Create new data sets. For example, the following data sets are created for this scenario where *HLQ* is the high-level qualifier that you define.
   - *HLQ*.OUT.LOAD.Z14. This data set will be populated with optimized binaries targeting the z14 machine.
   - *HLQ*.OUT.LOAD.Z15. This data set will be populated with optimized binaries targeting the z15 machine.
2. Define the IEFOPZ configuration.
   a. Create an IEFOPZxx member.
   b. For each old data set that contains the compiled module that you want to optimize, define an OLD/NEW pair in the IEFOPZxx member. Mark the OLD/NEW pair as INACTIVE so that the system does not do any OLDNEW processing unexpectedly. See the following example:

```
MAXARCH(13)
CHECKALL
OWNER(IBM) MINARCH(12)
OLDNEW(
    OWNER(IBM)
    OLD (DSN(HLQ.IN.LOAD))
    NEW (DSN(HLQ.OUT.LOAD.Z14)   ARCH(12))
    NEW (DSN(HLQ.OUT.LOAD.Z15)   ARCH(13))
    INCLUDEMEMBERS(M*) //Identifies to process all members beginning with M
    INACTIVE
 )
```

   Note: The OLD/NEW pairs can be defined in one or multiple IEFOPZxx members.
3. To activate the IEFOPZ configuration, use the following z/OS MVS System command:

```
SET IEFOPZ=(x1,...,xn)
```

   where x_1,..., x_n are the suffixes xx for the IEFOPZxx members. If in the previous step, you only create one member, the command is as follows:

```
SET IEFOPZ=x1
```

   Note: The SET command modification stays through the current IPL session only. Therefore, it is usually used for the new configuration quick test, or to override some permanent definitions during the current IPL session. For permanent configuration definitions, see step 5.
4. Run IBM Automatic Binary Optimizer for z/OS to populate the new data sets.

   To run the optimizer, write JCL as follows. In the in-stream data that starts with SYSIN, use the IEFOPZ directive.

```
...
//SYSIN    DD *
  IEFOPZ SEL_ARCH=12 LIST=HLQ.BOZOPT.ARCH12.LIST
  IEFOPZ SEL_ARCH=13 LIST=HLQ.BOZOPT.ARCH13.LIST
```

   The example is intended to reflect what the user should specify in the SYSIN file. For basic JCL configuration, see Appendix A, “JCL sample,” on page 91. For more sample JCL that you can use in the dynamic deployment scenario, see “Specifying optimization with IEFOPZ” on page 38.
5. Update your IEASYSxx SYS1.PARMLIB member with the IEFOPZ system parameter so that subsequent IPLs will properly activate the desired IEFOPZ configuration. For example, to have the IEFOPZ configuration specified in member IEFOPZ99 automatically activated with each subsequent IPL, put the IEF0PZ=99 statement into your IEASYSxx member. However, the SET command described in step 3, if issued for example as follows: SET IEF0PZ=99, will activate the desired IEFOPZ99 member for the current IPL session only.

6. Redefine the OLD/NEW pairs as ACTIVE. If you want OLD/NEW processing to be done for any DDNAMEs other than JOBLIB and STEPLIB, define those within an IEFOPZxx parmlib member using the DDNAME statement. Then activate that updated IEFOPZ configuration.
7. Run the optimized programs by using the existing JCL that was used to run the original programs.

Related reference

"Related publications" on page 135

Scenario 3: Optimization process using a hybrid approach

In the hybrid approach, specify the input binaries to optimize explicitly in your JCL as what you do in Scenario 1, but combine with dynamic deployment demonstrated in Scenario 2. With dynamic deployment, run the optimized modules without changing the existing JCL.

To perform the binary optimization using a hybrid approach, complete the following steps:

1. Create new data sets. For example, the following data sets are created for this scenario where *HLQ* is the high-level qualifier that you define.
   - *HLQ*.OUT.LOAD.Z14. This data set will be populated with optimized binaries targeting the z14 machine.
   - *HLQ*.OUT.LOAD.Z15. This data set will be populated with optimized binaries targeting the z15 machine.
2. Run IBM Automatic Binary Optimizer for z/OS to populate the new data sets.

   To run the optimizer, create new JCL. In the in-stream data that starts with SYSIN, use the BOPT optimizer directive to select compiled COBOL modules to optimize. For example, the following JCL instructs the optimizer to optimize all the members that begin with the letter M from *HLQ*.IN.LOAD. The optimized binaries targeting z14 and z15 are placed in *HLQ*.OUT.LOAD.Z14 and *HLQ*.OUT.LOAD.Z15 respectively.

   ```
   ...
   //SYSIN    DD *
     BOPT IN=HLQ.IN.LOAD(M*) OUT=HLQ.OUT.LOAD.Z14  ARCH=12
     BOPT IN=HLQ.IN.LOAD(M*) OUT=HLQ.OUT.LOAD.Z15  ARCH=13
   ```

   The example is intended to reflect what the user should specify in the SYSIN file. For basic JCL configuration, see Appendix A, "JCL sample," on page 91. For more sample JCL that you can use in the hybrid scenario, see "Specifying optimization with BOPT" on page 36.
3. Define the IEFOPZ configuration.
   a. Create an IEFOPZxx member.
   b. For each old data set that contains the compiled module that you want to optimize, define an OLD/NEW pair in the IEFOPZxx member. Mark the OLD/NEW pair as ACTIVE. See the following example.

      ```
      OLDNEW (
         OLD( DSNAME (HLQ.IN.LOAD) )
         NEW( DSNAME (HLQ.OUT.LOAD.Z14) ARCH(12) )
         NEW( DSNAME (HLQ.OUT.LOAD.Z15) ARCH(13) )
       INCLUDEMEMBERS(M*)  //Identifies to process all members beginning with M
       ACTIVE )
      ```

      Note: The OLD/NEW pairs can be defined in one or multiple IEFOPZxx members.
   c. If you want OLDNEW processing to be done for any DDNAMEs other than JOBLIB and STEPLIB, define those within an IEFOPZxx parmlib member using the DDNAME statement. Then activate that updated IEFOPZ configuration by using the following command:

      ```
      SET IEFOPZ=(x1,...,xn)
      ```

where x_1,..., x_n are the suffixes xx for the IEFOPZxx members. If in the preceding step, you create only a single member, the command is as follows:

```
SET IEFOPZ=x1
```

4. Run the optimized programs by using the existing JCL that was used to run the original programs.

Related reference

"Related publications" on page 135

Testing information

The optimized modules that ABO produces will run faster but will have the same behavior, except from some isolated error message and abend code differences, as the original COBOL modules. ABO is able do this because it processes the binary code within the COBOL module so it is able to ensure the low level logic of the program remains the same. This means that users of ABO do not have to perform full functional verification testing of the ABO optimized modules. Some limited testing is recommended to ensure basic functioning of the applications using the ABO optimized modules prior to deploying ABO optimized modules into a production environment.

Performance testing is best done in a controlled environment with the same input data used with the original application and with the application containing ABO optimized modules. Using a machine or LPAR that has as few as possible other applications running for performance testing will allow for stable and reproducible performance results. Comparing the CPU time between the original application and optimized application is the best way to see performance gains.

Chapter 9. Resolving problems with optimization and optimized module deployment

Resolving problems that occur during optimization time

The return code that ABO passes to z/OS is an indicator of whether a problem was encountered during optimization. A return code value of zero means that optimization was successful and no problems were encountered. A return code value other than zero, indicates something unexpected occurred or a problem was encountered. For more information on return codes, see Appendix B, “Return codes,” on page 93.

ABO produces output files that can be used to diagnose problems.

The following files can be helpful in diagnosing problems encountered during the optimization of COBOL programs:

- The log file gives a summary of what has been optimized or scanned, and error messages if applicable. See Messages for more information. The log file is the first place you should examine if problems are detected during the optimization process.
- The file specified by the OPTERR DD are written to in exceptional circumstances. If the OPTERR DD is not specified, those messages are written to the JOBLOG.
- The file specified by the CEEDUMP DD are written to in circumstances such as a program exception when running ABO. The CEEDUMP file is produced by Language Environment (LE) and includes information such as a traceback of procedures that were executing at the time of the abend.
- The JOBLOG includes additional diagnostic messages that complement error messages that were written elsewhere or the JOBLOG can be a default location for errors encountered in exceptional circumstances.

Resolving problems encountered during execution

Common execution error solutions

ABO optimized modules might fail with a U4038 abend if the "Language Environment Automatic Binary Optimizer Runtime Engine" PTF installed is not the latest PTF available. In this case, the module will output one of the following messages:

```
IGZ0153S Program BOZSRC1 was compiled with a level of the compiler that requires service to be
installed on Language Environment.
IGZ0355S Program BOZSRC1 was optimized with a level of the Automatic Binary Optimizer that
requires service to be installed on Language Environment.
```

PTFs on z/OS 2.3 will cause the first message to be issued, and PTFs on z/OS 2.4 will cause the second message to be issued. To resolve this problem, the latest "Language Environment Automatic Binary Optimizer Runtime Engine" PTF listed in the Program Directory should be installed. The latest information about the ABO PTFs can also be found on the fix list and new features page.

An 0C1 abend occurs if you attempt to run the ABO generated modules on a system that is not supported by ABO. See “Target hardware levels” on page 4 for the supported systems.

Execution error diagnosis

The problem determination tools provided by IBM in the Application Delivery Foundation for z/OS can be used to determine the source of execution time problems in applications that contain ABO optimized modules. If the problem determination tools are not available, the listing transform produced by ABO can help diagnose the execution time problems.

If diagnoses determine that an ABO optimized module causes the execution time problem, revert to the original COBOL module and contact IBM service to report the problem. To learn about the information that needs to be collected to report an ABO problem to IBM, see ABO Mustgather page.

Changes in COBOL module size after optimization

The size of an optimized module is typically larger than the original module due to the types of optimizations ABO does to improve performance.

Here are some common reasons for the module size increase:

- Use of higher ARCH instructions that are usually 6 bytes versus 4 or 2 bytes in length for many lower ARCH instructions. For example:
  - Using Decimal Floating Point (DFP) for packed/zoned decimal arithmetic to improve performance
  - Replacing "base locator" pointers in the original module with more efficient but larger long displacement instructions
  - Using more than one move immediate instruction instead of one in memory move
- A number of optimizations in ABO generate more code but shorter path length and better performance. For example:
  - More efficient handling of numeric edited variables
  - Unrolling long move and compare operations instead of using shorter but much slower instructions
  - Conditionally correcting decimal precision for binary data
- The inlining of the behavior of various runtime library routines results in more code in the optimized module but much faster performance in many cases.

So for these and similar reasons the optimized modules produced by ABO are often larger than the original modules and require more on disk storage. However, the amount of memory used by the optimized program itself when running is the same as that used by the original module. A slightly higher EXCP count will sometimes be observed when running the optimized program but this is only due to the few extra I/O operations required to load the larger module.

Note that an optimized module will keep its size unchanged if the optimized code happens to be smaller than the original code.

Error message and abend code differences

The optimized modules generated by ABO are in almost every case functionally equivalent to the corresponding original modules. However, in some rare cases an ABO generated module will produce different Language Environment (LE) runtime messages or different CICS abend codes than the original module.

This can happen when division on large data items and other complex operations are inlined or optimized in the generated code by ABO for more efficient processing, instead of being handled by an LE library routine or inefficient machine instructions.

In a non-CICS application, an ABO generated module:

- may produce a fixed-point divide exception (CEE3209S) message in places where the original module produced a decimal-divide exception (CEE3211S) or IGZ0061S message.
- may produce a decimal-divide exception (CEE3211S) in places where the original module produced a IGZ0061S message

For reference the full LE runtime message text for these differing exceptions is given below.

```
CEE3211S The system detected a decimal-divide exception (System Completion Code=0CB).
IGZ0061S Division by zero occurred in program 'program-name' at displacement 'displacement'.
CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).
```

For a CICS application the abend code returned by "CICS ASSIGN ABCODE" can change from '1061' from the original module to 'ASRA' from the ABO generated module.

Application Delivery Foundation for z/OS

You can use Application Delivery Foundation for z/OS (ADFz) on ABO generated modules.

Find out more about Application Delivery Foundation for z/OS at https://www.ibm.com//products/app-delivery-foundation-for-zos.

The following Application Delivery Foundation for z/OS family of problem determination tools can be used on ABO generated COBOL modules:

- Developer for z/OS Enterprise Edition, which includes IBM Debug for z/OS
- Fault Analyzer for z/OS (FA)
- Application Performance Analyzer for z/OS (APA)

Creating a LANGX side file

For ABO optimized programs originally compiled with COBOL 4 and earlier releases, you need to produce a LANGX side file for each optimized program to use with the ADFz tools. IBM Debug for z/OS, FA, and APA use the LANGX side file to provide a better experience for users. For example, source-level debugging is provided with Debug for z/OS when a LANGX side file is provided. Without this LANGX side file, source-level debugging is not possible.

IPVLANGO is a tool provided with IBM Problem Determination Tools Common Component for z/OS V1.7, which is shared by the Application Delivery Foundation for z/OS tools. IPVLANGO combines the SYSDEBUG data set or compiler listing or the LANGX side file of the original compiled program along with the ABO listing transform to produce a new LANGX side file appropriate for the ABO generated module. Use the new LANGX file when you use DT, FA, or APA on the ABO generated module.

Producing DWARF diagnostic information

For modules compiled with COBOL 5 and later releases, the LANGX side file is not necessary. Instead, for modules that were compiled with TEST or with NOTEST(DWARF), ABO will itself produce the updated DWARF diagnostic information for the optimized module either within the module itself (NOSEPARATE) or within a separate side file (SEPARATE).

For TEST(NOSEPARATE), no further configuration of ABO is required. When ABO optimizes the module, it will also update the DWARF information within the optimized module to be applicable to that module.

For TEST(SEPARATE), the OPTDEBUG DD is specified to determine where the updated DWARF side files will be written.

Under DSNAME, this is the only requirement. The original COBOL DWARF will be read from the original side file using the dataset name that was stored into original COBOL module at compilation (but see below for an alternative location), Table 13 on page 90 shows how the updated DWARF side file name is determined. Specify the updated DWARF side file name as shown below:

```
//OPT EXEC PGM=BOZOPT,REGION=0M
...
//OPTDEBUG DD DSN=updated-dwarf-library,DISP=SHR
```

Under NODSNAME (or under DSNAME when the original COBOL DWARF is to be read from another location), the additional DD statement, SYSDEBUG DD, is specified for the location storing the original COBOL DWARF. The updated DWARF will be written as given in Table 14 on page 90. This emulates

behaviour under the sample user exit IGZIUXB/IGZIUXC. If the user has used their own custom user exit at COBOL runtime, then the user will need to adapt this exit to relocate the DWARF input file to a file location that will be accepted under Table 14 on page 90. Specify the original and updated DWARF side file names as shown below:

```
//OPT EXEC PGM=BOZOPT,REGION=0M
...
//OPTDEBUG DD DSN=updated-dwarf-library,DISP=SHR
//SYSDEBUG DD DSN=original-dwarf-library,DISP=SHR
```

Under the above options, the optimized module will appear to have been compiled with the TEST options of the original module. However, if for any reason ABO determines that it cannot generate valid DWARF the optimized module will still be generated, but it will appear to have been compiled with NOTEST(NODWARF). In this case one or more warning (return code 4) messages will issued and the updated DWARF information will not be generated. Note that under TEST(NOSEPARATE) the original DWARF will remain in the optimized module but will not be used by the ADFz tools.

Under TEST(SEPARATE) it is possible if required to bypass the generation of updated DWARF; if the OPTDEBUG DD is not specified, then ABO will behave as though it cannot generate valid DWARF, but in this case, no warning messages are shown, and normal optimization will proceed at return code 0.

Table 13. Determination of Updated Side File Name under TEST(SEPARATE(DSNAME))

Input DWARF Type	**Output DWARF (OPTDEBUG DD)**	**Output DWARF Type (OPTDEBUG DD)**
Sequential Dataset	DD:OPTDEBUG(*csect*[1]).	PDS or PDSE directory
PDS or PDSE member	DD:OPTDEBUG(*member*) where *member* is the member of the input DWARF.	
HFS file	DD:OPTDEBUG(*csect*[1]).	

Table 14. Determination of Updated Side File Name when SYSDEBUG is specified. [2]

Input DWARF Type of SYSDEBUG DD	**Valid or not**	**Input DWARF (SYSDEBUG DD)**	**Output DWARF (OPTDEBUG DD)**	**Output DWARF Type (OPTDEBUG DD)**
Sequential Dataset	Not valid	N/A	N/A	N/A
PDS or PDSE directory	Valid	DD:SYSDEBUG(*name*[3])	DD:OPTDEBUG(*name*[3])	PDS or PDSE directory
HFS	Not valid	N/A	N/A	N/A

Notes:

1. *csect* is the csect name. When *csect* is used as the member of a PDS or PDSE directory, then it is upper cased and truncated to 8 characters, as per the SYSPRINT DD or LIST options.
2. This emulates behaviour under the sample user exit IGZIUXB/IGZIUXC.
3. *name* is:
   - *csect* if *debugFileName* is empty;
   - the member name of *debugFileName* if *debugFileName* is a PDS or PDSE member;
   - invalid otherwise

   where *debugFileName* is empty under NODSNAME or it is the the dataset name that was stored into original module under DSNAME.

For logical record length and block size of datasets in OPTDEBUG and SYSDEBUG other than HFS files, see Logical record length and block size in the *Enterprise COBOL for z/OS Programming Guide*.

Appendix A. JCL sample

The following JCL sample is included in the IBM Automatic Binary Optimizer for z/OS installation package.

```
//BOZJCLE JOB <job parameters>
//*********************************************************************
//* Job Name: BOZJCLE                                                 *
//*                                                                   *
//*    Licensed Materials - Property of IBM                           *
//*    5697-AB2, 5655-EC6                                             *
//*    Copyright IBM Corp. 2015, 2022                                 *
//* This file is part of product 5655-EC6, 5655-TY6, 5697-V61         *
//* 5697-AB2, 5697-TR2                                                *
//*                                                                   *
//*    US government users restricted rights                          *
//*    use, duplication or disclosure restricted                      *
//*    by GSA ADP schedule contract with IBM Corp.                    *
//*                                                                   *
//*********************************************************************
//*  Description: This JCL will optimize a COBOL program using        *
//*               IBM Automatic Binary Optimizer for z/OS             *
//*               V02.02.00                                           *
//*********************************************************************
// SET BOZJOBID = 'unique-identifier'
//*
//OPT      EXEC PGM=BOZOPT,REGION=0M
//STEPLIB  DD DSN=hlqboz.SBOZMOD1,DISP=SHR
//OPTLOG   DD DSN=hlq.BOZOUT.OPTLOG(&BOZJOBID),DISP=SHR
//OPTERR   DD DSN=hlq.BOZOUT.OPTERR(&BOZJOBID),DISP=SHR
//CEEDUMP  DD DSN=hlq.BOZOUT.CEEDUMP(&BOZJOBID),DISP=SHR
//SYSPRINT DD DSN=hlq.BOZOUT.LISTING,DISP=SHR
//SYSBIN   DD DSN=input-load-library,DISP=SHR
//SYSBOUT  DD DSN=output-load-library,DISP=SHR
//SYSIN    DD *
  ARCH=arch-number
  BOPT IN=DD:SYSBIN(member-name) OUT=DD:SYSBOUT(member-name)
```

Notes:

1. The SYSLIB DD card is not required when using ABO 2.2. ABO will dynamically allocate the SYSLIB to match the ABO load library found in the STEPLIB, JOBLIB or LNKLST.
2. The STEPLIB card in the above JCL is only optional and can be omitted when the ABO load library is in the LNKLST of the same system where the job is going to be submitted.

In the JCL example, hlqboz.SBOZMOD1 is the installation location chosen for the optimizer.

This example requires the following data sets to be allocated beforehand:

- hlq.BOZOUT.OPTLOG
- hlq.BOZOUT.OPTERR
- hlq.BOZOUT.LISTING
- hlq.BOZOUT.CEEDUMP

You can allocate these data sets with the recommended parameters in the following table:

Table 15. Recommended allocation parameters

Data sets	Recommended allocation parameters
hlq.BOZOUT.OPTLOG hlq.BOZOUT.OPTERR	Space units: CYLS Primary quantity: 50 Secondary quantity: 50 Directory blocks: 10 Record format: VB[1] Record length: 512 Block size: 27998 Data set name type: Library

Table 15. Recommended allocation parameters (continued)

Data sets	Recommended allocation parameters
hlq.BOZOUT.LISTING	Space units: CYLS Primary quantity: 50 Secondary quantity: 50 Directory blocks: 10 Record format: FB Record length: 133[2] Block size: 0[3] Data set name type: Library
hlq.BOZOUT.CEEDUMP	Space units: CYLS Primary quantity: 10 Secondary quantity: 10 Directory blocks: 10 Record format: FB Record length: 133 Block size: 27930 Data set name type: Library

Notes:

1. hlq.BOZOUT.OPTLOG and hlq.BOZOUT.OPTERR **must** have a record format of VB to be opened successfully.
2. hlq.BOZOUT.LISTING has a recommended record length (LRECL) of 133 so it can contain all the ABO listing data. If the record length is less then any excess characters will be truncated from the listing.
3. A block size of 0 will use a system determined block size (BLKSIZE). You may specify your own block size but it must be a multiple 'n' of the record length (LRECL) : n x 133.

&BOZJOBID is a unique identifier for this job chosen by the user. It is used as the member name in each of the hlq.BOZOUT.OPTLOG, hlq.BOZOUT.OPTERR, and hlq.BOZOUT.CEEDUMP data sets. &BOZJOBID must be a valid member name.

This JCL sample shows a definition of the SYSIN DD for optimizing a single module. For more examples, see “JCL examples” on page 36 and “Optimization and deployment usage scenarios ” on page 82. For descriptions of the ddnames used in the example, see “Required DD statements” on page 17.

Appendix B. Return codes

IBM Automatic Binary Optimizer for z/OS emits messages to provide information, provide possible warnings, or to report errors. Each message has a "Message return code" that is documented in Appendix C, “Messages,” on page 95. On termination, ABO passes a return code value to z/OS that is the maximum of the "Message return code" values of all the messages that were emitted. If no messages were emitted, then a return code of 0 is returned to z/OS.

Table 16. IBM Automatic Binary Optimizer for z/OS return codes

Return code (decimal)	**Description**
0	Successful completion of all processing. One or more informational messages may have been emitted.
4	Successful completion but an unusual condition was detected. One or more warning messages have been emitted.
12	An error was detected during the processing of a BOPT or IEFOPZ directive or global option. One or more messages have been emitted. • If the error occurs during syntax processing of a line of input, the rest of the line is rejected and ABO proceeds to process the next line of input. • If the error occurs while processing a BOPT or IEFOPZ directive, ABO proceeds to process the next applicable module of the BOPT or IEFOPZ directive. If there are no further input modules to process for the BOPT or IEFOPZ directive, processing of the directive is terminated and ABO proceeds to the next line of input to process the next directive or terminates if there are no more lines of input.
16	An unrecoverable error was detected. One or more messages are emitted and ABO immediately terminates processing.

Appendix C. Messages

The "ABO messages" on page 95 described in this section are written to the OPTLOG DD. In some exceptional cases, it may not be possible to write to the OPTLOG DD in which case the message is written to write to the OPTERR DD otherwise to the JOBLOG. Each message, listed below, has a "Message return code" that is used to determine the return code returned to z/OS as described in Appendix B, "Return codes," on page 93.

Each ABO message in this section has the form BOZnnnnX where BOZ indicates that the message is an ABO message, nnnn is the message number, X is the severity indicator.

The "RTI Profiler messages" on page 115 described in this section are written to the JOBLOG.

Each RTI message in this section has the form BOZRnnnX where BOZR indicates that the message is an RTI message, nnn is the message number, X is the severity indicator.

The "ABO Assistant messages" on page 118 described in this section are written to the JOBLOG.

Each ABO Assistant message in this section has the form BOZAnnnX where BOZA indicates that the message is an ABO Assistant message, nnn is the message number, X is the severity indicator

Severity indicators can be any of the following: I, W, E, S, or U.

I
Informational message (RC=0)

W
Warning message (RC=4)

E
Error message (RC=8)

S
Severe error message (RC=12)

U
Unrecoverable error message (RC=16)

ABO messages

BOZ1031S An error occurred while attempting to open "&1".

Explanation:
The optimizer was unable to open the file specified by &1.

System action:
If the open failure was associated with one of the required optimizer DDs, such as the SYSIN DD, the optimizer immediately terminates with a return code of 16. Otherwise, if the file is specified on a line in the SYSIN input file (in a global option or BOPT or IEFOPZ directive), processing of the line is terminated and the optimizer proceeds to process the next line of the SYSIN input file.

User response:
Ensure that the file name is correct, and that the file has been allocated and with an appropriate record format and with an appropriate record length.

Message return code
16 when attempting to open a mandatory DD, otherwise 12.

BOZ1145U Insufficient memory in the compiler to continue compilation.

Explanation

The optimizer was unable to continue due to memory being low.

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response:
Consider using the JCL MEMLIMIT or JCL REGION parameters to increase the memory used by the optimizer. For more information, see the z/OS MVS Initialization and Tuning Reference and the z/OS MVS Initialization and Tuning Guide.

Message return code

16

BOZ1147U **Internal error detected in “CDA" phase at &1.**

Explanation

The optimizer encountered an error when trying to produce updated DWARF debug information.

System action

The optimization is terminated with a failure.

User response

Contact IBM support.

Message return code

16

BOZ1148S **An I/O on "&1" was detected in"CDA" phase at &2.**

Explanation

The optimizer encountered an error when reading or writing file "&1" while trying to produce updated DWARF debug information.

System action

The optimization is terminated with a failure.

User response

Contact IBM support.

Message return code

12

BOZ1149S **An error opening "&1" was detected in "CDA" phase at &2.**

Explanation

The optimizer encountered an error when opening file "&1" while trying to produce updated DWARF debug information.

System action

The optimization is terminated with a failure.

User response

Contact IBM support.

Message return code

12

BOZ1150S **An error reading from "&1" was detected in "CDA" phase at &2.**

Explanation

The optimizer encountered an error when reading from file "&1" while trying to produce updated DWARF debug information.

System action

The optimization is terminated with a failure.

User response

Contact IBM support.

Message return code

12

BOZ1151S **An error writing to "&1" was detected in "CDA" phase at &2.**

Explanation

The optimizer encountered an error when writing to file "&1" while trying to produce updated DWARF debug information.

System action

The optimization is terminated with a failure.

User response

Contact IBM support.

Message return code

12

BOZ1152S **An error seeking within "&1" was detected in "CDA" phase at &2.**

Explanation

The optimizer encountered an error when reading from file "&1" while trying to produce updated DWARF debug information.

System action

The optimization is terminated with a failure.

User response

Contact IBM support.

Message return code

12

BOZ1400S Directive is missing "&1" specifier.

Explanation

The optimizer encountered a BOPT or IEFOPZ directive that requires an &1 specifier, but the specifier was missing.

System action:
The optimizer discards the directive and proceeds to process the next line in the SYSIN input file.

User response:
Correct the directive by adding an appropriate &1 specifier.

Message return code

12

BOZ1401S "&1" directive must be specified at start of line.

Explanation

An option preceded the &1 directive on a line of the SYSIN input file, or the line is missing the &1 directive.

System action:
The optimizer discards the directive and proceeds to process the next line in the SYSIN input file.

User response:
Fix the line by specifying &1 directive at the start of the line.

Message return code

12

BOZ1402S Invalid specifier in "&1".

Explanation

The &1 option of BOPT or IEFOPZ directive contained an invalid specifier. For example, "H" is an invalid specifier in the option: "SCAN=H".

System action:
The optimizer discards the directive with the invalid specifier and proceeds to process the next line in the SYSIN input file.

User response:
Change the specifier in the option to one that is valid.

Message return code

12

BOZ1403S Invalid option "&1".

Explanation

While processing the SYSIN input file, &1 was encountered such that &1 is not an optimizer directive and &1 is not a valid option.

System action:
The optimizer discards the line with the invalid option and proceeds to process the next line in the SYSIN file.

User response:
Correct the line of SYSIN by using a properly spelled directive or option.

Message return code

12

BOZ1404S "&1" can only be specified on "&2" directive.

Explanation

The &1 option was specified on a directive but it was not the &2 directive. For example, SEL_ARCH cannot be specified on the BOPT directive as SEL_ARCH is only applicable to the IEFOPZ directive.

System action:
The optimizer discards the line with the invalid option and proceeds to process the next line in the SYSIN file.

User response:
Fix the line with &1, by specifying a proper option or proper directive.

Message return code

12

BOZ1405S "&1" not allowed on "&2" directive.

Explanation

The &2 directive contained an option, &1, that is not applicable to the &2 directive. For example, the "IN" option is not applicable and cannot be specified on the IEFOPZ directive.

System action:
The optimizer discards the line with the invalid option and proceeds to process the next line in the SYSIN file.

User response:
Fix the line by specifying a proper option that applies to the &2 directive.

Message return code

12

BOZ1406S Wildcards not supported in member specifier of "&1".

Explanation

A line of SYSIN was encountered with an "IN" option with a member specifier (&1) that included wildcards, and an "OUT" option that included a dataset member specifier. When wildcards are used in an "IN" option, the "OUT" option must not include a member specifier.

System action:
The optimizer discards the line with the invalid option and proceeds to process the next line in the SYSIN file.

User response:
Change the "IN" option to not specify wildcards or remove the member specifier from the "OUT" option.

Message return code

12

BOZ1407S Output specifier "&1" invalid when using wildcards on input.

Explanation

The optimizer detected a BOPT directive with a wildcard specifier on the "IN" option and the "OUT" option specified a USS path of &1. When member wildcards are used on the "IN" option, the "OUT" option must specify a dataset and not a USS path.

System action:
The optimizer discards the BOPT option with the invalid "OUT" option and proceeds to process the next line in the SYSIN file.

User response:
Change the "IN" option to not specify wildcards or change the "OUT" option to specify a dataset.

Message return code

12

BOZ1408S Module specifier "&1" is an existing directory.

Explanation

The optimizer detected an HFS directory, &1, specified as the module location on the "IN" or "OUT" option of the BOPT directive. In HFS, a module is an ordinary file and not a directory.

System action:
The optimizer discards the directive and proceeds to process the next line in the SYSIN file.

User response:
Change the path specifier on the "IN" or "OUT" option to be an ordinary file and not a directory.

Message return code

12

BOZ1409S Output specifier "&1" is not an existing PDS(E).

Explanation

The optimizer encountered either:

- &1 as a NEW dataset from an IEFOPZ configuration
- &1 as a DD name or dataset name on a file specifier in the SYSIN file where the file specifier included a member name
- &1 as a dataset specifier for the LOG option

but, the dataset associated with &1 did not exist or the dataset was a sequential dataset and not a PDS(E).

System action:
In the IEFOPZ case, the optimizer ignores the NEW dataset and continues on processing the IEFOPZ configuration. Otherwise, the optimizer discards the directive and proceeds to process the next line in the SYSIN file.

User response:
Change the dataset location to an existing PDS(E) or allocate the PDS(E) prior to running the optimizer.

Message return code

12

BOZ1410I Output module "&1" cannot be replaced as REPLACE=N is in effect.

Explanation

When the REPLACE=Y option is specified, the optimizer issues this informational message when it detects that the output module (&1) of the same name already exists.

System action:
The optimizer bypasses optimizing the input module and proceeds to process the next module or next directive.

User response:
No action is required by the user.

Message return code

0

BOZ1411S Error getting member list from dataset specifier "&1".

Explanation

The optimizer was processing either:

- a BOPT directive where a PDS(E) (&1) was specified on the "IN" option (that included member wildcards) and the PDS(E) had no members
- a IEFOPZ directive, and an OLD dataset (&1) in the IEFOPZ configuration was found to have no members

System action:
In the case of a BOPT directive, the optimizer discards the directive and proceeds to process the next line of the SYSIN file. In the case of the IEFOPZ directive, the optimizer ignores the OLD dataset and proceeds to process the rest of the IEFOPZ configuration.

User response:
Check that the proper dataset was specified on the BOPT directive or that the proper dataset was specified in the IEFOPZ configuration.

Message return code

12

BOZ1412S **IEFOPZ is not available on this system.**

Explanation

The optimizer was processing an IEFOPZ directive on a z/OS system that did not have the IEFOPZ feature

System action:
The optimizer discards the IEFOPOZ directive and proceeds to process the next line in the SYSIN file.

User response:
The IEFOPZ facility is only available on z/OS V2R2 and above. If the optimizer is run on a z/OS system prior to V2R2, change SYSIN to not specify the IEFOPZ directive. If the optimizer is run on z/OS V2R2 or higher, have your system programmer install the appropriate PTFs required for the IEFOPZ feature.

Message return code

12

BOZ1413S **Problem with IEFOPZQ system service (return code="&1", reason code="&2"): &3.**

Explanation

The optimizer encountered a problem reading an IEFOPZ configuration while processing an IEFOPZ directive. &1 specifies the error return code and &2 specifies the error reason code of the IEFOPZQ system service that is used to read the configuration. &3 gives a short description of the reason code.

System action:
The optimizer discards the IEFOPZ directive and proceeds to process the next line in the SYSIN file.

User response:
Provide this error message to your system programmer to see if the error is valid If there are no issues with IEFOPZ usage, consult IBM service providing this optimizer message and any other IEFOPZ configuration information.

Message return code

12

BOZ1414S **Input specifier "&1" is not an existing PDS(E).**

Explanation

The optimizer encountered either:

- &1 as an OLD dataset from an IEFOPZ configuration
- &1 as a DD name or dataset name on a file specifier in the SYSIN file where the file specifier included a member name

but, the dataset associated with &1 did not exist or the dataset was a sequential dataset and not a PDS(E).

System action:
In the IEFOPZ case, the optimizer ignores the OLD dataset and continues on processing the IEFOPZ configuration. Otherwise, the optimizer discards the directive and proceeds to process the next line in the SYSIN file.

User response:
Change the dataset name to an existing PDS(E) or allocate the PDS(E).

Message return code

12

BOZ1415S **No DD definition is supplied for "&1".**

Explanation

The optimizer could not find a DD definition for a mandatory optimizer DD (&1), or no DD definition was specified for a DD (&1) used in the SYSIN file.

System action:
If the DD is for a mandatory DD of the optimizer, the optimizer immediately terminates with a return code of 16. Otherwise, the optimizer discards the line of the SYSIN file that included the DD definition and processes the next line of SYSIN.

User response:
Provide a DD definition for the DD in error.

Message return code

16, if &1 is a mandatory DD, otherwise 12

BOZ1416S **A member name is not specified for PDS(E) specifier "&1".**

Explanation

The optimizer encountered an "IN" or "OUT" option that specified a PDS(E) (&1) and requires a member name, but no member was included on the option.

System action:
The optimizer discards the directive with the invalid "IN" or "OUT" option and proceeds to process the next line in the SYSIN file.

User response:
Change the "IN" or "OUT" option to include a dataset member.

Message return code

12

BOZ1417S **File "&1" does not exist.**

Explanation

Input file &1 could not be located. Two common reasons this can happen are:

1. A member of an existing input module PDS(E) does not exist
2. An invalid HFS path was specified

System action:
The optimizer ignores processing the directive (or input module) that used the invalid file specification and proceeds to process the next input module or next directive

User response:
Correct the problem by specifying an existing dataset member or correct the path specification to point to an existing HFS file.

Message return code

12

BOZ1418S **Invalid file specification "&1".**

Explanation

The specification of the file &1 is incorrect. Examples of incorrect specifications include:

1. A member name is specified twice:
   - once, in the definition of a DD
   - second, in an optimizer option or directive that included the DD definition
2. A HFS path is specified, but a directory in the path is non-existent or the path is not accessible.
3. The length of a DD name or dataset name is too long.

System action:
The optimizer ignores processing the directive (or input module) that used the invalid file specification and proceeds to process the next input module or next directive

User response:
Specify a proper format for the file specification (&1).

Message return code

12

BOZ1419S **Output of load module(s) to "&1" is not supported when input has program object format.**

Explanation

An input module has a newer program object format but the optimized module (&1) is targeted to the older load module format. This happens when the input module is a member of a PDSE or a file in a HFS path, but the optimized module is targeted to a member of a PDS or is targeted to a sequential dataset.

System action:
The optimizer terminates processing the input module and proceeds to process the next input module or next directive.

User response:
Correct the output location (&1) of the optimized module to be a member of a PDSE or to a HFS path.

Message return code

12

BOZ1420S **Path "&1" must be absolute and begin with a "/".**

Explanation

The specification (&1) of an input or output file is to an HFS file, but a full path specification is not provided for &1. A full or absolute file specification must begin with a "/" character. This error can happen, for example, when the optimizer processes an HFS specification of an a input module, output module or listing transform.

System action:
The optimizer bypasses optimizing a module when an invalid path is specified and proceeds to process the next input module or next directive.

User response:
Correct the specification of the path (&1) to be absolute.

Message return code

12

BOZ1421S **Binder API "&1" failed: return code=&2 reason code=&3.**

Explanation

While processing a module using a binder API (&1), the binder API returned with an unexpected return code (&2) and reason code (&3).

System action:
In most cases, the optimizer discontinues processing the input module and proceeds to process the next input module or next directive. In some cases (for example, return code=4, reason code = 0x83000526), the binder is able to recover from the problem (in this case unexpected input) and the optimization of the input module proceeds.

User response

Examine binder documentation for information on the reason code. The reason code information can help determine the cause of the problem. For example, the reason code may indicate that the input file for optimization is not a proper load module or program object file. In this case, correct the JCL or SYSIN file to specify a proper input module. For information about binder API return codes and reason codes, see z/OS MVS Program Management: Advanced Facilities.

This message is normally proceeded by a BOZ4116 binder message that might provide additional information that helps in your response.

Message return code

12 when the optimizer discontinues processing, otherwise 4.

BOZ1422S **Module cannot be processed as it is not marked executable.**

Explanation

The optimizer encountered an input module that was not marked executable. The optimizer requires the module to be marked executable in order for the optimization process to succeed.

System action:
The optimizer discontinues processing the input module and proceeds to process the next input module or next directive.

User response:
If this problem is expected for the module, then either ignore the message or change the optimizer directives to exclude the module. Otherwise, if the problem is unexpected, correct the bind steps that produced the input module so that the module resulting from the bind is marked executable.

Message return code

12

BOZ1423S **Module cannot be processed as it was linked EDIT=NO or otherwise cannot be reprocessed.**

Explanation

The optimizer encountered an input module that cannot be edited. The most common case where this can happen is when the bind step used to produce the module included the EDIT=NO binder option. Modules that cannot be edited are missing important information required by the optimizer.

System action:
The optimizer discontinues processing the input module and proceeds to process the next input module or next directive.

User response:
If this problem is expected for the module, then either ignore the message or change the optimizer directives to exclude the module. Otherwise, if the problem is unexpected, remove the EDIT=NO option from the bind steps that produced the input module.

Message return code

12

BOZ1424S **Module cannot be appropriately processed as program is SIGNed.**

Explanation

The optimizer encountered an input module that is marked SIGNed.

System action:
The optimizer does not supported SIGNed modules and discontinues processing the input module and proceeds to process the next input module or next directive.

User response:
If this problem is expected for the module, then either ignore the message or change the optimizer directives to exclude the module. Otherwise, if the problem is unexpected, correct the bind steps used to produce the module so that the module is not marked SIGNed.

Message return code

12

BOZ1426S **Link library "SYSLIB" not specified or does not specify a PDS(E).**

Explanation

The DD name card allocating link library SYSLIB is missing or doesn't specify a PDS(E) data set in the optimizer job step JCL.

System action:
The optimizer terminates optimizing the input module and proceeds to process the next module.

User response:
Specify SYSLIB correctly in your optimizer job step JCL.

Message return code

12

BOZ1428U **Insufficient memory encountered during binder API "&1": return code=&2 reason code=&3. Terminating optimizer.**

Explanation

While processing a module using a binder API (&1), the binder was unable to proceed due to memory being low. The binder produces the return code (&2) and reason code (&3) indicating the memory problem.

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response

Consider using the JCL MEMLIMIT or JCL REGION parameters to increase the memory used during the optimization process. For more information, see the z/OS MVS Initialization and Tuning Reference and the z/OS MVS Initialization and Tuning Guide.

This message is normally proceeded by a BOZ4116 binder message that might provide additional information that helps in your response.

Message return code

16

BOZ1429U **"&1" I/O error encountered during binder API "&2": return code=&3 reason code=&4. Terminating optimizer.**

Explanation

While processing a module using a binder API (&2), the binder detected an I/O error of type &1. The binder API provided the return code (&3) and reason code (&4).

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response

The type (&1) of I/O problem and the reason code (&4) can help direct the steps of how to diagnose and fix the problem. See binder documentation for an explanation of the reason code. Now, an example is a "WRITE" (&1) error of the optimized module because the output PDS(E) or file system is full. The binder API information (&2) or reason code (&4) can help confirm, or lead to, the cause of the "WRITE" problem. Note that increasing the size of the PDS(E) (or file system) could fix the "WRITE" problem. For information about binder API return codes and reason codes, see z/OS MVS Program Management: Advanced Facilities.

This message is normally proceeded by a BOZ4116 binder message that might provide additional information that helps in your response.

Message return code

16

BOZ1430U **Unrecoverable "&1" error encountered during binder API "&2": return code=&3 reason code=&4. Terminating optimizer.**

Explanation

While processing a module using a binder API (&2), the binder detected an error of type &1. The binder API provided the return code (&3) and reason code (&4).

System action:
The binder immediately terminates with a return code of 16.

User response

The type (&1) of problem and the reason code (&4) can help direct the steps of how to diagnose and fix the problem. If you are unable to diagnose the problem, consult IBM service for assistance. For information about binder API return codes and reason codes, see z/OS MVS Program Management: Advanced Facilities.

This message is normally proceeded by a BOZ4116 binder message that might provide additional information that helps in your response.

Message return code

16

BOZ1431S **Input module with unsupported feature (&1) encountered during binder API "&2": return code=&3 reason code=&4. Module bypassed.**

Explanation

While processing a module using a binder API (&2), the binder detected that the module cannot be optimized due to the module contained a feature &1 that is not supported. The binder API provided the return code (&3) and reason code (&4).

One example of an unsupported feature is when the input is an object module (as opposed to the input being a load module or program object). Another example of this problem, is when the input module is not fully bound and contains "UNRESOLVED" references.

System action:

User response

Since the input module cannot be supported, the choices are:

- ignore the message
- change optimizer input to avoid optimizing the module
- fix the problem. For example, in the case of a module with "UNRESOLVED" references, change the build steps used to produce the module so that the module is fully bound

For information about binder API return codes and reason codes, see z/OS MVS Program Management: Advanced Facilities.

This message is normally proceeded by a BOZ4116 binder message that might provide additional information that helps in your response.

Message return code

12

BOZ1432S **Output module size exceeded module format limitations and cannot be saved.**

Explanation

The optimizer attempted to write the optimized module but ran into output format restrictions. A load module, saved into a PDS member (or sequential dataset), has the most restrictive format. Far less common is encountering a format limitation with a program objects (written to PDSE or HFS path).

System action:

The optimizer terminates optimizing the input module and proceeds to process the next module or next directive.

User response:

If the output module is to be saved into a PDS member or sequential dataset, consider changing the output location to be a member of a PDSE. Otherwise, consider splitting the program into multiple modules.

Message return code

12

BOZ1433S **RTIBIND is not supported for 64-bit module "&1".**

Explanation

The optimizer detected that the request to rebind with RTI profiling modules was issued for the AMODE 64 compiled module (&1). AMODE 64 modules are those produced, for example, from using the LP(64) or LP64 options in IBM compilers. RTIBIND is not supported for AMODE 64 compiled modules.

System action:

The optimizer stops processing and does not rebind the input module with the RTI profiling modules. The optimizer proceeds to process the next module.

User response:

Avoid using RTIBIND for AMODE 64 compiled modules.

Message return code

12

BOZ1434W **At least one character in the member name of "&1" is not valid.**

Explanation

The optimizer attempted to read DWARF debug information from the module or a separate debug file, but the member name did not conform to standard rules defined in z/OS data set and UNIX file naming conventions and z/OS data set naming rules. These rules apply even if the member name is accepted under other rules, such as those for compiling or linking the module. The DWARF debug information could not be read.

System action

The optimization of the CSECT will complete as normal, but updated DWARF debug information is not generated, and it will appear to subsequent tooling that the modules was compiled under NOTEST(NODWARF), and therefore may not behave as expected.

User response

Use member names that conform to established naming rules.

Message return code

4

BOZ1436S Invalid ARCH specification : &1

Explanation

An invalid or unsupported architecture specification (&1) was detected in one of the following cases:

1. In an ARCH option or SEL_ARCH option when processing the SYSIN file
2. When processing a NEW dataset in an IEFOPZ configuration

System action:
If the invalid specification was detected on a line of the SYSIN file, the optimizer discards the line with the invalid option and proceeds to process the next line in the SYSIN file. If the invalid specification was detected processing a NEW dataset of an IEFOPZ configuration, the optimizer ignores the NEW dataset and proceeds to process the remainder of the IEFOPZ configuration

User response:
Correct the SYSIN file or IEFOPZ configuration by specifying an ARCH level supported by the optimizer.

Message return code

12

BOZ1437S No BOPT or IEFOPZ directive found

Explanation

The optimizer can neither find a BOPT nor an IEFOPZ directive.

System action:
The optimizer terminates execution and returns to the operating system with a return code of 12.

User response:
Check that your JCL includes at least one BOPT or IEFOPZ directive.

Message return code

12

BOZ1438U dynfree dyn failed: rc=&1 for DD &2

Explanation

The optimizer detected an error attempting to free an internal input DD name (&2) that the optimizer was using as part of the optimization process.

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response:
This problem could have been caused by an earlier problem. Correct any problems reported in the log file and retry the optimization process. If the problem persists, consult IBM service for assistance.

Message return code

16

BOZ1439U dynfree saveDyn failed: rc=&1 for DD &2

Explanation

The optimizer detected an error (the dynfree service returned &1) attempting to free an internal output DD name (&2) that the optimizer was using as part of the optimization process.

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response:
This problem could have been caused by an earlier problem. Correct any problems reported in the log file and retry the optimization process. If the problem persists, consult IBM service for assistance.

Message return code

16

BOZ1446U An I/O error occurred while writing &1

Explanation

The optimizer detected an I/O error when writing to &1, where &1 could be either 'the Listing transform' or 'the Log file'.

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response:
Check whether the PDS(E) or file systems is full and allocate a larger file for the PDS(E) or increase the size of the file system. Also check whether the dataset

was allocated with a proper record format and record length.

Message return code

16

BOZ1447U An Unexpected I/O error occurred

Explanation

The optimizer detected an I/O error during execution.

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response:
Check the definitions of the output files (you should be able to exclude the output modules) to ensure a proper record length and record format is used and check whether the files are full.

Message return code

16

BOZ1448U Terminating due to unhandled exception &1

Explanation:
The optimizer was unable to continue because an unexpected condition was encountered during processing.

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response:
Unexpected problems could happen due to an earlier problem. Correct any problems reported in the log file and retry the optimization process. If the problem persists, consult IBM service for assistance.

Message return code

16

BOZ1449U Unhandled out of memory exception

Explanation

The optimizer was unable to continue due to memory being low.

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response:
Consider using the JCL MEMLIMIT or JCL REGION parameters to increase the memory used by the optimizer. For more information, see thez/OS MVS Initialization and Tuning Reference and the z/OS MVS Initialization and Tuning Guide.

Message return code

16

BOZ1450U Assertion failure, check logs for traceback

Explanation

The optimizer was unable to continue as an unexpected condition was encountered during processing.

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response:
Unexpected problems could happen due to an earlier problem. Correct any problems reported in the log file and retry the optimization process. If the problem persists, consult IBM service for assistance.

Message return code

16

BOZ1451S dynalloc(): failed for DSN &1 with DD &2, errcode &3, info code &4

Explanation

The optimizer encountered an error allocating an internal DD (&2) for dataset (&1). &3 is the error code returned by the MVS dynamic allocation functions. &4 is the information code returned by the MVS dynamic allocation functions.

System action:
The optimizer terminates optimizing the input module and proceeds to process the next module or next directive.

User response:
Check that the dataset (&2) exists and is accessible and check that your JCL does NOT include a definition for the same DD (&2). Also, unexpected problems could happen due to an earlier problem. Correct any problems reported in the log file and retry the optimization process. If the problem persists, consult IBM service for assistance.

Message return code

12

BOZ1452S **dynalloc(): failed for path &1 with DD &2, errcode &3, info code &4**

Explanation

&3 is the error code returned by the MVS dynamic allocation functions. &4 is the information code returned by the MVS dynamic allocation functions.

System action:
The optimizer terminates optimizing the input module and proceeds to process the next module or next directive.

User response:
Check that the path is accessible and can be written to and check that your JCL does NOT include a definition for the same DD (&2). Also, unexpected problems could happen due to an earlier problem. Correct any problems reported in the log file and retry the optimization process. If the problem persists, consult IBM service for assistance.

Message return code

12

BOZ1453U **dynalloc(): failed for DUMMY DD &1 errcode &2, info code &3**

Explanation

The optimizer encountered an error allocating a mandatory DUMMY DD (&1) that is required for the optimization process. &2 is the error code returned by the MVS dynamic allocation functions. &3 is the information code returned by the MVS dynamic allocation functions.

System action:
The optimizer immediately terminates execution and returns to the operating system with a return code of 16.

User response:
Check that your JCL does NOT include a definition for the same DD (&1). Also, unexpected problems could happen due to an earlier problem. Correct any problems reported in the log file and retry the optimization process. If the problem persists, consult IBM service for assistance.

Message return code

16

BOZ1455W **Unsupported feature "&1" found**

Explanation

This message is issued in one of the following situations:

1. When ABO encounters a COBOL CSECT (i.e. compiled COBOL program) built by a compiler not eligible for use with ABO or the CSECT contains a COBOL language feature not supported by ABO.
2. When ABO encounters a CSECT that is too complex to safely optimize.

In the first situation, ABO detected a feature "&1" that it does not support in the CSECT being processed. See COBOL module requirements for the compilers eligible for use with ABO and the COBOL language features not supported by ABO.

In the second situation, ABO has determined that the CSECT is too complex to be safely optimized so it has been skipped. ABO will only optimize a CSECT if it can ensure the optimized program will execute with the same logic as the original compiled program. In cases where the CSECT is so complex that ABO cannot guarantee this, ABO stops the optimization process and skips this CSECT. Any other eligible CSECTs in the module will still be processed.

Note that this message is issued for informational purposes only and does not indicate a functional issue with ABO.

System action:
ABO bypasses optimization of the CSECT and proceeds to process the next CSECT.

User response:
If you see message BOZ1455 issued for a particular unsupported feature while optimizing a large number of your modules, you may open an RFE to indicate that the lack of this feature is inhibiting your ability to use ABO effectively.

Message return code

4

BOZ1456S **"&1" cannot be both optimizer input and optimizer output.**

Explanation

The optimizer does not allow a dataset or file to be used as input to the optimizer as well as output from the optimizer. For example, an optimized module cannot be written to a PDS(E) if that PDS(E) is also a source of input modules. This message is emitted when &1 is used as both a location of input to the optimizer and a location of output from the optimizer.

System action:
The optimizer terminates optimizing the input module and proceeds to process the next module.

User response:
Correct your JCL or SYSIN file such that the output datasets are separate from input datasets.

Message return code

12

BOZ1457S Invalid filter expression "&1".

Explanation

The optimizer cannot process the *mem_expr* parameter of BOPT optimizer directive or the *expr* parameter of the CSECT optimizer option. The expression contains invalid syntax or wildcards and cannot be processed as written.

System action:
The optimizer terminates optimizing the current directive and proceeds to process the next directive.

User response:
Correct the expression. See the description of the *mem_expr* parameter of "BOPT" on page 19 and the *expr* parameter of the CSECT optimizer option.

Message return code

12

BOZ1458W Listing file "&1" has LRECL below recommended &2. Truncation may occur.

Explanation

The listing file &1 has a lower maximum record length than the recommended amount &2 as specified in the recommended allocation parameters. Note that for V record formats, the maximum record length of a file is 4 less than the LRECL.

System action

The optimizer writes the listing transform to the file &1, but the lines will be truncated to the maximum record length of file &1.

User response

Update the allocation parameters for file &1 increasing the LRECL to at least the recommended amount &2. Note that for V type record formats, the amount should be 4 more than &2.

Message return code

4

BOZ1487W File "&1" does not exist, no debug file is generated.

Explanation

The optimizer attempted to read DWARF debug information from a separate debug file, but the file did not exist.

System action

The optimization of the CSECT will complete as normal, but updated DWARF debug information is not generated, and it will appear to subsequent tooling that the modules was compiled under NOTEST(NODWARF), and therefore may not behave as expected.

User response

Find the separate debug file for the compilation and make it available at the file location that the optimizer is expecting.

Message return code

4

BOZ1488W DD name "SYSDEBUG" not specified or does not specify a PDS(E), no debug file is generated.

Explanation

The optimizer could not read DWARF debug information from a separate debug file, because the SYSDEBUG DD was not specified or did not specify a PDS or PDSE directory, or concatenation.

A SYSDEBUG DD that refers to a PDS or PDSE directory, or concatenation, is required for modules compiled with TEST(SEPARATE(NODSNAME)).

System action

The optimization of the CSECT will complete as normal, but updated DWARF debug information is not generated, and it will appear to subsequent tooling that the modules was compiled under NOTEST(NODWARF), and therefore may not behave as expected.

User response

Find the separate debug file for the compilation and make it available at the file location that the optimizer is expecting.

Message return code

4

BOZ1489W **Member "&1" not found under SYSDEBUG concatenation, no debug file is generated.**

Explanation

The optimizer attempted to read DWARF debug information from a separate debug file, where the file is a member of the PDS or PDSE directory, or concatenation specified by the SYSDEBUG DD, but the member did not exist.

System action

The optimization of the CSECT will complete as normal, but updated DWARF debug information is not generated, and it will appear to subsequent tooling that the modules was compiled under NOTEST(NODWARF), and therefore may not behave as expected.

User response

Find the separate debug file for the compilation and make it available at the file location that the optimizer is expecting.

Message return code

4

BOZ1490W **Warning: AMODE/RMODE conflicts encountered during binder API "&1": return code=&2 reason code=&3. Operation performed and processing continues.**

Explanation

During the optimization process, the binder detected a conflict with the AMODE and RMODE settings. This problem was detected by the binder API (&1) for which the binder issued a return code (&2) and reason code (&3). You can use binder reason code documentation along with the reason code (&3) to determine the precise nature of the conflict. Normally, the conflict is already present in the input module and not introduced by the optimization process.

System action:
The binder issues this BOZ1490 warning message and continues with the optimization of the input module.

User response:
The warning message may be an indicator of a problem with the input module being optimized. Fixing the problem may require fixing the build steps used to produce the input module. For information about binder API return codes and reason codes, see z/OS MVS Program Management: Advanced Facilities.

Message return code

4

BOZ1491W **Warning: problems encountered adding aliases to directory during binder API "&1": return code=&2 reason code=&3. Module saved and processing continues.**

Explanation

During the optimization process, the binder detected an issue adding aliases to a directory. This problem was detected by the binder API (&1) for which the binder issued a return code (&2) and reason code (&3). An alias cannot be added to a PDS(E) directory when the binder finds that there is a member of the PDS(E) directory with the same name as the name of the alias.

System action:
The binder issues this BOZ1491 warning message and continues with the optimization of the input module.

User response

To resolve this problem, it is important to understand why there is an existing member in the output PDS(E) with the same name as the name of the alias. For example:

- Do not specify an alias name on the member specifier of IN option of a BOPT directive. If an alias name was specified, delete this BOPT directive and delete the member from the target dataset.
- An incorrect member specifier on the OUT option of a BOPT directive could cause a conflict with the name of an alias. Ensure that the OUT option of a BOPT directive has the same member specifier as the member specifier supplied of the IN option.
- Merging aliases from more than one input datasets could cause conflicts with aliases and member names of the two datasets. It is recommended that a different output dataset be used for each input dataset. For information about binder API return codes and reason codes, see *z/OS MVS Program Management: Advanced Facilities*.

Message return code

4

BOZ1492W **Warning: input module "&1" with exported symbols is saved to different named module "&2".**

Explanation

The optimizer detected an input module (&1) in a PDS(E) that included exported symbols and the

optimized module was being written to a differently named member (&2) in an output PDS(E).

System action:
The binder issues this BOZ1492 warning message and continues with the optimization of the input module.

User response:
To fix the problem, change your JCL or SYSIN file such that the member name of the optimized module is the same as the member name of the input module. Failing to do so could result in runtime problems with locating the optimized module because of the change in the member name.

Message return code

4

BOZ1493S **Concatenated DD "&1" encountered and not is allowed for "&2".**

Explanation

The optimizer detected an input or output DD definition (&1) that was the concatenation of two or more datasets. &2 provides the context in which the DDs were used. For example, &2 may indicate that the DD was used as an input module location, or as an output module location, or as an output listing transform location.

System action:
The optimizer bypasses directives that include concatenated DD definitions and the optimizer proceeds to process the next directive.

User response:
Fix your JCL to not include a concatenated DD definition for input modules, output modules and for listing transforms.

Message return code

12

BOZ1494S **Module not processed as it is not fully bound.**

Explanation

During the optimization process, a module was encountered in input that is not fully bound and the ALLOW=NOUNRESEXE option was specified. The optimizer will not process the module that is not fully bound unless the ALLOW=UNRESEXE option is specified.

System action:
The optimizer terminates optimizing the input module and proceeds to process the next module or next directive.

User response:
If the intent is to optimize partially bound modules, remove the ALLOW=NOUNRESEXE option. If the intent is to only optimize fully bound modules, ignore the error, or correct your JCL or SYSIN file to only process fully bound modules.

Message return code

12

BOZ1495W **Rebind of dependent DLL "&1" with RTI bypassed.**

Explanation:
The optimizer emits this warning message when the RTIBIND=IN | OUT | ALL option was specified for the dependent DLL module &1 it's processing.

System action:
The optimizer bypasses rebinding the dependent DLL module &1 and continues to process the next module if it's present. It has no impact on the profiling of the main DLL module. As long as the main DLL is successfully rebound with the RTI Profiler modules then profiling information of the main and the dependent DLL modules will be collected.

User response:
No action is required by the user. In case the RTI Profiler enabled dependent DLL module is required for some other purpose, you can use the Manual RTI rebinding instructions.

Message return code

4

BOZ1496W **Rebind of dependent DLL "&1" ("&2") with RTI bypassed.**

Explanation:
The optimizer issues this warning message when the RTIBIND=IN | OUT | ALL option was specified for the dependent DLL module &1(&2) it's processing.

System action:
The optimizer bypasses rebinding the dependent DLL module &1(&2) and continues to process the next module if it's present. It has no impact on the profiling of the main DLL module. As long as the main DLL is successfully rebound with the RTI Profiler modules then profiling information of the main and the dependent DLL modules will be collected.

User response:
No action is required by the user. In case the RTI Profiler enabled dependent DLL module is required for some other purpose, you can use the Manual RTI rebinding instructions.

Message return code

4

BOZ1497W **CSECT &1 compiled at ARCH(&2) cannot be optimized at the lower ABO ARCH=&3 level.**

Explanation

The optimizer detected a CSECT (&1) which was originally compiled at an ARCH level (&2) higher than the ABO ARCH level (&3).

System action

The optimizer issues this BOZ1497W warning message and skips optimization for this CSECT and continues with the optimization of rest of the module. The optimizer does not support this scenario where the original compiled ARCH level is reduced through optimization.

User response

Ensure that the ARCH level specified for ABO is equal to, or higher than, the ARCH level the input program was compiled at.

Message return code

4

BOZ1498W **Required feature "&1" must be specified under "&2" compilation.**

Explanation:
Binary metadata is required for modules that are compiled under the specified COBOL version (&2), however the data could not be found in the CSECT being optimized.

System action:
ABO bypasses optimization of the CSECT and proceeds to process the next CSECT.

User response

This message is not expected under normal circumstances, because the unsupported message BOZ1455W for NOSMARTBIN should have been emitted earlier, and this message bypassed. If this does happen it means that there is no binary metadata detected in the CSECT even though SMARTBIN was specified. Contact IBM Support.

Message return code

4

BOZ4089I **IEFOPZ: did not get ARCH=&1 match for dataset: '&2' which has ARCH=&3.**

Explanation

While processing an IEFOPZ optimizer directive, the optimizer emits this informational message whenever a NEW dataset (&2) is found in an IEFOPZ configuration that has an ARCH specification (&3) that does not match the SEL_ARCH selector (&1) that was specified on the IEFOPZ optimizer directive.

System action:
The optimizer bypasses the NEW dataset and processes the next NEW dataset in the configuration.

User response:
No action is required by the user.

Message return code

0

BOZ4091I **IEFOPZ: did not get STATE=&1 match for dataset: '&2' which has STATE=&3.**

Explanation

While processing an IEFOPZ optimizer directive, the optimizer emits this informational message whenever a NEW dataset (&2) is found in an IEFOPZ configuration that has a STATE specification (&3) that does not match the SEL_STATE selector (&1) that was specified on the IEFOPZ optimizer directive.

System action:
The optimizer bypasses the NEW dataset and processes the next NEW dataset in the configuration.

User response:
No action is required by the user.

Message return code

0

BOZ4092I **IEFOPZ: did not get DSN='&1' match for dataset '&2'.**

Explanation

While processing an IEFOPZ optimizer directive, the optimizer emits this informational message whenever an OLD dataset (&2) is found in an IEFOPZ configuration that does not match the SEL_OLD selector value (&1) that was specified on the IEFOPZ optimizer directive.

System action:
The optimizer bypasses the OLD dataset and processes the next OLD dataset in the configuration.

User response:
No action is required by the user.

Message return code

0

BOZ4097I **No members in dataset '&1' to process**

Explanation

This message is emitted when there are no members in the dataset to process.

System action:
The optimizer continues processing the next dataset.

User response:
No action is required by the user.

Message return code

0

BOZ4101W **No applicable COBOL code section found, return code 4**

Explanation

This message is emitted in the following cases:

1. When the optimizer encounters a load module but does not optimize any CSECTs within the load module (note: the message is not printed if the REPLACE=Y option is specified and an optimized module already exists)
2. After a BOPT directive that has member wildcards in the "IN" option, but no modules in the "IN" dataset were optimized
3. After an OLD dataset is processed and no modules in an OLD dataset were optimized
4. After an IEFOPZ directive is processed but no modules were optimized

System action:
The optimizer continues processing the next module of input.

User response:
No action is required by the user.

Message return code

4

BOZ4107I **INFO: IDRL record not added to CSECT &1 as load module format does not support three IDRLs. Processing continues.**

Explanation

The optimizer issues this informational message when an IDRL record (for the binary optimizer itself) could not be added to an optimized CSECT in a load module because that CSECT already has 2 IDRL records. Note: a maximum of 2 IDRLs per CSECT is a restriction of load modules in PDS (but is not a restriction for program objects in PDSE).

System action:
The optimizer continues its processing of the output load module.

User response:
No action is required by the user.

Message return code

0

BOZ4109I **INFO: Adding a third IDRL to load module CSECT "&1".**

Explanation

While processing a CSECT (&1) in a load module, the optimizer emits this informational message when the optimizer adds its language record as the third IDRL of the CSECT (&1). Note that an update to the binder is required so that the binder can properly add a third IDRL to a CSECT. If the binder update is not installed on your system, the optimizer will emit a subsequent warning message when attempting to save the optimized load module.

System action:
If the optimizer emits a warning message when saving the module, the language record may not have been added to the CSECT and the optimizer continues processing of the CSECT. Otherwise, processing of the CSECT was successfully performed.

User response:
If the optimizer emits a warning message when saving the module, you should contact your system programmer to install the binder update and perform the optimization process again. Otherwise, no action is required by the user. The required binder update for this message is under APAR OA50460. See “Supported operating systems” on page 3 for more information.

Message return code

0

BOZ4110I **INFO: performing a second bind to handle private section "&1" in class "&2" referring to ENTRY "&3" at offset &4.**

Explanation

The optimizer emits this message when processing a CSECT that had a COBOL ENTRY statement (&3) and there was a reference to the ENTRY (&3) from a private section (&1) from within a class (&2, which is normally

C_WSA) at an offset (&4). Note that an update to the binder is required so that the second bind works successfully. Without the binder update, the optimized program could experience problems.

System action:
If the binder update is available, processing completes successfully. But, if the binder update is not available, the second bind may appear to complete successfully, but, runtime errors may happen.

User response:
If the binder update is installed on your system, no action is required. Otherwise, have your system programmer install the binder update and perform the optimization process again. The required binder update for this message is under APAR OA50460. See "Supported operating systems" on page 3 for more information.

Message return code

0

BOZ4111I **INFO: performing update to private section "&1" in class "&2" referring to ENTRY "&3" at offset &4.**

Explanation

The optimizer emits this message when processing a CSECT that had a COBOL ENTRY statement (&3) and there was a reference to the ENTRY (&3) from a private section (&1) from within a class (&2) at an offset (&4). Note that an update to the binder is required so that the second bind works successfully. Without the binder update, the optmized program will emit an error message when processing the reference.

System action:
If the binder update is not available, the optimizer emits an error processing the reference. Otherwise, the binder processes the reference successfully.

User response:
If the optimizer emitted an error processing the reference, have your system programmer install the binder update and perform the optimization process again. Otherwise, no action is required. The required binder update for this message is under APAR OA50460. See "Supported operating systems" on page 3 for more information.

Message return code

0

BOZ4113I **CSECT &1 was excluded by filter - skip**

Explanation

This message is emitted when a CSECT is excluded by the optimizer due to the expression in the CSECT optimizer option.

System action:
The optimizer continues processing the next CSECT.

User response:
No action is required by the user.

Message return code

0

BOZ4114I **INFO: processing module that is not fully bound with ALLOW=UNRESEXE option in effect.**

Explanation

When the ALLOW=UNRESEXE option is specified, the optimizer issues this informational message when it encounters a module in input that is not fully bound.

This message is not issued if the module is fully bound.

This message can be used to determine which partially bound modules were processed by the optimizer.

System action:
If the binder update is not available, the optimizer emits an error processing the reference. Otherwise, the binder processes the reference successfully.

User response:
The optimizer processes the partially bound module and outputs an optimized partially bound module.

Message return code

0

BOZ4116I **Binder message "&1"**

Explanation

The optimizer uses binder services and a service might fail which prevents a module from being optimized. In response to a binder service that fails, ABO produces a message such as BOZ1429 and terminates processing in some manner. The BOZ1429 message might lack detailed information as to why the binder service failed. For example, the BOZ1429 message indicates that the binder found some problem with the input module but BOZ1429 does not include the precise input problem that the binder found. To provide more information about binder services that fail, the optimizer captures severe binder messages and includes the text of a binder message within &1 of the BOZ4116 informational message. This means that,

when a binder service fails, ABO normally emits two messages:

1. The BOZ4116 message with &1 holding the text of a severe binder message. The BOZ4116 message is followed by
2. A summary message such as BOZ1429, indicating the general nature of binder failure.

System action:
See the "System action" section of the summary message to determine the actions of the optimizer.

User response:
See the "User response" section of the summary message to determine what to do. The BOZ4116 message might provide information that helps guide your response.

Message return code

0

BOZ4117I **Member "&1" was excluded by filter - skip**

Explanation

This message is emitted when a module is excluded by the optimizer due to the expression in the IN option of the BOPT directive.

System action:
The optimizer continues processing the next module.

User response:
No action is required by the user.

Message return code

0

BOZ4119S **Continuation indicated on SYSIN line &1. Unable to read SYSIN line &2.**

Explanation

Continuation was indicated on line &1 of SYSIN with the last non-blank character of line &1 being a continuation char (either '+' or '-'). While reading SYSIN, the optimizer was unable to read line &2.

System action:
The optimizer discards the line.

User response:
Either remove the continuation character at the end of line &1 of SYSIN, or add a new line &2 to the SYSIN that will continue line &1.

Message return code

12

BOZ4120S **Cannot have more than one IN= or OUT= specifier.**

Explanation

This message is emitted when more than one IN or OUT specifier is detected in a BOPT directive.

System action:
The optimizer continues processing the next BOPT directive.

User response:
Remove IN or OUT specifiers from the BOPT directive until there is no more than one of each per directive.

Message return code

12

BOZ4121S **Invalid log specification. '&1' is not a directory.**

Explanation

Only directories are valid specifiers for the LOG option. &1 is not a directory.

System action:
The LOG option is ignored.

User response:
Correct the LOG specification to a supported format, see the LOG option for supported specifications.

Message return code

12

BOZ4124I **The HANDLERS option is now deprecated and no longer necessary; the previously default HANDLERS=Y behaviour now always applies.**

Explanation

The optimizer option HANDLERS is now deprecated and is no longer supported nor required.

System action:
Regardless of the HANDLERS option specified, the optimizer will always behave as if the previously default and conservative HANDLERS=Y option is in effect.

User response:
No action is required by the user.

Message return code

0

BOZ4125I **The binary metadata contains invalid data.**

Explanation

The binary metadata in the CSECT mentioned in an earlier message is invalid.

System action

The optimizer issues warning message BOZ4088W and proceeds to process the next CSECT.

User response

Contact IBM support.

Message return code

0

BOZ4126I The binary metadata format is not supported by the optimizer.

Explanation

The binary metadata in the CSECT mentioned in an earlier message is of a different format than that supported by the optimizer.

System action

The optimizer issues warning message BOZ4088W and proceeds to process the next CSECT.

User response

Contact IBM support.

Message return code

0

BOZ4128W Optimization bypassed for not LE enabled member "&1"

Explanation

The optimizer detected a not LE enabled module (&1). These types of modules are not eligible for optimization.

System action:
The optimizer bypasses optimizing the input module and proceeds to process the next module.

User response:
No action is required by the user.

Message return code

4

BOZ4131S Required z/OS PTF for APAR &1 has not been installed.

Explanation

The optimizer requires certain PTFs to be installed as pre-requisites, otherwise it cannot continue.

System action

The optimizer will terminate all processing immediately.

User response

Check the pre-requisites of the optimizer, and install all the PTFs listed there including the one in the given APAR number of this message.

Message return code

12

BOZ4132W SERVICE LABEL is missing for &1.

Explanation

The optimizer detected that a required SERVICE LABEL is missing for a call to (&1).

System action

The optimizer skips optimization for the current CSECT and proceeds to process the next CSECT.

User response

A missing SERVICE LABEL is an indicator of an issue with the original source of the original compilation. For ABO to optimize this CSECT, the source would need to be modified.

Message return code

4

BOZ4133I Debug file &1 was generated for CSECT&2.

Explanation

The optimizer wrote updated DWARF debug information for the separate debug file successfully for the given CSECT. Note that the message id BOZ4133I for this message is not shown in the OPTLOG.

System action

The optimization proceeds as normal.

User response

None required.

Message return code

0

BOZ4134I **Debug information was removed or not generated for CSECT &1.**

Explanation

The optimizer could not generate updated DWARF debug information for the given CSECT. The optimized CSECT will appear to have been compiled with NOTEST(NODWARF); if the module was compiled with TEST(NOSEPARATE), the original DWARF will remain in the module. Note that the message id BOZ4134I for this message is not shown in the OPTLOG.

System action

The optimization proceeds as normal, but updated debug information will not be present, and it will appear to subsequent tooling that the modules was compiled under NOTEST(NODWARF), and therefore may not behave as expected. The return code of earlier messages will override the return code of this message.

User response

See the User Response section(s) of earlier messages in the log.

Message return code

0

BOZ4135I **Debug information was generated for CSECT &1.**

Explanation

The optimizer wrote updated DWARF debug information into the optimized module for the given CSECT. Note that the message id BOZ4135I for this message is not shown in the OPTLOG.

System action

The optimization proceeds as normal.

User response

None required.

Message return code

0

BOZ4136I **"&1" is supported up to SLEVEL &2.**

Explanation

The CSECT was compiled by the COBOL Compiler version and release given in &1. The optimizer supports such CSECTs up to an SLEVEL value of &2. A previous BOZ1455W message shows that the CSECT has an SLEVEL that is above this value. See “Handling CSECT eligibility for COBOL compilers that are released after ABO” on page 11 for further explanation.

System action

A BOZ1455W message was issued prior to this message; the optimizer bypasses optimization of the CSECT and proceeds to process the next CSECT.

User response

See “Handling CSECT eligibility for COBOL compilers that are released after ABO” on page 11. There may be a later ABO version that supports the SLEVEL reported in the previous BOZ1455W message.

Message return code

0

RTI Profiler messages

BOZR001I **SYSPROFD: "&1"**

Explanation

When the SYSPROFD DD card is not provided in the JCL of the step executing a *program* being profiled with the RTI Profiler, then a default data set (&1) either is allocated dynamically with attributes: PS FB 80 27920 and the following data set name format: *userid*.BOZRTI.SYSPROFD.*program*

This informational message displays the name of this data set that holds the profiling output so it can be located by the user.

System action

The RTI report is being generated in (&1) data set.

User response

The displayed data set name contains the RTI Profiler output location in cases where SYSPROFD DD card is not provided.

Message return code

0

BOZR002S AUTH REQUEST FAILED (RC="&1", RSN="&2")

Explanation

The current task has requested to be authorized for the RI facility through HISUSER REQUEST=AUTH with store key 0 to 15, but the request failed with return code &1 and reason code &2.

System action

The RTI Profiler immediately terminates execution and returns to the operating system with a return code of ABEND=S000 U1130

.

User response

This message may be issued due to an earlier problem. Correct any problems reported in the JESMSGLG file and retry the RTI Profiler execution. If the problem persists, contact IBM support.

Message return code

-

BOZR003S ATTACHING BOZRIDT FAILED (RC="&1")

Explanation

The ATTACH macro request issued for the BOZRIDT module failed with return code &1.

System action

The RTI Profiler immediately terminates execution and returns to the operating system with an ABEND=S000 U1130.

User response

If the return code in the message is 8, then this may be an indication of a transient environmental error and the RTI Profiler execution can be retried. If this does not resolve the problem, or for any other nonzero return code, contact IBM support.

Message return code

-

BOZR004S CORRUPTED RI BUFFER

Explanation

The RICHK thread has discovered unrecoverable RI buffer corruption.

System action

The RTI Profiler immediately terminates execution and returns to the operating system with an ABEND=S000 U1130.

User response

It may be an indication of the environmental or machine check error, retry request. If the problem persists, contact IBM support.

Message return code

-

BOZR005S OPENING SYSPROFD FAILED

Explanation

Opening of the file allocated under the SYSPROFD DD name failed. Possible reasons for this failure could be file corruption, or the file may be held exclusively by another task.

System action

The RTI Profiler immediately terminates execution and returns to the operating system with an ABEND=S000 U1130.

User response

Verify that your SYSPROFD file is correctly allocated and available, then retry RTI Profiler execution. If the problem persists, contact IBM support.

You can also try not specifying a SYSPROFD DD in your JCL and let the RTI Profiler dynamically allocate the Profiler output file. See BOZR001I message description for more information.

Message return code

-

BOZR006E READQ TS FAILED (RESP=&1, RESP2=&2)

Explanation

Reading the temporary storage queue prefixed BOZRTIQ. failed. The failure reason is indicated by the CICS response code values returned as &1 and &2.

System action

The RTI Profiler immediately terminates execution, but the original modules that are linked with RTI program continue to run.

User response

Check the meaning of &1 and &2 values in the CICS READQ TS conditions description READQ TS.

For example, if RESP=17 and RESP2=5, then this is an input/output error. The single occurrence of this condition can be ignored. In this and some other code &1 and &2 combinations, the problem can be fixed by sysadmin. Otherwise, contact IBM support.

Message return code

-

BOZR007E **WRITEQ TS FAILED (RESP=&1, RESP2=&2)**

Explanation

Writing the temporary storage queue prefixed BOZRTIQ. failed. The failure reason is indicated by the CICS response code values returned as &1 and &2.

System action

The RTI Profiler immediately terminates execution, but the original modules that are linked with RTI program continue to run.

User response

Check the meaning of &1 and &2 values in the CICS WRITEQ TS conditions description WRITEQ TS.

For example, if RESP=18, then this is a failure regarding "no space in the storage". The single occurrence of this condition can be ignored. In this or some other code &1 and &2 combinations, the problem can be fixed by sysadmin. Otherwise, contact IBM support.

Message return code

-

BOZR008W **The RTI number of logs limit has been set to &1**

Explanation

The RTIS transaction issued from the CICS terminal has changed the RTI number of logs limit per program to &1. The purpose of this message is to keep you informed about this limit change because it may not match your expectation of the number of logs produced per program if this change was performed by another user.

System action

The message is being generated into the CICS job log. Process continues.

User response

-

Message return code

-

BOZR009E **READQ TS FAILED (RESP=&1, RESP2=&2)**

Explanation

Reading the temporary storage queue with prefix BOZRTIQ@ failed. The failure reason is indicated by the CICS response code values returned as &1 and &2.

System action

The RTI Profiler immediately terminates execution, but the original modules that are linked with RTI program continue to run.

User response

Check the meaning of &1 and &2 values in the CICS READQ TS conditions description READQ TS.

For example, if RESP=70 and RESP2=101, then this is a security check failure. You need to obtain authorization from your system administrator to run RTI in CICS.

In some other code &1 and &2 combinations, the problem can be fixed by sysadmin. Otherwise, contact IBM support.

Message return code

-

BOZR010E **WRITEQ TS FAILED (RESP=&1, RESP2=&2)**

Explanation

Writing the temporary storage queue with prefix BOZRTIQ@ failed. The failure reason is indicated by the CICS response code values returned as &1 and &2.

System action

The RTI Profiler immediately terminates execution, but the original modules that are linked with RTI program continue to run.

User response

Check the meaning of &1 and &2 values in the CICS WRITEQ TS conditions description WRITEQ TS.

For example, if RESP=70 and RESP2=101, then this is a security check failure. You may need to obtain authorization from your system administrator to run RTI in CICS.

In some other code &1 and &2 combinations, the problem can be fixed by sysadmin. Otherwise, contact IBM support.

Message return code

-

ABO Assistant messages

BOZA001E **Invalid value "&1" specified with THRSHOLD parameter.**

Explanation

Invalid value &1 specified for the required input parameter THRSHOLD of ABO Assistant job BOZSMFJ. The expected value is a decimal number. The THRSHOLD parameter accepts any digit's combination with or without decimal point, but it rejects any combination with non-digit character included, or if no value specified for the parameter.

System action:
The BOZSMFJ job fails with JCL ERROR.

User response:
Correct the value &1 specified for the job BOZSMFJ parameter THRSHOLD, then rerun the job.

Message return code

8

BOZA002W **No report generated in "&1".**

Explanation

When customizing the ABO Assistant job BOZSMFJ step PGMHOME, there is a possibility of generating the programs' location report by supplying a list of JCL libraries within the in-stream file of this step starting after the JCLLIST DD name. This message informs the user that the report won't be generated either because the JCLLIST in-stream file is empty, or because it contains of one or more nonexistent data sets. The message BOZA004W or BOZA005W accompanies this BOZA002W message clarifying the reason for no report being generated.

System action

The BOZSMFJ job completes and generates all required reports except the optional JCL location section.

User response:
Optionally, adjust the JCLLIST content, then rerun the BOZSMFJ job.

Message return code

4

BOZA003W **Report is generated in "&1", but some file listed in JCLLIST does not exist.**

Explanation

When customizing the ABO Assistant job BOZSMFJ step PGMHOME there is a possibility of generating the programs' location report by supplying a list of JCL libraries within the in-stream file of this step starting after the JCLLIST DD name. This message informs the use that the report is generated, however it might not be complete because the JCLLIST provided within in-stream file contains one or more nonexistent data sets. The message BOZA005W accompanies this BOZA003W message clarifying the list of nonexistent data sets.

System action:
The BOZSMFJ job completes generating all required reports.

User response

Look for the BOZA005W message accompanying this BOZA003W message to clarify the list of nonexistent data sets. Optionally, adjust the JCLLIST content, then rerun the BOZSMFJ job.

Message return code

4

BOZA004W **JCLLIST is missing.**

Explanation

When customizing the ABO Assistant job BOZSMFJ step PGMHOME there is a possibility of generating the programs' location report by supplying a list of JCL libraries within the in-stream file of this step starting after the JCLLIST DD name. This message accompanies the earlier message BOZA002W to inform the user that their JCLLIST file is empty.

System action:

The BOZSMFJ job completes generating all required reports.

User response:
Optionally, adjust the JCLLIST content, then rerun the BOZSMFJ job.

Message return code

4

BOZA005W Check JCLLIST names: “&1”.

Explanation

When customizing the ABO Assistant job BOZSMFJ step PGMHOME there is a possibility of generating the programs’ location report by supplying a list of JCL libraries within the in-stream file of this step starting after the JCLLIST DD name. This message is accompanying the earlier messages, BOZA002W or BOZA003W, to inform the user about the list &1 of nonexistent data sets provided in the JCLLIST in-stream file.

System action:
The BOZSMFJ job completes generating all required reports.

User response:
Optionally, adjust the JCLLIST content, then rerun the BOZSMFJ job.

Message return code

4

BOZA006E Data set “&1” could not be found.

Explanation

One of the BOZPAJ input parameters (JCLNAME, ABO or PROCNAME) specifies the nonexistent data set name &1.

System action:
The BOZPAJ initialization job fails.

User response:
Correct the data set name &1, then rerun the BOZPAJ job.

Message return code

8

BOZA007E Invalid data set “&1”.

Explanation

One of the BOZPAJ input parameters (JCLNAME, ABO or PROCNAME) specifies the invalid data set name &1.

System action:
The BOZPAJ initialization job fails.

User response:
Correct the data set name &1, then rerun the BOZPAJ job.

Message return code

8

BOZA008E Invalid OPTLOAD “&1”.

Explanation

The BOZPAJ job input parameter OPTLOAD specifies the invalid value &1. The OPTLOAD parameter value may be set to TEMP to use a temporary data set, or any valid data set name to store the optimized modules in.

System action:
The BOZPAJ initialization job fails.

User response:
Correct the OPTLOAD parameter value &1, then rerun the BOZPAJ job.

Message return code

8

BOZA009E Invalid DYNSCAN “&1”.

Explanation

The BOZPAJ job input parameter DYNSCAN specifies the invalid value &1. The only valid option to be specified with parameter DYNSCAN is Y or N. Y initiates a module scan for COBOL, N means no scan is performed.

System action:
The BOZPAJ initialization job fails.

User response:
Correct the DYNSCAN parameter value &1, then rerun the BOZPAJ job.

Message return code

8

BOZA010E Error allocating data set “&1”.

Explanation

The BOZPAJ job input parameter OPTLOAD specifies the data set name &1 that cannot be allocated. Look for the accompanying messages in the job log and SYSTSPRT file for more information and messages on this error.

System action:
The BOZPAJ initialization job fails.

User response:
Correct the error, then rerun the BOZPAJ job.

Message return code

8

BOZA011E **No EXEC card provided in “&1”.**

Explanation

The JCL source located in the data set &1 specified by the BOZPAJ input parameter JCLNAME or PROCNAME does not contain any JCL EXEC card.

System action:
The BOZPAJ initialization job fails.

User response:
Ensure that data set specified by the jobs BOZPAJ input parameter JCLNAME or PROCNAME is a correct JCL source location, then rerun the BOZPAJ job.

Message return code

8

BOZA012E **No JOB card provided in “&1”.**

Explanation

No JOB card found in the data set &1 specified as the JCL source location by the BOZPAJ input parameter JCLNAME.

System action:
The BOZPAJ initialization job fails.

User response

Ensure that the data set specified by the BOZPAJ input parameter JCLNAME is a correct JCL source, then rerun the BOZPAJ job.

Message return code

8

BOZA013E **No step with PGM=”&1” located in “&2”.**

Explanation

The JCL source located in the data set &2 specified by the BOZPAJ input parameter JCLNAME or PROCNAME, does not contain any step with the program name &1 specified in the BOZPAJ input parameter PGMNAME.

System action:
The BOZPAJ initialization job fails.

User response

Ensure that data set specified by the BOZPAJ input parameter JCLNAME or PROCNAME is a correct JCL source location, then rerun the BOZPAJ job.

Message return code

8

BOZA014E **No STEPLIB provided for ”&1” step in “&2”.**

Explanation

No STEPLIB card found for the step &1 in the JCL source located in the data set &2 specified by the BOZPAJ input parameter JCLNAME or PROCNAME. In this case, step &1 is expected to be the execution step of the program specified in the BOZPAJ input parameter PGMNAME.

System action:
The BOZPAJ initialization job fails.

User response

Ensure that data set specified by the BOZPAJ input parameter JCLNAME or PROCNAME is a correct JCL source location, then rerun the BOZPAJ job.

Message return code

8

BOZA015W **Architecture level for ABO Assistant job #1 ARCH=“&1” mismatch architecture level for job #2 ARCH=“&2”.**

Explanation

The ABO Assistant job #2 has detected discrepancy in architecture level of the systems where job #1 and job #2 run. The system architecture level of the job #1 is &1 and the system architecture level of the job #2 is &2.

System action

The ABO Assistant job #1 completes normally. Then job #2 completes with this warning message indicating a possible inaccuracy in the report generated.

User response:
The described above discrepancy may be a result of the predefined job routing. To avoid it, you can adjust the original job JCL system affinity or routing directives to ensure the same single system selection for each job submission. Then rerun the BOZPAJ job.

Message return code

4

BOZA016E **TSO SUBMIT command failure.**

Explanation

The TSO user running ABO Assistant may be not authorized to use the TSO SUBMIT command.

System action:
The BOZPAJ initialization job fails.

User response:
Obtain TSO user authorization for using TSO SUBMIT command, then rerun the BOZPAJ job.

Message return code

8

BOZA017E **z/OS version: “&1” is not a supported z/OS version to run ABO.**

Explanation

The ABO Assistant job #1 or 2 has automatically detected that z/OS version &1 of the current run doesn’t meets minimum requirement for ABO. See “Supported operating systems” on page 3 for more information.

System action:
The BOZPAJ job #1 or 2 fails.

User response:
Ensure the system being used for the failed ABO Assistant job meets minimum z/OS system version level requirement for ABO.

Message return code

8

BOZA018E **IBM z server: “&1” is not a supported hardware level to run ABO optimized modules.**

Explanation

The ABO Assistant job #1 or 2 has automatically detected that IBM z server hardware level &1 of the current run doesn't meet the minimum requirement for running ABO optimized modules.

System action:
The BOZPAJ job #1 or 2 fails.

User response

Ensure the system being used for the failed ABO Assistant job meets the minimum IBM zSystem hardware level requirement for ABO. For more information, see “Target hardware levels” on page 4.

Message return code

8

BOZA019E **Program “&1” not in the step “&2” STEPLIB.**

Explanation

The program &1 specified with the ABO Assistant input parameter PGMNAME is not found in any data set concatenated under the STEPLIB DD card of the step &2. However, the step &2 is expected to be the program &1 execution step accordingly with the EXEC card found in this step.

System action:
The BOZPAJ initialization job fails.

User response:
Ensure that JCL source provided by ABO Assistant input parameter JCLNAME or PROCNAME contains a correct JCL for the step &2 execution, then rerun the BOZPAJ job.

Message return code

8

Appendix D. Run Time Instrumentation report

Note: The RTI Profiler output format is subject to change at any time and with no prior notice. Therefore, do not rely on the specific layout or content of the output as described below in building any post-processing tooling that uses the RTI Profiler output.

The output of the RTI Profiler is a text file split into the following sections:

1. RTI options section
2. Library Resource section (for CICS only)
3. Module info section
4. Summary report section
5. Sampling tables section
6. Module locations section
7. Report Totals section

RTI options section

The RTI options section displays the default RTI configuration. These options are fixed and cannot be changed. For example:

```
*******************************************************************************
* RTI options                                                  2020/03/02 14:23:04 *
*******************************************************************************
SCALING-FACTOR=0000500000
ANALYSIS-MODE=1, PDF-MODE=0, VERBOSE=0
BUF-SIZE=0000001024, #BUFS=0000000004, TIMER-INTERVAL=0000000050
*******************************************************************************
```

Library Resource section

The Library Resource section displays the CICS library resource digital signature taken before the RTI Profiler-enabled program starts execution. It includes a primary characteristic of the library at definition, installation, and modification time. It allows for maintaining the 3-point library auditing through the CICS RTI-generated report. The Library Resource section describes the first audit point. The second audit point is established in the Module info section when the RTI Profiler-enabled program starts execution, for example, when RTI reports Module info corresponding with the first sample of this program. The end of the RTI report establishes the third audit point in the Module location section, clarifying the program location after its execution. The following example shows a library resource section:

```
*******************************************************************************
* List of Libraries                                                           *
*******************************************************************************
Library=DFHRPL  , Status=ENABLED   , Rank=10, Critical=YES
        | Agent     | User     | Source   | Release | Time
Install | SYSTEM    | USERA1   |          |         | 2022/10/19 17:50:06
Change  | SYSTEM    | USERA1   |          | 0720    | 2022/10/19 17:50:06
Define  |           |          | SYSTEM   |         | 2022/10/19 17:50:06
Data set list:
CICS.TS55.SEYULOAD
CICS.TS55.SDFHLOAD
COBOL.TS55.SDFHAUTH
CEE.SCEECICS
CEE.SCEERUN
CEE.SCEERUN2
HLQ.ABO.LOAD
*-----------------------------------------------------------------------------*
Library=CICSST0 , Status=DISABLED  , Rank=40, Critical=NO
        | Agent      | User     | Source   | Release  | Time
Install | GRPLIST    | USERB1   |          |          | 2022/10/19 17:50:08
Change  | CSDBATCH   | USERA1   |          | 0720     | 2022/09/22 16:05:44
```

```
Define  |            |          | LIBS     |          | 2022/09/22 16:05:44
Data set list:
USER.LIBRARY.ONE
USER.LIBRARY.TWO
*-------------------------------------------------------------------------------*
Library=CICS001 , Status=ENABLED   , Rank=45, Critical=NO
        | Agent      | User     | Source   | Release | Time
Install | GRPLIST    | USERB1   |          |         | 2022/10/19 17:50:08
Change  | CSDBATCH   | USERC1   |          | 0720    | 2022/09/22 16:05:44
Define  |            |          | LIBS     |         | 2022/09/22 16:05:44
Data set list:
USER.LIBRARY.THREE
*-------------------------------------------------------------------------------*
```

Module info section

Module info section for batch

Detailed module information is displayed next. This information includes the module name (specified by MJNAME), address, and length and other information on the content of the module. For example:

```
*******************************************************************************
* Module Info                                                                 *
*******************************************************************************
QUERYING MODULE INFO: ADDR=X'26EA5154'
CSVQUERY: RC=00, VALID=X'573C0800'
MJNAME=COBMOD
EPTKN=X'000003B703E8003B'
EPA=X'26EA4C48'
EXTENT=X'0001', ADDR=X'26EA4C48', LEN=X'000013B8'
ATTR1=X'08'
ATTR2=X'10'
ATTR3=X'C0'
XATTR1=X'00'
PID=PGMF
DIAG=X'7F6E4F48'
MJNAME=COBMOD , EPADDR=X'26EA4C48'
EXTENT=X'0001', ADDR=X'26EA4C48', LEN=X'000013B8'
OPENING BINDER INTERFACE
CLASS=B_TEXT          , SEG=X'0001', LOAD=X'0', OFF=X'00000000', LEN=X'000013B8'
CLASS=B_MAP           , SEG=X'0001', LOAD=X'2', OFF=X'00000000', LEN=X'00000000'
...
SEG=X'0001' => EXTENT=X'0001'
```

The module info section is followed by a list of all the CSECTs and the release and version information for the translator(s) that produced the CSECT. The translator is typically a compiler, optimizer or assembler product. For example:

```
*******************************************************************************
* List of CSECTs processed for module COBMOD                                  *
*******************************************************************************
COBMAIN: Enterprise COBOL V4       , VER=42, MOD=00, PID=5655S7100
                        TYPE=B_TEXT          , OFF=X'00000000', LEN=X'00000988'
                        ADDR=X'26EB6A30'-X'26EB73B7'
COBMAIN: Automatic Binary Optimizer, VER=13, MOD=00, PID=5697-AB1
SUB1: Enterprise COBOL V4       , VER=42, MOD=00, PID=5655S7100
                        TYPE=B_TEXT          , OFF=X'00000670', LEN=X'00000C26'
                        ADDR=X'26900670'-X'26901295'
CEESTART: HIGH-LEVEL ASSEMBLER      , VER=01, MOD=06, PID=569623400
                        TYPE=B_TEXT          , OFF=X'00001F00', LEN=X'000000B0'
                        ADDR=X'26901F00'-X'26901FAF'
...
```

Each entry in the CSECT list represents CSECT details in the format:

```
Module name: Translator name, Major and minor version, mod level, PID
```

There are also additional data provided, such as CSECT offset (OFF) in the module, length (LEN) of the CSECT, and the loaded address (ADDR) of the CSECT.

Note that some CSECTs may have been translated by more than one product, in which case an additional entry appears one for each translator used. In this example, there are two entries for the CSECT

COBMAIN – one for Enterprise COBOL 4 and another for ABO. This indicates that COBMAIN is the ABO optimized version of a program originally compiled by Enterprise COBOL 4.

If no translator name is available then a dash sign (-) is displayed in the translator name field, but the PID of the translator product is always displayed.

Module info section for CICS

The module information for CICS RTI differs from the batch version. The main difference is that the CICS-specific program arguments display after the block of module location among libraries information. This block represents the second audit point of the library resource provided when RTI processes the first sample for the program. For example:

```
*******************************************************************************
* Module Info
*
    *
*******************************************************************************
QUERYING MODULE INFO: ADDR=X'2A29C17A'
BOZQUERY: RC=00
MJNAME=SMPMOD11
EPA=X'2A2991C0'
EXTENT=X'0001',ADDR=X'2A2991C0',LEN=X'0000E6B8'
CICS_PROG_ATTR=REUSABLE
OPENING BINDER INTERFACE
INCLUDING MEMBER SMPMOD11                                2023/01/27 10:17:23.206
FROM| LIBRARY | STATUS    | DSNAME                                        | VOLSER
    | DFHRPL  | ENABLED   |                                                |
    | CICSST0 | DISABLED  |                                                |
 => | SMP0003 | ENABLED   | YOUR.LIBRARY.LOAD.RTILNKS                      | SMPVOL1
    | SMPNOOPT| ENABLED   |                                                |
CLASS=B_TEXT          , SEG=X'0001', LOAD=X'0', OFF=X'00000000',  LEN=X'0000E6B8'
CLASS=B_MAP           , SEG=X'0001', LOAD=X'2', OFF=X'00000000',  LEN=X'00000000'
...
SEG=X'0001' => EXTENT=X'0001'
```

Summary report section

A high-level summary report section is displayed next, which includes a summary report by language subsection and a summary report by module subsection. This section is useful for obtaining an overall view of CPU performance without having to drill down into the detailed profiling information per CSECT.

For example:

```
*******************************************************************************
* Summary report                                          2020/03/02 14:23:18 *
*******************************************************************************
* CPU TIME 00:00:56.30                                                        *
*******************************************************************************
#TOTAL SAMPLES=0000000000012450, #MAPPED SAMPLES=0000000000011946
*******************************************************************************
* Summary report by language                                                  *
*******************************************************************************
Language         |  Number Of Samples |  % TOTAL
 COBOL           |              32724  |   ( 100.00% )
    Application  |                 12  |   (   0.03% )
    LE           |              32712  |   (  99.96% )
 PLI             |                  0  |   (   0.00% )
 C/C++           |                  0  |   (   0.00% )
 Java            |                  0  |   (   0.00% )
*******************************************************************************
* Summary report by module                                                    *
*******************************************************************************
MODULE=COBMOD, EPADDR=X'26B2E8E0', #SAMPLES=0000000000011946, ( 93.55% )
CSECT Name | StartADDR     | Number Of Samples | % of COBMOD | % TOTAL
COBMAIN    |  X'26B954F8'  |             10234 | ( 85.67% )  | ( 82.20% )
SUB1       |  X'26B96D18'  |              1203 | ( 10.07% )  | ( 9.66% )
SUB2       |  X'26B97100'  |               507 | ( 4.24% )   | ( 4.07% )
RICHK      |  X'26B81B28'  |                 2 | ( 0.02% )   | ( 0.02% )
MODULE=IGZCPAC, EPADDR=X'27239318', #SAMPLES=0000000000000430, ( 3.45% )
CSECT Name | StartADDR      | Number Of Samples | % of IGZCPAC  | % TOTAL
IGZCXDI    |  X'27196F78'   |               267 | ( 62.09% )    |  ( 1.87% )
IGZCXMU    |  X'272411B0'   |               163 | ( 37.91% )    |  ( 1.33% )
```

```
MODULE=MODC, EPADDR=X'270FCA30', #SAMPLES=0000000000000074, ( 0.59% )
*******************************************************************************
```

RTI for batch reports the CPU time of a program that is running in the `CPU TIME` section in the `hh:mm:ss.th` format.

CICS RTI reports the CPU time of a program that is running in the `CPU TIME` section and in the `hh:mm:ss.ssssss` format.

The portion of CICS activity is included in the `Summary report` by language block as shown in the following example:

```
*******************************************************************************
* Summary report                                          2023/01/27 10:17:23 *
*******************************************************************************
* CPU TIME 00:00:00.002139                                                    *
*******************************************************************************
#TOTAL SAMPLES=0000000000000023, #MAPPED SAMPLES=0000000000000023
*******************************************************************************
* Summary report by language                                                  *
*******************************************************************************
Language       |  Number Of Samples |  % TOTAL
 CICS          |                 13 |   (  56.52% )
 COBOL         |                  8 |   (  34.78% )
   Application |                  7 |   (  30.43% )
   LE          |                  1 |   (   4.34% )
 PLI           |                  0 |   (   0.00% )
 C/C++         |                  0 |   (   0.00% )
 Java          |                  0 |   (   0.00% )
*******************************************************************************
```

Note:

- For the RTI Profiler-enabled programs running in CICS, the RTI report displays the CPU TIME in the above `Summary report` only for the execution of the first program.
- For the subsequent RTI Profiler-enabled programs running in CICS, the elapsed `CPU TIME` is displayed.

All numeric values in the profiling report are displayed in either decimal or hexadecimal. All hexadecimal values in the profiling output are indicated by wrapping the number in the X` ` notation. If the value is not wrapped in the X` `notation, then it is in a decimal representation. All the samples values in the profiling report are in decimal.

A larger number of samples attributed to a particular module or CSECT means a higher relative amount of CPU time is spent in these parts of the application when it ran.

The percentage values provided per module (for example, 93.55% for MODULE=COBMOD) are the total number of samples found for this module out of the overall TOTAL SAMPLES.

The percentage values provided per CSECT (for example, 62.09% for CSECT IGZCXDI) are the number of samples found for this CSECT out of the overall SAMPLES found for only the particular module where this CSECT is contained (MODULE=IGZCPAC in this case).

Note that it is normal and expected to see CSECT names starting with the RI prefix (such as RICHK in this example) in the profiling report. These RI* CSECTs are part of the RTI program modules bound into the modules using RTIBIND and they are used to manage the profiling and report collection. The samples attributed to these RI* CSECTs will be very low as the profiling overhead using the RTI Profiler is negligible.

Sampling tables section

This next section provides a detailed per CSECT breakdown that shows the samples attributed to the CSECT offsets.

A samples count is shown for an offset if at least one sample was attributed to this offset. The offsets are sorted in increasing order and match those found in the compiler or optimizer listing file.

For example:

```
*******************************************************************************
* Sampling tables                                                             *
*******************************************************************************
MODULE=IGZCPAC , EXTENT=X'0001', #SAMPLES=0000000000000430
CSECT=IGZCXDI, #SAMPLES=0000000000000267
TABLE=X'26D970B0'
SIZE=X'00000048', ALIGN=X'00000002', #ENTRIES=0000000082
  Offset          | Number Of Samples |
   X'00000000'    |                25 |
   X'00000028'    |                10 |
   X'0000002C'    |                 3 |
   X'0000002E'    |                12 |
   X'00000032'    |                38 |
   X'00000040'    |                 8 |
   X'00000048'    |                32 |
...
MODULE=COBMOD , EXTENT=X'0001', #SAMPLES=0000000000011946
CSECT=COBMAIN, #SAMPLES=0000000000010234
TABLE=X'26EB51A0'
SIZE=X'00000077', ALIGN=X'00000002', #ENTRIES=0000000006
  Offset          | Number Of Samples |
   X'000004E2'    |                 1 |
   X'000004EC'    |                 1 |
   X'000004F8'    |                 2 |
   X'00000506'    |               121 |
   X'0000050C'    |               411 |
   X'0000051A'    |               234 |
   X'0000051E'    |                34 |
...
```

Module locations section

For each module profiled by RTI, this section provides information about the module locations. Such as, the module entry point address, followed by the name of the data set from which the module was loaded for execution, followed by the volume serial number where the data set is located.

For example:

```
*******************************************************************************
* Module locations                                                            *
*******************************************************************************
Module   | EPADDR      | DSNAME                                          | VOLSER
IGZCPAC  | X'26B2E8E0' | SYS2.CEE.SCEERUN                                | SYS004
IGZXLPKA | X'27239318' | SYS2.CEE.SCEERUN                                | SYS004
IGZOLPA  | X'2710C390' | N/A                                             | N/A
COBMOD   | X'270FCA30' | USER1.OPTLOAD3                                  | PROD79
COBOL1   | X'27105000' | USER1.INLOAD                                    | TEST0A
*******************************************************************************
```

The STEPLIB data set concatenation is searched first for the module location. If not found in the STEPLIB, then the JOBLIB data set concatenation is searched. If the module is not found in these concatenations, then its data set and volume names are marked as N/A (not available).

There is no difference between the appearance of the module locations section for RTI running in the batch and the CICS environment. However, the search order used for the module location determination after the program execution differs. In the CICS environment, a program can be loaded from the associated with CICS library resource DD name concatenation. The DD name from which the module is executed will be searched by RTI first. If the module is found, then the data set name will be displayed. Otherwise, its data set and volume names are marked as N/A, which indicates not available. This section fulfills the 3-point library auditing provided by CICS RTI. For more information about 3-point library auditing, check “Library Resource section” on page 123.

Report totals section

At the bottom of the RTI report, the sum totals for the internal state used to store and process the RTI data are displayed.

This portion of the output does not normally need to be examined and is used for IBM diagnostic purposes only. For example:

```
List=0000004192 bytes, ListEntry=0000032053 bytes
SamplingTable=0001101276 bytes, SamplingEntry=0000028080 bytes
#ListEntries=0000000517, #SamplingEntries=0000001404
RIBuffers=0004194304 bytes
```

RTIS transaction

RTIS is an interactive CICS transaction provided with the RTI Profiler. RTIS allows you to display or modify the current limit of reports generated per RTI Profiler-enabled program. This transaction is available from the CICS terminal if it is defined among other RTI-related CSD definitions. For more information, see “Capturing profiling results during CICS application program execution” on page 30. Note that the transaction name *RTIS* is optional and can be customized.

If a particular RTI Profiler-enabled program executes many times during the application run, then by default, three profiling reports are generated for this program. The name of the dynamically allocated RTI report file has the following format:

```
userid.BOZRTI.SYSPROFD.pgmname.LOG#nnn
```

The nnn corresponds with the report number generated for the program named pgmname.

For the RTI Profiler-enabled program pgmname’s first run, the report's suffix will be LOG#001.

For each subsequent run of the same program, the corresponding report name suffix will be increased by 1. However, the limit of profiling reports generated per program can be modified using RTIS transaction. You can follow the syntax:

```
RTIS [n]
```

where n is a number in the range of 0:250. If RTIS is issued without parameter:

```
RTIS
```

then the current limit is displayed. For example:

```
The maximum number of RTI logs per program is 3
```

When you have many RTI Profiler-enabled programs in your CICS application, a bigger limit can be helpful to show program execution frequency.

Example 1: a default number of three reports per program

The below fragment of the CICS job log shows an example of the RTI report files generation for the default number of three reports per program:

```
09.05.09 JOB07887  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#001
09.05.12 JOB07887  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#002
09.05.15 JOB07887  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#003
09.05.18 JOB07887  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPPX3VV.LOG#001
09.05.21 JOB07887  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPPX2VV.LOG#001
09.05.24 JOB07887  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPPX3VV.LOG#002
09.05.27 JOB07887  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPIX2VV.LOG#001
```

Example 2: the number of 99 reports per program

If RTIS is entered with some higher value in the range 0-250. For example:

```
RTIS 99
```

The response is sent to console and to the CICS job log:

```
The maximum number of RTI logs per program is 99
```

The below fragment of the CICS job log shows an example of the RTI report files generation for 99 reports per program:

```
16.53.12 JOB05312  +BOZR008W The RTI number of logs limit has been set to 099
16.53.15 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#001
16.53.18 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#002
16.53.21 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#003
16.53.24 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#004
16.53.27 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPSX6VV.LOG#001
16.53.30 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#005
16.53.33 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#006
16.53.36 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#007
16.53.39 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPIT2VV.LOG#001
16.53.43 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#008
16.53.46 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#009
16.53.49 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#010
16.53.52 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#011
16.53.55 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#012
…
```

As you can see from Example 2, the log number of program SMPFORVV grows faster than the log number of other programs, which indicates SMPFORVV executes more frequently than the other programs.

Example 3: a practical usage of zero report per program

If RTIS is set to a value of zero,

```
RTIS 0
```

then this will stop the log generation and disable RTI reports from being produced.

The below fragment of the CICS job log shows the continuation of an example of the RTI report files generation for 99 reports per program that was interrupted by `RTIS 0` and then resumed 35 seconds later by subsequent `RTIS 1`:

```
16.53.52 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#011
16.53.55 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPFORVV.LOG#012
16.53.56 JOB05312  +BOZR008W The RTI number of logs limit has been set to 000
16.54.31 JOB05312  +BOZR008W The RTI number of logs limit has been set to 001
16.54.34 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPTX1VV.LOG#001
16.54.37 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPPS3VV.LOG#001
16.54.40 JOB05312  +BOZR001I SYSPROFD: USERA1.BOZRTI.SYSPROFD.SMPIT8VV.LOG#001
```

Using RTIS, you can suppress or alter your RTI profiling activity and cover the desired time frame in your application.

Appendix E. Manual RTI rebinding instructions

It is recommended to use the RTIBIND option in order to enable modules for RTI Profiling. However, if more low-level control of the rebinding process is required then the manual steps below can be used instead.

The RTI Profiler consists of the two members BOZBXITA and BOZRIDT included in the same dataset where ABO was installed.

- BOZBXITA: links CEEBXITA and the related profiling routines to the main program of the application. This step enables the starting and stopping of the profiling as well as the monitoring and managing of the buffers for the RTI Profiler during the program execution.
- BOZRIDT: processes the RTI Profiler buffer data and generates the text file report.

To use the RTI Profiler, follow these steps:

1. In the link-edit step, rebind your existing program to include BOZBXITA
2. In the execution step, specify the location of the dataset that will hold the profiling results

In the steps and JCL examples below, hlqboz.SBOZMOD1 is the installation location chosen for ABO.

Step 1. Rebind your existing program to include BOZBXITA

The first step is to rebind your existing program to include BOZBXITA so the RTI Profiler is enabled when running your program.

To perform this rebind, modify the link-edit step of the program containing the main entry point to your application:

- Add hlqboz.SBOZMOD1 to the link-edit step as SYSLIB
- Include BOZBXITA as additional input to link-edit step

Below is a JCL example for this step:

```
//LKED EXEC PGM=IEWL,PARM='options' <- original link options
//SYSLIB DD DISP=SHR,DSN=hlqboz.SBOZMOD1 <-- add hlqcee.BOZ210.SBOZMOD1 as SYSLIB
//       DD DSN=hlqcee.SCEELKED,DISP=SHR
//       DD DSN=hlqcee.SCEELKEX,DISP=SHR
//LOAD DD DISP=SHR,DSN=&LOAD
//SYSLMOD DD DISP=SHR,DSN=&SYSLMOD(pgmname)
//SYSPRINT DD SYSOUT=*
//SYSLIN DD *
INCLUDE LOAD(pgmname)
INCLUDE SYSLIB(BOZBXITA) <-- add INCLUDE for BOZBXITA
NAME pgmname(R)
```

For step 1, the same set of the link options as the original module must be used when rebinding for RTI Profiler usage. To verify the same set of the link options were used, you can use AMBLIST on the original and rebound modules. The most likely mismatch for COBOL programs is inadvertently changing AMODE from 24 to 31 when rebinding. Below is a JCL example that shows how to set the link options for AMODE=24.

```
//LKED EXEC PGM=IEWL,PARM='LIST,MAP,AMODE=24'
```

Step 2. Specify the location of the dataset that will hold the profiling results

See Capturing profiling results during program Execution.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Corporation
J74/G4
555 Bailey Avenue
San Jose, CA 95141-1099
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES
THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF NON-INFRINGEMENT,
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors.

Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information”.

List of resources

IBM Automatic Binary Optimizer for z/OS publications

You can find the latest and most complete information about the IBM Automatic Optimizer for z/OS APARs and PTFs on the fix list and new features page.

You can find the following publications in the IBM Automatic Binary Optimizer for z/OS library:

- *User's Guide*, SC27-9587-01
- *Program Directory*, GI13-4513-05

Related publications

z/OS publications

You can find the following publications in the z/OS Internet Library.

- *Initialization and Tuning Reference*, SA23-1380, contains information about the parmlib member IEFOPZxx.
- *Program Management: Advanced Facilities*, SA23-1392, contains information on binder API return codes and reason codes.
- *System Management Facilities (SMF)*, SA38-0667, contains information about the SMF record 90 subtype 38, which captures the IEFOPZ configuration.
- *System Messages, Volume 8*, SA38-0675, contains information about the messages.

Enterprise COBOL for z/OS publications

You can find the following publications in the Enterprise COBOL for z/OS library.

- *Customization Guide*, SC27-8712, contains information that helps you plan for and customize Enterprise COBOL under z/OS.
- *Language Reference*, SC27-8713, contains information about COBOL language and references needed to write a program for an IBM COBOL compiler.
- *Programming Guide*, SC27-8714, contains information and examples that help you write, compile, and debug programs and classes.
- *Migration Guide*, GC27-8715, contains information that helps you move to the latest version of IBM Enterprise COBOL.
- *Performance Tuning Guide*, SC27-9202, identifies key performance benefits and tuning considerations when using IBM Enterprise COBOL for z/OS.
- *Messages and Codes*, SC27-4648, helps you understand compiler and preprocessor messages.

Application Delivery Foundation for z/OS publications

You can find the following publication in the IBM Documentation.

- *IBM Application Performance Analyzer for z/OS User's Guide*, SC27-8403, contains information that helps identify system constraints and improve application performance.
- *IBM Developer for z/OS documentation* (online version only), contains information about the Integrated Development Environment (IDE), designed to increase developer productivity.
- *IBM Fault Analyzer for z/OS User's Guide and Reference*, SC19-4116, contains information about analyzing and fixing application and system failures.

Product Number: 5697-AB2

SC27-9587-01