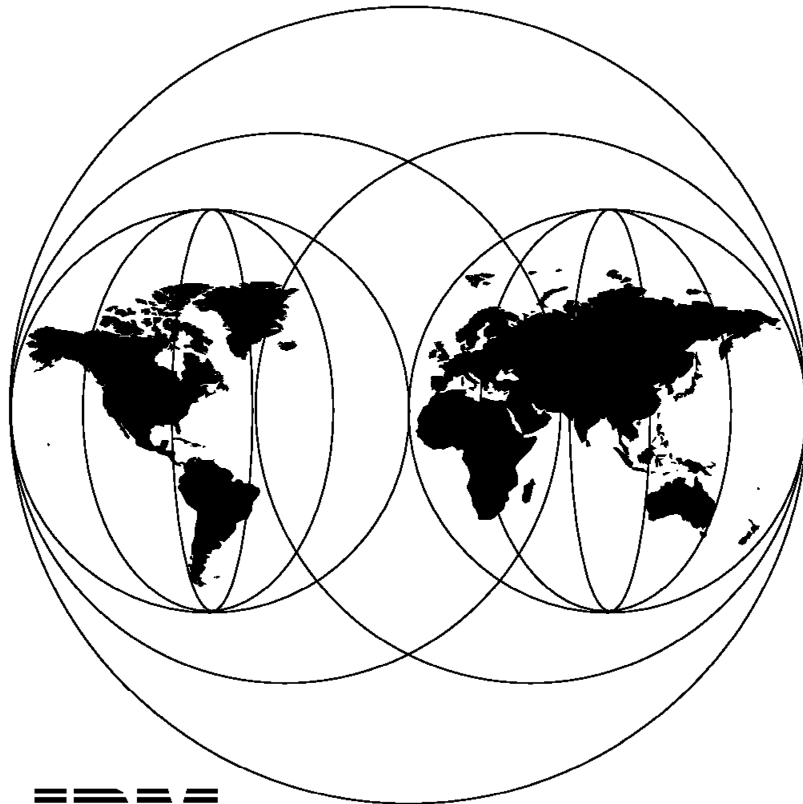
International Technical Support Organization

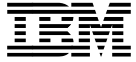
GG24-4271-00

31-bit Addressing in PL/I for VSE
Getting Started

January 1995



International Technical Support Organization
Boeblingen Center



International Technical Support Organization

GG24-4271-00

31-bit Addressing in PL/I for VSE Getting Started

January 1995

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Special Notices” on page xiii.

First Edition (January 1995)

This edition applies to Version 1, Release 1 of PL/I for VSE (5686-069) and Version 1, Release 1 of Language Environment/VSE (5686-067) for use with VSE/ESA version 2.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader′s feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. 3222 Building 71032-02
Postfach 1380
71032 Boeblingen, Germany

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1995. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Abstract

This document is unique in its detailed coverage of 31-bit addressing considerations in a system with LE for VSE and PL/I for VSE installed. It provides detailed guidance on setting up and running a system to provide effective initial exploitation of 31-bit addressing for the PL/I user in a VSE/ESA environment.

This document was written for systems programmers and administrators of installations that plan to migrate their applications from DOS PL/I to PL/I for VSE in order to gain the advantages of 31-bit exploitation both in batch and on-line. Some knowledge of CICS/VSE, the PL/I language, and 31-bit principles is assumed.

(83 pages)

© Copyright IBM Corp. 1995

Contents

Abstract . . . iii

Special Notices . . . xiii

Preface . . . xv
How This Document is Organized . . . xv
Related Publications . . . xvi
International Technical Support Organization Publications . . . xvi
Acknowledgments . . . xvii

Chapter 1. Introduction . . . 1
1.1 Why the Interest? . . . 1
1.2 Some Definitions . . . 2
1.2.1 The LE/VSE Program Management Model . . . 2
1.2.2 LE/VSE Storage Management Model . . . 4
1.3 Pricing Considerations . . . 5

Chapter 2. SVA Eligibility and Residence . . . 7
2.1 Available SVA Space . . . 7
2.2 SVA Eligibility . . . 7
2.3 Changes to Implement the Recommended SVA Residence . . . 8
2.3.1 The Base Layout of the SVA in Environment B . . . 8
2.3.2 Enlarged SVAs . . . 9
2.3.3 SVA After Loading Recommended LE 24-bit Components . . . 10
2.3.4 SVA After Loading All Recommended LE Components . . . 11
2.3.5 SVA After Loading All Recommended LE Components and PL/I 31-bit Components . . . 12
2.3.6 SVA After Loading All Recommended LE and PL/I Components . . . 14
2.4 Changes to Make all Eligible Modules SVA Resident . . . 15
2.4.1 SVA After Loading All LE 24-bit SVA-eligible Phases . . . 15
2.4.2 SVA After Loading All LE SVA-eligible Phases . . . 17
2.4.3 SVA After Loading All PL/I 24-bit and All LE SVA-eligible Phases . . . 19
2.4.4 After Loading All PL/I and LE SVA-eligible Phases . . . 20
2.5 Summary . . . 21
2.5.1 How About Some Simple Rules? . . . 22

Chapter 3. Run-time Options and Their Effects . . . 23
3.1 Program Residence . . . 23
3.1.1 Batch Considerations . . . 23
3.1.2 CICS Considerations . . . 24
3.2 Stack Residence . . . 24
3.2.1 Stack Storage Overview . . . 24
3.2.2 Tuning Stack Storage . . . 24
3.2.3 PL/I Usage of Stack Storage . . . 24
3.2.4 STACK Run-time Option . . . 24
3.3 Heap Residence . . . 25
3.3.1 Heap Storage Overview . . . 25
3.3.2 Tuning Heap Storage . . . 25
3.3.3 PL/I Usage of Heap Storage . . . 26
3.3.4 HEAP Run-time Option . . . 26
3.3.5 Increment Rounding . . . 27

© Copyright IBM Corp. 1995

3.4 Effects of Other Run-time Options 27
3.4.1 STORAGE Run-time Option 27
3.4.2 RPTOPTS Run-time Option 27
3.4.3 RPTSTG Run-time Option 28
3.5 ALL31 Run-time Option 30
3.6 PL/I VSE Resource Usage Compared with DOS PL/I 31
3.6.1 Executable Modules 31
3.6.2 Object Modules 31
3.6.3 Virtual Storage Use 31

Chapter 4. Interfacing with Other Subsystems 35
4.1 CICS/VSE 35
4.2 DL/I DOS/VS 35
4.3 DB2 for VM and VSE (SQL/DS) 36
4.4 IBM MQSeries for VSE/ESA 36

Chapter 5. How to Determine Compile-time and Run-time Options 37
5.1 Minimum Storage Requirements for PL/I VSE Compiler 37
5.2 Defining Compile Time Options 37
5.2.1 Customizing Installation Default Compile-time Options 37
5.2.2 Specifying Compile-time Options in JCL 39
5.2.3 Specifying Compile-time Options in %PROCESS or *PROCESS Statement 39
5.2.4 Compile-time Option Effects on Storage Requirements and Run Time 39
5.2.5 CICS Compile-time Considerations 40
5.3 How to Decide on your Compile-time Options 41
5.4 Defining Run-time Options 41
5.4.1 Defining Run-time Options Globally 41
5.5 Defining Run-time Options by Application 43
5.5.1 Batch Considerations 44
5.5.2 CICS Considerations 46
5.6 How to Decide on your Storage Related Run-time Options 48
5.6.1 Common Run-time Storage Usage Considerations 48
5.6.2 Using the STORAGE Compile-time Option 49
5.6.3 Using RPTSTG Run-time Option 49
5.6.4 Recommended Initial and Increment Values 50

Chapter 6. A Check-list for Initial Implementation 51
6.1 The Shared Virtual Areas 51
6.2 Run-time Options 51
6.3 Compilation 52
6.4 Differences from DOS PL/I 52

Appendix A. Performance Considerations 53
A.1 Compile Time Optimization 53
A.1.1 VDISK Usage 54
A.2 The Global Optimization Features 54
A.2.1 Limitations 55
A.3 Storage Efficiency 55
A.4 Performance Tuning Checklist 55
A.4.1 Obtaining the Listings Needed for Tuning 56
A.4.2 Selection of Compile-time Options 56
A.4.3 Tuning the Run-time Storage Use 57
A.4.4 Analysis of Compiler Messages Affecting Performance 57
A.4.5 Analysis of the Attributes of Procedures and Begin Blocks 58

A.4.6 Tuning Statement Prefixes and Labels 59
A.4.7 Tuning Expressions 59
A.4.8 Tuning Data Types and Data Structures 62
A.4.9 Tuning the Type and Use of Data Files 64
A.4.10 Examining Program's Run-time Behavior 65

Appendix B. Full Text of SDL Load Jobs 67
B.1 Job to Load All SVA-Eligible 24-bit PL/I Modules 67
B.2 Job to Load All SVA-Eligible 31-bit LE Modules 69

Glossary 73

List of Abbreviations 79

Index 81

Figures

1. LE/VSE Resource Ownership . . . 4
2. Price Comparison . . . 6
3. Default Shared Area Display . . . 8
4. Display of the Increased Shared Area . . . 9
5. Job to Load LE into 24-bit SVA . . . 10
6. Display of the SVA after Loading the Recommended LE 24-bit Code . . 10
7. Job to Load LE Components into 31-bit SVA . . . 11
8. Display of the SVA after Loading LE 31-bit Code . . . 11
9. Job to Load Recommended PL/I Components into 31-bit SVA . . . 12
10. Display of the SVA after Loading PL/I 31-bit Code . . . 13
11. Job to Load Recommended PL/I Components into 24-bit SVA . . . 14
12. Display of the SVA after Loading LE 24-bit Code . . . 14
13. Job to Load all Eligible LE Components into 24-bit SVA . . . 15
14. Display of the SVA after Loading all LE 24-bit Phases . . . 16
15. List of all Eligible LE Components for the 31-bit SVA . . . 17
16. Display of the SVA after Loading all LE 24-bit Phases . . . 18
17. List of all Eligible PL/I Components for the 24-bit SVA . . . 19
18. Display of the SVA after Loading all PL/I 31-bit and all LE Phases . . . 19
19. Job to Load all Eligible LE Components into 31-bit SVA . . . 20
20. Display of the SVA after Loading all LE and PL/I Phases . . . 21
21. Example of RPTOPTS Run-time Option Report . . . 28
22. Example of RPTSTG Report . . . 29
23. Sample JCL to Assemble Installation Default Options . . . 38
24. Sample JCL to Link Edit Installation Default Options Phase . . . 39
25. JCL to Translate and Compile CICS Sample Program DFH$PMNU . . . 40
26. Sample CEEDOPT Source Program . . . 42
27. Sample CEECOPT Source Program . . . 43
28. Sample CEEWUOPT Source Program . . . 45
29. Run-time Options Specified in JCL . . . 46
30. Run-time Options Specified in PL/I Source Program . . . 46
31. Sample CICS Compilation Job with User Run-time Options Included . . 47
32. Run-time Options Specified in PL/I Source Program . . . 48
33. Storage Requirements Table for Program TEST . . . 49
34. Storage Requirements Table for Sample Program IEL1ESO . . . 49
35. A Comparison of Performance Options . . . 61
36. A Structure Optimization Example . . . 62
37. The Effect of Data Attributes on Performance . . . 62
38. Default Attributes for Various Data Types . . . 63

© Copyright IBM Corp. 1995

Tables

1. SVA Usage for Recommended Phases 21
2. SVA Usage for All Eligible Phases 21
3. Difference Between Space Requirement of Recommended and Eligible Phases 22
4. ALL31, AMODE and RMODE vs HEAP and STACK Parameters ANY and BELOW 30
5. Storage Use DOS PL/I vs PL/I VSE without and with Phases in SVA . . 32
6. Storage Use DOS PL/I vs PL/I VSE with Phases in SVA 33
7. Formats for Specifying Run-time Options and Program Arguments . . . 45
8. Recommended Run-time Storage Values 50
9. Some Compiler Warning Messages Related to Tuning 58

© Copyright IBM Corp. 1995

Special Notices

This publication is intended to help systems programmers and system administrators set up their systems for an efficient and successful exploitation of the 31-bit capabilities of LE for VSE and PL/I for VSE. The information in this publication is not intended as the specification of any programming interfaces that are provided by Version 1, Release 1 of PL/I for VSE (5686-069) and Version 1, Release 1 of Language Environment/VSE (5686-067). See the PUBLICATIONS section of the IBM Programming Announcement for Version 1, Release 1 of PL/I for VSE (5686-069) and Version 1, Release 1 of Language Environment/VSE (5686-067) for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of

© Copyright IBM Corp. 1995

including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

CICS	CICS/VSE
DB2	IBM
MQSeries	VSE/ESA

The following terms are trademarks of other companies:

DOS	Microsoft Corporation
Microsoft	Microsoft Corporation

Preface

This document is unique in its detailed coverage of 31-bit addressing considerations in a system with LE for VSE and PL/I for VSE installed. It provides detailed guidance on setting up and running a system to provide effective initial exploitation of 31-bit addressing for the PL/I user in a VSE/ESA environment.

This document is intended for systems programmers and administrators of installations that plan to migrate their applications from DOS PL/I to PL/I for VSE in order to gain the advantages of 31-bit exploitation both in batch and on-line. Some knowledge of CICS/VSE, the PL/I language, and 31-bit principles is assumed.

How This Document is Organized

The document is organized as follows:

- Chapter 1, “Introduction”
- Chapter 2, “SVA Eligibility and Residence”

  This provides details of the programs which are eligible for SVA residence, and gives recommendations on which should in fact be loaded in the SVA.
- Chapter 3, “Run-time Options and Their Effects”

  This chapter describes the various storage-related run-time options.
- Chapter 4, “Interfacing with Other Subsystems”

  This chapter describes other sub-systems and their ability to work with PL/I for VSE programs.
- Chapter 5, “How to Determine Compile-time and Run-time Options”

  This chapter describes how to determine the correct run-time options for an application.
- Chapter 6, “A Check-list for Initial Implementation”

  This chapter provides a simple check-list for getting started with 31-bit exploitation in PL/I for VSE.
- Appendix A, “Performance Considerations”

  This appendix describes performance considerations for PL/I for VSE and LE for VSE.
- Appendix B, “Full Text of SDL Load Jobs”

  This appendix provides in full two job-streams to load programs into the SVA. This has been done, rather than putting them in the main text, to improve readability.

© Copyright IBM Corp. 1995

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

- *VSE/ESA V2.1.0 GIM Kit*, GBOF-2300-00

The following manuals may be *unavailable* at the time of printing this document. However, they can be *ordered* at General Availability of the respective product.

- *CICS/VSE 2.3 Release Guide*, GC33-0700-02
- *CICS/VSE 2.3 System Definition and Operations Guide*, SC33-0706-02
- *IBM Language Environment for VSE/ESA Diagnosis Guide*, SC26-8060
- *IBM Language Environment for VSE/ESA Licensed Program Specifications*, GC26-8061
- *IBM Language Environment for VSE/ESA Fact Sheet*, GC26-8062
- *IBM Language Environment for VSE/ESA Concepts Guide*, GC26-8063
- *IBM Language Environment for VSE/ESA Installation and Customization Guide*, SC26-8064
- *IBM Language Environment for VSE/ESA Programming Guide*, SC26-8065
- *IBM Language Environment for VSE/ESA Debugging Guide and Run-Time Messages*, C26-8066
- *IBM Language Environment for VSE/ESA Reference Summary*, SX26-3835
- *IBM PL/I for VSE/ESA Fact Sheet*, GC26-8052
- *IBM PL/I for VSE/ESA Programming Guide*, SC26-8053
- *IBM PL/I for VSE/ESA Language Reference*, SC26-8054
- *IBM PL/I for VSE/ESA Licensed Program Specifications*, GC26-8055
- *IBM PL/I for VSE/ESA Migration Guide*, SC26-8056
- *IBM PL/I for VSE/ESA Installation and Customization*, SC26-8057
- *IBM PL/I for VSE/ESA Diagnosis Guide*, SC26-8058
- *IBM PL/I for VSE/ESA Compile-Time Messages and Codes*, SC26-8059
- *IBM PL/I for VSE/ESA Reference Summary*, SX26-3836

International Technical Support Organization Publications

- *VSE/ESA 1.3 Migration Guide*, GG24-4025
- *VSE/ESA 1.3 Using the 31-bit Addressing Facility*, GG24-4191
- *CICS/VSE V2R2 Data in Memory and Virtual Storage Usage*, GG24-4185

A complete list of International Technical Support Organization publications, with a brief description of each, may be found in:

Bibliography of International Technical Support Organization Technical Bulletins, GG24-3070.

To get listings of ITSO technical bulletins (redbooks) online, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

How to Order ITSO Technical Bulletins (Redbooks)

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-284-4721. Visa and Master Cards are accepted. Outside the USA, customers should contact their IBM branch office.

Customers may order hardcopy redbooks individually or in customized sets, called GBOFs, which relate to specific functions of interest. IBM employees and customers may also order redbooks in online format on CD-ROM collections, which contain the redbooks for multiple products.

Acknowledgments

The advisor for this project was:

Peter Lockwood
International Technical Support Organization, Boeblingen Center

The authors of this document are:

Marijan Pucelj
IBM Slovenia

Peter Lockwood
ITSO Boeblingen

This publication is the result of a residency conducted at the International Technical Support Organization, Boeblingen Center.

Chapter 1. Introduction

1.1 Why the Interest?

When VSE/ESA first provided virtual storage constraint relief by making virtual storage above 16MB available to application programs, one large community of customers found that they were excluded. This group consisted of those customers whose applications are written in PL/I.

In fact, PL/I users were also victims of the fact that a high-level language tends to use more virtual storage than an Assembler language program performing similar functions. Indeed, because of its sophisticated capabilities, PL/I will tend to use more virtual storage than COBOL.

Thus the community of users who were most in need of VSCR were unable to get it, at least directly for their application programs, although they were able to benefit indirectly in CICS, from such mechanisms as the moving of VSAM buffers into 31-bit GETVIS. Now, with the announcement of Language Environment for VSE and PL/I for VSE, the benefits of using 31-bit storage for user applications are also available to PL/I users, both in batch and running as CICS applications.

This, in turn, introduces some new considerations:

1. Storage allocations for both on-line and batch programs

   In DOS PL/I the type of storage used in an application was defined in the attributes of the data definition. In PL/I for VSE, the PL/I attributes of the data are also defined in the attributes of the data definition. However these now map to LE storage types, and the location and allocation units of these LE storage types are controlled by run-time options.

2. Handling of run-time attributes

   With many new run-time options, many of which relate to storage management, the decision must be made whether these should be established by application specific options, or whether they should be handled by installation standard defaults.

3. Choice of linkage edit attributes

   Here also care must be taken not to restrict the AMODE and RMODE attributes, and therefore lose the benefits of the VSCR which is now available.

4. Services which were previously provided by PL/I itself

   It is important to understand that PL/I itself no longer provides run-time services, and that all run-time services for a batch PL/I for VSE application are now provided by LE/VSE. This is true both of CEL routines, and also of PL/I language specific routines.

© Copyright IBM Corp. 1995

1.2 Some Definitions

As a start to understanding the terminology used within LE program and storage management, some basic definitions are extracted from the LE for VSE Concepts Guide. For more detailed descriptions and discussions you should consult the LE for VSE Concepts Guide.

1.2.1 The LE/VSE Program Management Model

The LE/VSE program management model provides a framework within which an application runs. It is the foundation of all of the component models - condition handling, run-time message handling, and storage management - that comprise the LE/VSE architecture. The program management model defines the effects of programming language semantics in mixed-language applications and integrates transaction processing and multi-threading.

LE/VSE Program Management Model Terminology: Some terms used to describe the program management model are common programming terms; other terms are described differently in other languages. It is important that you understand the meaning of the terminology in an LE/VSE context as compared with other contexts. For more detailed definitions of these and other LE/VSE terms, please consult the Glossary.

General Programming Terms:

Application program A collection of one or more programs cooperating to achieve particular objectives, such as inventory control or payroll.

Environment In LE/VSE, normally a reference to the run-time environment of HLLs at the enclave level.

LE/VSE Terms and their HLL Equivalents:

Routine In LE/VSE, refers to either a procedure, function, or subroutine.

Equivalent HLL terms:

COBOL program

PL/I procedure or begin block

Thread An execution construct that consists of synchronous invocations and terminations of routines. The thread is the basic run-time path within the LE/VSE program management model, and is dispatched by the system with its own run-time stack, instruction counter, and registers. Threads may exist concurrently with other threads.

Enclave The enclave defines the scope of HLL semantics. In LE/VSE, a collection of routines, one of which is named as the main routine. The enclave contains at least one thread.

Equivalent HLL terms:

COBOL run unit

PL/I main procedure and its subroutines

Process
: The highest level of the LE/VSE program management model. A process is a collection of resources, both program code and data, and consists of at least one enclave.

Terminology for Data

Automatic data
: Data that does not persist across calls. It is allocated with the same value on entry and reentry into a routine.

External data
: Data that can be referenced by one or more routines and data areas. External data is known throughout an enclave.

Local data
: Data that is known only to the routine in which it is declared. Equivalent terms:

  COBOL WORKING-STORAGE data items

  PL/I data declared with the PL/I INTERNAL attribute

The following figure illustrates some of these functions and their interaction.

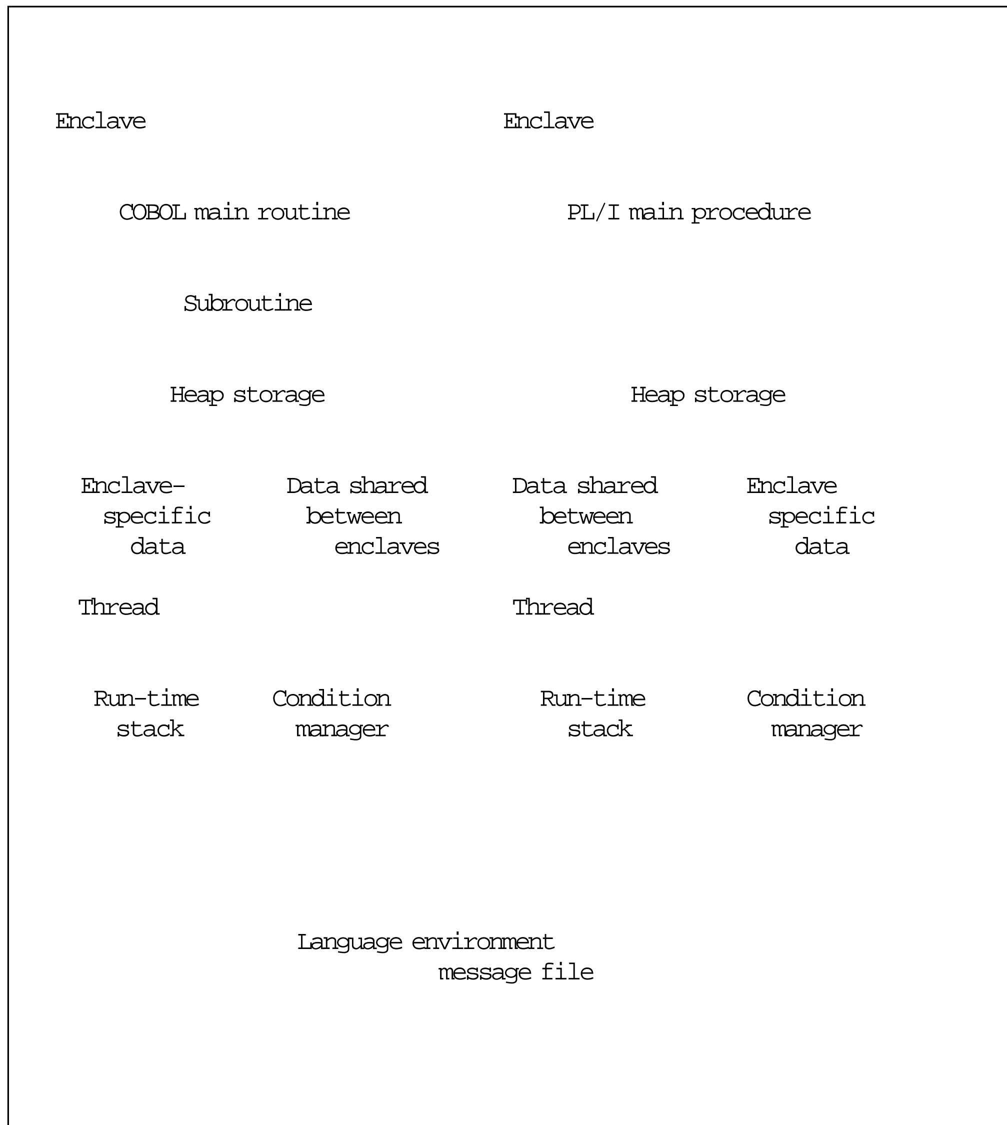


Figure 1. LE/VSE Resource Ownership

1.2.2 LE/VSE Storage Management Model

Common storage management services are provided for all LE/VSE-conforming programming languages; LE/VSE controls stack and heap storage used at run time. It allows single and mixed language applications to access a central set of storage management facilities, and offers a multiple-heap storage model to languages that do not now provide one. The common storage model removes the need for each language to maintain a unique storage manager, and avoids the incompatibilities between different storage mechanisms.

Storage Management Terminology:

Stack An area of storage in which stack frames are allocated.

Stack frame An area of storage that is allocated when a routine runs, and that represents the history of execution of that routine.

Heap — An area of storage used for allocation of storage whose lifetime is not related to the execution of the current routine. The heap consists of the initial heap segment and zero or more increments. Heap storage contains storage acquired by the ALLOCATE statement in PL/I.

Heap element — A contiguous area of storage allocated by a call to the CEEGTST service. Heap elements are always allocated within a single heap segment.

Heap increment — Additional heap segments allocated when the initial heap segment does not have enough free storage to satisfy a request for heap storage.

Heap segment — A contiguous area of storage obtained directly from the operating system.

1.3 Pricing Considerations

Migrations from DOS/VS COBOL to VS COBOL II were sometimes delayed because VS COBOL II attracts both an Initial License Charge, and a significantly higher Monthly License Charge than does DOS/VS COBOL II. This has been addressed with the combination of COBOL for VSE and LE/VSE which are, on many processors, cheaper than VS COBOL II, and may in fact be cheaper than DOS/VS COBOL. Similarly there are cases where the combination of PL/I for VSE and LE/VSE is cheaper than DOS PL/I with its Transient and Resident Libraries. The chart in Figure 2 was plotted using prices derived from the UK Software Prices Database in November 1994.

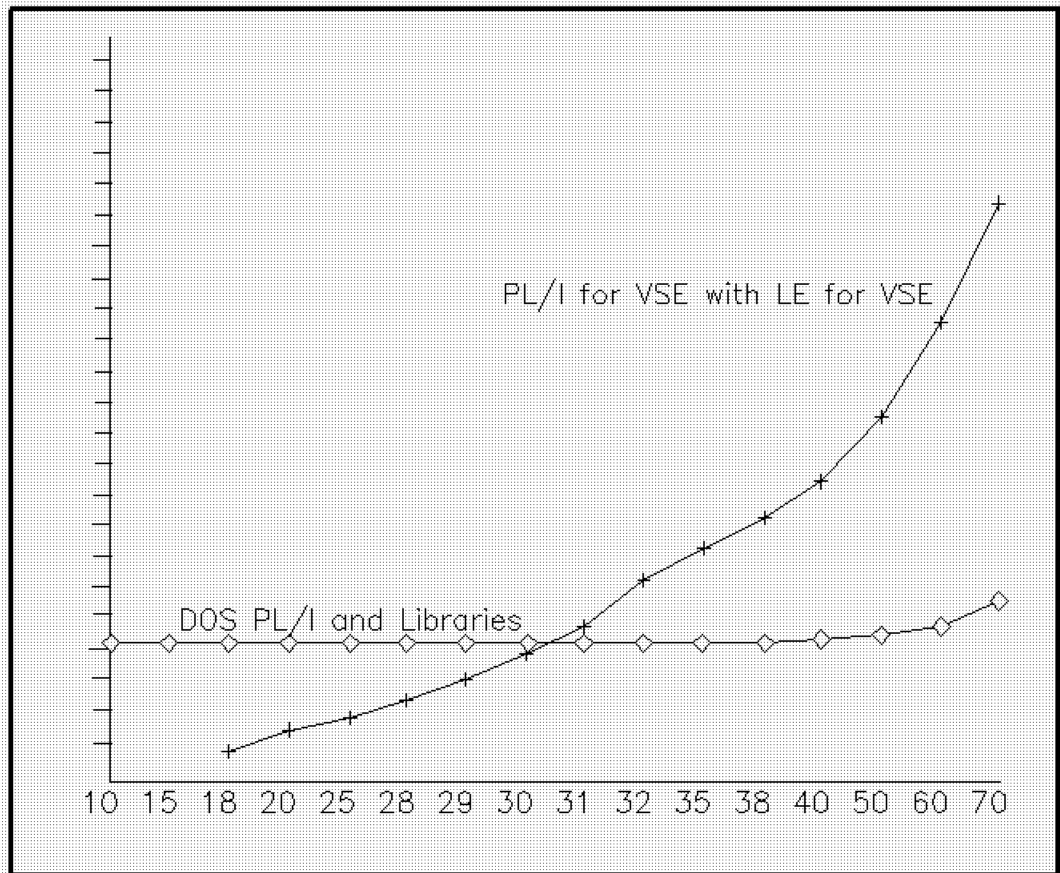


Figure 2. Price Comparison. The exact price comparison may vary slightly from country to country. You should obtain a current, local quotation.

Chapter 2. SVA Eligibility and Residence

This chapter describes the investigation into SVA requirements of the 24-bit and 31-bit SVA-eligible phases in the LE general and PL/I language specific sub-libraries. As a result of the investigation some recommendations are made for initial SVA placement of these modules.

2.1 Available SVA Space

We installed the system with the supplied Environment B, which has 4MB of shared space. There was only 201K available in the 24-bit program area in the SVA, although there was 365K available in the 31-bit program area.

Warning

The actual amounts will depend on modification level, maintenance, other product use of the SVA, pre-defined environment chosen, and other factors.

The layout is shown in Figure 3 on page 8. An attempt to load a load-list for HLASM failed because the load-list did not exist.

Most customers will also install other products, from IBM or other vendors which either need modules to be loaded in the SVA or where there is a strong recommendation that modules should be loaded into the SVA. Indeed one such candidate is CICS/VSE itself, since although some phases are pre-loaded in the SVA, there are many others which can benefit, especially since the majority of users have more than one CICS partition.

We must also consider that if modules in the SVA are altered, the new version will also be loaded in the SVA, but into previously available space - no space reclamation is done. As a result some spare space is needed for maintenance, especially in a system when high availability or continuous operation is needed.

Where extra phases need to be loaded in the SVA, it may also be necessary to increase the number of SDL entries. We did so by a large enough margin to allow all the LE generic and PL/I specific run-time modules to be loaded in the SVA.

Since the 24-bit SVA minimum increase was going to take our system above the next megabyte boundary, we ensured that all of the megabyte increment was allocated, by increasing PSIZE. The PSIZE operand for 31-bit SVA was also increased by 0.5MB.

2.2 SVA Eligibility

At this stage we considered only the specific requirements of LE for VSE and PL/I for VSE.

SVA eligibility of modules of both products are documented in

PL/I Compiler This was identified by using the VSE librarian

LE General Services LE/VSE Installation and Customization

LE language-specific PL/I components LE/VSE Installation and Customization

The compiler modules may benefit from SVA residence, but this will most noticeably be the case in an installation which mainly does development. For most customers the maximum benefit will be obtained from making run-time services SVA resident. The manual we have cited breaks down the modules concerned into those whose residence is recommended, and those others which are SVA-eligible. In addition, the functions performed and the residence and addressing mode of each one is listed, allowing the user to make a more precise decision on which to use.

We measured the impact both of making all SVA eligible modules resident, and also of concentrating on the recommended modules. We made no measurements of the compiler modules.

2.3 Changes to Implement the Recommended SVA Residence

This section describes the effect of making different groups of components SVA resident.

2.3.1 The Base Layout of the SVA in Environment B

This represents base environment B on an early system. No optional CICS components have been loaded, nor have LE or PL/I components been loaded.

```
IESADMDSV1              SHARED VIRTUAL AREA LAYOUT

---------------------------------------------------------------------------
¢024FFFFF¢X SYSTEM GETVIS(31) USED:   (HWM: 1232K ) |  436K |       |
¢02492000¢X                   FREE:                 |    0B |  436K |
--------------------------------------------------------------------|  3072K
            PROGRAM AREA(31)  USED:                 | 2267K |       |
¢02200000¢X                   AVAILABLE:            |  365K | 2632K |
===========================================================================
¢003FFFFF¢X V-POOL                                  |  128K |       |
¢003C5000¢X SYSTEM LABEL AREA (SLA)                 |  108K |  236K |
--------------------------------------------------------------------|
            SYSTEM GETVIS(24) USED:   (HWM:  796K ) |  780K |       |
¢00253000¢X                   FREE:                 |  700K | 1480K |
--------------------------------------------------------------------|  3248K
            PROGRAM AREA(24)  USED:                 | 1299K |       |
¢000DC000¢X                   AVAILABLE:            |  201K |       |
----------------------------------------------------------| 1532K |
            SYSTEM DIRECTORY LIST (SDL)             |   32K |       |
¢000D4000¢X                                         |       |       |
---------------------------------------------------------------------------
PF1=HELP      2=REFRESH   3=END       4=RETURN               6=PARTITION
```

Figure 3. Default Shared Area Display. Note the highlighted values for Program Area(24) and Program Area(31).

2.3.2 Enlarged SVAs

We selected a PSIZE which would allow all eligible LE and PL/I phases to be loaded in the SVA, consistent with ending the shared area exactly on a megabyte boundary. After increasing the PSIZE value to (1256K,2312K) the following is the SVA display:

```
IESADMDSV1               SHARED VIRTUAL AREA LAYOUT

-------------------------------------------------------------------------------
¢026FFFFF¢X SYSTEM GETVIS(31) USED:   (HWM: 1232K ) |   660K |        |
¢02612000¢X                   FREE:                 |   292K |   952K |
----------------------------------------------------------------------|  4096K
            PROGRAM AREA(31)  USED:                 |  2267K |        |
¢02300000¢X                   AVAILABLE:            |   877K |  3144K |
===============================================================================
¢004FFFFF¢X V-POOL                                  |   128K |        |
¢004C5000¢X SYSTEM LABEL AREA (SLA)                 |   108K |   236K |
----------------------------------------------------------------------|
            SYSTEM GETVIS(24) USED:   (HWM:  568K ) |   552K |        |
¢00353000¢X                   FREE:                 |   928K |  1480K |
----------------------------------------------------------------------|  4268K
            PROGRAM AREA(24)  USED:                 |  1299K |        |
¢000E2000¢X                   AVAILABLE:            |  1201K |        |
-------------------------------------------------------------|  2552K |
            SYSTEM DIRECTORY LIST (SDL)             |    52K |        |
¢000D5000¢X                                         |        |        |
-------------------------------------------------------------------------------
PF1=HELP      2=REFRESH   3=END       4=RETURN                6=PARTITION
```

Figure 4. Display of the Increased Shared Area. Note the highlighted values for Program Area(24) and Program Area(31). The shared area now occupies 5MB.

2.3.3 SVA After Loading Recommended LE 24-bit Components

The job SDLLE24R shown in Figure 5 produced the result in Figure 6.

The modules in this list are recommended because they are high usage. Some of them are also very large. CEEEV010 is big enough that you cannot afford to have it loaded dynamically. Thus it should be loaded in the SVA as a matter of course.

CEECCICS, although defined in the CSD, is part of the CICS nucleus, and this is also very large and should be in the SVA if more than one CICS partition is run. If there is only one CICS with LE, then you may wish to omit this from the SVA load-list.

Batch components should be in the SVA if multiple batch LE partitions will be used.

```
* $$ JOB JNM=SDLLE24R,DISP=D,CLASS=0
// JOB LOAD LE 24-BIT RECOMMENDED MODULES INTO SVA
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.SCEECICS)
SET SDL
CEEBINIT,SVA
CEECCICS,SVA
CEEEV010,SVA
CEEPIPI,SVA
CEEPLPKD,SVA
/*
/&
* $$ EOJ
```

Figure 5. Job to Load LE into 24-bit SVA

```
IESADMDSV1                SHARED VIRTUAL AREA LAYOUT

------------------------------------------------------------------------------
¢026FFFFF¢X SYSTEM GETVIS(31) USED:    (HWM: 1340K ) |   660K |        |
¢02612000¢X                   FREE:                  |   292K |   952K |
---------------------------------------------------------------------|  4096K
            PROGRAM AREA(31)  USED:                  |  2267K |        |
¢02300000¢X                   AVAILABLE:             |   877K |  3144K |
==============================================================================
¢004FFFFF¢X V-POOL                                   |   128K |        |
¢004C5000¢X SYSTEM LABEL AREA (SLA)                  |   108K |   236K |
---------------------------------------------------------------------|
            SYSTEM GETVIS(24) USED:    (HWM:  680K ) |   556K |        |
¢00353000¢X                   FREE:                  |   924K |  1480K |
---------------------------------------------------------------------|  4268K
            PROGRAM AREA(24)  USED:                  |  2261K |        |
¢000E2000¢X                   AVAILABLE:             |   239K |        |
------------------------------------------------------------|  2552K |
            SYSTEM DIRECTORY LIST (SDL)              |    52K |        |
¢000D5000¢X                                          |        |        |
------------------------------------------------------------------------------
PF1=HELP      2=REFRESH   3=END      4=RETURN             6=PARTITION
```

Figure 6. Display of the SVA after Loading the Recommended LE 24-bit Code. Note the highlighted value for Program Area(24). Program Area(31) has not changed.

2.3.4 SVA After Loading All Recommended LE Components

The jobs SDLLE24R and SDLLE31R shown in Figure 5 and in Figure 7 produced the result in Figure 8.

Since 31-bit storage is basically unconstrained for most users, the recommended 31-bit (high-use) components should be loaded as a matter of course.

```
* $$ JOB JNM=SDLLE31R,DISP=D,CLASS=0
// JOB LOAD LE 31-BIT RECOMMENDED MODULES INTO SVA
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.SCEECICS)
SET SDL
CEECOPP,SVA
CEEPLPKA,SVA
CEEQMATH,SVA
CEEYDTS,SVA
/*
/&
* $$ EOJ
```

Figure 7. Job to Load LE Components into 31-bit SVA

```
IESADMDSV1               SHARED VIRTUAL AREA LAYOUT

------------------------------------------------------------------------------
¢026FFFFF¢X SYSTEM GETVIS(31) USED:   (HWM: 1340K ) |   660K |        |
¢02612000¢X                   FREE:                 |   292K |   952K |
----------------------------------------------------------------------|  4096K
            PROGRAM AREA(31)  USED:                 |  2753K |        |
¢02300000¢X                   AVAILABLE:            |   391K |  3144K |
==============================================================================
¢004FFFFF¢X V-POOL                                  |   128K |        |
¢004C5000¢X SYSTEM LABEL AREA (SLA)                 |   108K |   236K |
----------------------------------------------------------------------|
            SYSTEM GETVIS(24) USED:   (HWM:  680K ) |   560K |        |
¢00353000¢X                   FREE:                 |   920K |  1480K |
----------------------------------------------------------------------|  4268K
            PROGRAM AREA(24)  USED:                 |  2261K |        |
¢000E2000¢X                   AVAILABLE:            |   239K |        |
-------------------------------------------------------------|  2552K |
            SYSTEM DIRECTORY LIST (SDL)             |    52K |        |
¢000D5000¢X                                         |        |        |
------------------------------------------------------------------------------
PF1=HELP      2=REFRESH   3=END       4=RETURN                6=PARTITION
```

Figure 8. Display of the SVA after Loading LE 31-bit Code. Note the highlighted value for Program Area(31).

2.3.5 SVA After Loading All Recommended LE Components and PL/I 31-bit Components

The jobs SDLLE24R, SDLLE31R, and SDLPL31R shown in Figure 5, Figure 7, and Figure 9 produced the result in Figure 10.

Since 31-bit SVA is a cheap resource, the 31-bit recommended phases should be loaded as a matter of course.

```
* $$ JOB JNM=SDLPLI31,DISP=D,CLASS=0
// JOB LOAD PL/I 31-BIT MODULES INTO SVA
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.SCEECICS)
SET SDL
IBMMSGT,SVA
IBMRBCGA,SVA
IBMRBCIA,SVA
IBMRBCTA,SVA
IBMRBCVA,SVA
IBMRBEIA,SVA
IBMRBGBA,SVA
IBMRBGCA,SVA
IBMRBGIA,SVA
IBMRBGVA,SVA
IBMRBMPA,SVA
IBMRDMPJ,SVA
IBMRDMPM,SVA
IBMRDMPU,SVA
IBMREOCA,SVA
IBMREOLA,SVA
IBMRJDDA,SVA
IBMRJDTA,SVA
IBMRJTTA,SVA
IBMRKDMA,SVA
IBMRPTLA,SVA
IBMRSAP,SVA
IBM9LMSA,SVA
IBM9LMSU,SVA
IBM9LM2A,SVA
IBM9LM2U,SVA
/*
/&
* $$ EOJ
```

Figure 9. Job to Load Recommended PL/I Components into 31-bit SVA

```
IESADMDSV1                SHARED VIRTUAL AREA LAYOUT

------------------------------------------------------------------------------
¢026FFFFF¢X SYSTEM GETVIS(31) USED:    (HWM: 1348K ) |   660K |        |
¢02612000¢X                   FREE:                  |   292K |   952K |
---------------------------------------------------------------------|  4096K
            PROGRAM AREA(31)  USED:                  |  2758K |        |
¢02300000¢X                   AVAILABLE:             |   386K |  3144K |
==============================================================================
¢004FFFFF¢X V-POOL                                   |   128K |        |
¢004C5000¢X SYSTEM LABEL AREA (SLA)                  |   108K |   236K |
---------------------------------------------------------------------|
            SYSTEM GETVIS(24) USED:    (HWM:  688K ) |   560K |        |
¢00353000¢X                   FREE:                  |   920K |  1480K |
---------------------------------------------------------------------|  4268K
            PROGRAM AREA(24)  USED:                  |  2261K |        |
¢000E2000¢X                   AVAILABLE:             |   239K |        |
------------------------------------------------------------|  2552K |
            SYSTEM DIRECTORY LIST (SDL)              |    52K |        |
¢000D5000¢X                                          |        |        |
------------------------------------------------------------------------------
PF1=HELP      2=REFRESH   3=END       4=RETURN                6=PARTITION
```

Figure 10. Display of the SVA after Loading PL/I 31-bit Code. Note the highlighted value for Program Area(31) which has increased again, but by a smaller amount.

2.3.6 SVA After Loading All Recommended LE and PL/I Components

The jobs SDLLE24R, SDLLE31R, SDLPL31R and SDLPL24R shown in Figure 5, Figure 7, Figure 9, and Figure 11 produced the result in Figure 12.

This loaded only a single module, but because of its characteristics, it is likely to be commonly used. If possible this should also be put in the SVA.

```
* $$ JOB JNM=SDLP24R,DISP=D,CLASS=0
// JOB LOAD PL/I 24-BIT RECOMMENDED MODULES INTO SVA
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.SCEECICS)
SET SDL
IBMRLIB1,SVA
/*
/&
* $$ EOJ
```

Figure 11. Job to Load Recommended PL/I Components into 24-bit SVA

```
IESADMDSV1              SHARED VIRTUAL AREA LAYOUT

------------------------------------------------------------------------------
¢026FFFFF¢X SYSTEM GETVIS(31) USED:   (HWM: 1348K ) |   660K |         |
¢02612000¢X                   FREE:                 |   292K |   952K  |
--------------------------------------------------------------------|  4096K
            PROGRAM AREA(31)  USED:                 |  2758K |         |
¢02300000¢X                   AVAILABLE:            |   386K |  3144K  |
==============================================================================
¢004FFFFF¢X V-POOL                                  |   128K |         |
¢004C5000¢X SYSTEM LABEL AREA (SLA)                 |   108K |   236K  |
--------------------------------------------------------------------|
            SYSTEM GETVIS(24) USED:   (HWM:  688K ) |   560K |         |
¢00353000¢X                   FREE:                 |   920K |  1480K  |
--------------------------------------------------------------------|  4268K
            PROGRAM AREA(24)  USED:                 |  2288K |         |
¢000E2000¢X                   AVAILABLE:            |   212K |         |
-----------------------------------------------------------|  2552K  |
            SYSTEM DIRECTORY LIST (SDL)             |    52K |         |
¢000D5000¢X                                         |        |         |
------------------------------------------------------------------------------
PF1=HELP      2=REFRESH   3=END       4=RETURN             6=PARTITION
```

Figure 12. Display of the SVA after Loading LE 24-bit Code. Note the highlighted values for Program Area(24) and Program Area(31). The amount of Program Area(24) has decreased further, but this now shows the net requirements for the recommended SVA resident phases for both the generic LE routines, and the PL/I language-specific routines.

2.4 Changes to Make all Eligible Modules SVA Resident

As a further step all SVA-eligible phases were made SVA resident. The different steps are shown in this section for comparison with the requirements of the recommended modules.

The base figure shown in Figure 3 on page 8 modified as in Figure 4 on page 9 is the basis for these measurements also.

2.4.1 SVA After Loading All LE 24-bit SVA-eligible Phases

The job SDLLE24 shown in Figure 13 produced the result in Figure 14.

This only leads to an increase of 5KB in the 24-bit SVA requirement compared with loading only the recommended phases, and should be considered a feasible course of action.

```
* $$ JOB JNM=SDLLE24,DISP=D,CLASS=0
// JOB LOAD LE 24-BIT MODULES INTO SVA
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.SCEECICS)
SET SDL
CEECCICS,SVA
CEEBINIT,SVA
CEEEV010,SVA
CEEPIPI,SVA
CEEPLPKD,SVA
CEEYCD0,SVA
CEEYPR0,SVA
CEEYTCIL,SVA
/*
/&
* $$ EOJ
```

Figure 13. Job to Load all Eligible LE Components into 24-bit SVA

```
IESADMDSV1              SHARED VIRTUAL AREA LAYOUT

------------------------------------------------------------------------------
¢026FFFFF¢X SYSTEM GETVIS(31) USED:   (HWM: 1244K ) |   660K |        |
¢02612000¢X                   FREE:                 |   292K |   952K |
--------------------------------------------------------------------|  4096K
            PROGRAM AREA(31)  USED:                 |  2267K |        |
¢02300000¢X                   AVAILABLE:            |   877K |  3144K |
==============================================================================
¢004FFFFF¢X V-POOL                                  |   128K |        |
¢004C5000¢X SYSTEM LABEL AREA (SLA)                 |   108K |   236K |
--------------------------------------------------------------------|
            SYSTEM GETVIS(24) USED:   (HWM:  580K ) |   560K |        |
¢00353000¢X                   FREE:                 |   920K |  1480K |
--------------------------------------------------------------------|  4268K
            PROGRAM AREA(24)  USED:                 |  2266K |        |
¢000E2000¢X                   AVAILABLE:            |   234K |        |
-----------------------------------------------------------|  2552K |
            SYSTEM DIRECTORY LIST (SDL)             |    52K |        |
¢000D5000¢X                                         |        |        |
------------------------------------------------------------------------------
PF1=HELP      2=REFRESH   3=END       4=RETURN              6=PARTITION
```

Figure 14. Display of the SVA after Loading all LE 24-bit Phases. Note the highlighted values for Program Area(24).

2.4.2 SVA After Loading All LE SVA-eligible Phases

The job SDLLE24 shown in Figure 13, and loading the list shown in Figure 15 (also shown as a complete job in B.2, “Job to Load All SVA-Eligible 31-bit LE Modules”), produced the results shown in Figure 16.

Once again, since 31-bit storage is a cheap resource, and usually unconstrained, these modules should be loaded as a matter of course.

```
CEE5ABD   CEE5ADDM  CEE5CTY   CEE5DMP   CEE5GRC   CEE5GRN   CEE5LNG
CEE5MCS   CEE5MDS   CEE5MTS   CEE5PRM   CEE5RPH   CEE5SPM   CEE5SRC
CEE5USR   CEECMI    CEECNTY   CEECOPP   CEECRHP   CEECZST   CEEDATE
CEEDATM   CEEDAYS   CEEDCOD   CEEDSHP   CEEDYWK   CEEFMDA   CEEFMDT
CEEFMTM   CEEFRST   CEEGMT    CEEGMTO   CEEGPID   CEEGQDT   CEEGTST
CEEHDLR   CEEHDLU   CEEISEC   CEEITOK   CEEKDS    CEELOCT   CEEMENU0
CEEMENU2  CEEMGET   CEEMMS    CEEMOUT   CEEMRCR   CEEMSG    CEEMUEN0
CEEMUEN2  CEEMUEN3  CEENCOD   CEEOPMF   CEEPARM   CEEPLPKA  CEEQCEN
CEEQMATH  CEEQUMF   CEERAN0   CEESCEN   CEESDABS  CEESDACS  CEESDASN
CEESDAT2  CEESDATH  CEESDATN  CEESDCOS  CEESDCSH  CEESDCTN  CEESDDIM
CEESDERC  CEESDERF  CEESDEXP  CEESDGMA  CEESDINT  CEESDLG1  CEESDLG2
CEESDLGM  CEESDLOG  CEESDMOD  CEESDNIN  CEESDNWN  CEESDSGN  CEESDSIN
CEESDSNH  CEESDSQT  CEESDTAN  CEESDTNH  CEESDXPD  CEESDXPI  CEESEABS
CEESEATH  CEESEATN  CEESECI   CEESECJG  CEESECOS  CEESECS   CEESECSH
CEESEDVD  CEESEEXP  CEESEIMG  CEESELOG  CEESEMLT  CEESESIN  CEESESNH
CEESESQT  CEESETAN  CEESETNH  CEESEXPE  CEESEXPI  CEESGL    CEESGLT
CEESIABS  CEESIDIM  CEESIMOD  CEESISGN  CEESIXPI  CEESQABS  CEESQACS
CEESQASN  CEESQAT2  CEESQATH  CEESQATN  CEESQCOS  CEESQCSH  CEESQCTN
CEESQDIM  CEESQERC  CEESQERF  CEESQEXP  CEESQINT  CEESQLG1  CEESQLG2
CEESQLOG  CEESQMOD  CEESQSGN  CEESQSIN  CEESQSNH  CEESQSQT  CEESQTAN
CEESQTNH  CEESQXP2  CEESQXPI  CEESQXPQ  CEESRABS  CEESRATH  CEESRATN
CEESRCJG  CEESRCOS  CEESRCSH  CEESRDVD  CEESREXP  CEESRIMG  CEESRLOG
CEESRMLT  CEESRSIN  CEESRSNH  CEESRSQT  CEESRTAN  CEESRTNH  CEESRXPI
CEESRXPR  CEESSABS  CEESSACS  CEESSASN  CEESSAT2  CEESSATH  CEESSATN
CEESSCOS  CEESSCSH  CEESSCTN  CEESSDIM  CEESSERC  CEESSERF  CEESSEXP
CEESSGMA  CEESSINT  CEESSLG1  CEESSLG2  CEESSLGM  CEESSLOG  CEESSMOD
CEESSNIN  CEESSNWN  CEESSSGN  CEESSSIN  CEESSSNH  CEESSSQT  CEESSTAN
CEESSTNH  CEESSXPI  CEESSXPS  CEESTABS  CEESTATH  CEESTATN  CEESTCJG
CEESTCOS  CEESTCSH  CEESTDVD  CEESTEXP  CEESTIMG  CEESTLOG  CEESTMLT
CEESTSIN  CEESTSNH  CEESTSQT  CEESTTAN  CEESTTNH  CEESTXPI  CEESTXPT
CEEURTB   CEEUTC    CEEYDTS
```

Figure 15. List of all Eligible LE Components for the 31-bit SVA

```
IESADMDSV1              SHARED VIRTUAL AREA LAYOUT

------------------------------------------------------------------------------
¢026FFFFF¢X SYSTEM GETVIS(31) USED:   (HWM: 1244K ) |   660K |        |
¢02612000¢X                   FREE:                 |   292K |   952K |
--------------------------------------------------------------------|  4096K
            PROGRAM AREA(31)  USED:                 |  2950K |        |
¢02300000¢X                   AVAILABLE:            |   194K |  3144K |
==============================================================================
¢004FFFFF¢X V-POOL                                  |   128K |        |
¢004C5000¢X SYSTEM LABEL AREA (SLA)                 |   108K |   236K |
--------------------------------------------------------------------|
            SYSTEM GETVIS(24) USED:   (HWM:  580K ) |   560K |        |
¢00353000¢X                   FREE:                 |   920K |  1480K |
--------------------------------------------------------------------|  4268K
            PROGRAM AREA(24)  USED:                 |  2266K |        |
¢000E2000¢X                   AVAILABLE:            |   234K |        |
-----------------------------------------------------------|  2552K |
            SYSTEM DIRECTORY LIST (SDL)             |    52K |        |
¢000D5000¢X                                         |        |        |
------------------------------------------------------------------------------
PF1=HELP      2=REFRESH   3=END       4=RETURN                 6=PARTITION
```

Figure 16. Display of the SVA after Loading all LE 24-bit Phases. Note the highlighted values for Program Area(24) and Program Area(31).

2.4.3 SVA After Loading All PL/I 24-bit and All LE SVA-eligible Phases

The job SDLLE24 shown in Figure 13, and loading the lists shown in Figure 15 and Figure 17 (also shown as complete jobs in B.1, "Job to Load All SVA-Eligible 24-bit PL/I Modules" on page 67 and B.2, "Job to Load All SVA-Eligible 31-bit LE Modules" on page 69), produced the results shown in Figure 18.

Here the increase over loading the recommended phases is much greater. Few installations can afford to load almost 100KB into the 24-bit SVA without being sure of the benefits. You will probably need to take care in selecting a subset which will benefit your particular use.

```
IBMRCCLA   IBMRCCRA   IBMRCOMP   IBMRKMRA   IBMRLANA   IBMRLANN
IBMRLANU   IBMRLIB1   IBMRMCTA   IBMROCAA   IBMROPAA   IBMROPEA
IBMROPZA   IBMRPDBA   IBMRRAAA   IBMRRABA   IBMRRACA   IBMRRADA
IBMRRAEA   IBMRRAFA   IBMRRAGA   IBMRRAHA   IBMRRBAA   IBMRRBBA
IBMRRBEA   IBMRRBFA   IBMRRCAA   IBMRRCBA   IBMRRCCA   IBMRRDAA
IBMRRDBA   IBMRRDCA   IBMRRDDA   IBMRREAA   IBMRRECA   IBMRREEA
IBMRREFA   IBMRRQAA   IBMRRQBA   IBMRRQCA   IBMRRQEA   IBMRRQFA
IBMRRQGA   IBMRRVAA   IBMRRVGA   IBMRRVHA   IBMRRVIA   IBMRSOFA
IBMRSOUA   IBMRSOVA   IBMRSTFA   IBMRSTIA   IBMRSTUA   IBMRSTVA
```

Figure 17. List of all Eligible PL/I Components for the 24-bit SVA

```
IESADMDSV1               SHARED VIRTUAL AREA LAYOUT

---------------------------------------------------------------------------
¢026FFFFF¢X SYSTEM GETVIS(31) USED:   (HWM: 1244K ) |   660K |        |
¢02612000¢X                   FREE:                 |   292K |   952K |
--------------------------------------------------------------------|  4096K
            PROGRAM AREA(31)  USED:                 |  2950K |        |
¢02300000¢X                   AVAILABLE:            |   194K |  3144K |
===========================================================================
¢004FFFFF¢X V-POOL                                  |   128K |        |
¢004C5000¢X SYSTEM LABEL AREA (SLA)                 |   108K |   236K |
--------------------------------------------------------------------|
            SYSTEM GETVIS(24) USED:   (HWM:  580K ) |   560K |        |
¢00353000¢X                   FREE:                 |   920K |  1480K |
--------------------------------------------------------------------|  4268K
            PROGRAM AREA(24)  USED:                 |  2375K |        |
¢000E2000¢X                   AVAILABLE:            |   125K |        |
-----------------------------------------------------------|  2552K |
            SYSTEM DIRECTORY LIST (SDL)             |    52K |        |
¢000D5000¢X                                         |        |        |
---------------------------------------------------------------------------
PF1=HELP      2=REFRESH   3=END      4=RETURN                6=PARTITION
```

Figure 18. Display of the SVA after Loading all PL/I 31-bit and all LE Phases. Note the highlighted value for Program Area(24).

2.4.4 After Loading All PL/I and LE SVA-eligible Phases

The job SDLLE24 shown in Figure 13, loading the lists shown in Figure 15 and Figure 17 (also shown as complete jobs in B.1, "Job to Load All SVA-Eligible 24-bit PL/I Modules" on page 67 and B.2, "Job to Load All SVA-Eligible 31-bit LE Modules" on page 69), and the job SDLPLI31 in Figure 19 produced the results shown in Figure 20.

```
* $$ JOB JNM=SDLPLI31,DISP=D,CLASS=0
// JOB LOAD PL/I 31-BIT MODULES INTO SVA
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.SCEECICS)
SET SDL
IBMMSGT,SVA
IBMRBCGA,SVA
IBMRBCIA,SVA
IBMRBCTA,SVA
IBMRBCVA,SVA
IBMRBEIA,SVA
IBMRBGBA,SVA
IBMRBGCA,SVA
IBMRBGIA,SVA
IBMRBGVA,SVA
IBMRBMPA,SVA
IBMRDMPJ,SVA
IBMRDMPM,SVA
IBMRDMPU,SVA
IBMREOCA,SVA
IBMREOLA,SVA
IBMRJDDA,SVA
IBMRJDTA,SVA
IBMRJTTA,SVA
IBMRKDMA,SVA
IBMRPTLA,SVA
IBMRSAP,SVA
IBM9LMSA,SVA
IBM9LMSU,SVA
IBM9LM2A,SVA
IBM9LM2U,SVA
/*
/&
* $$ EOJ
```

Figure 19. Job to Load all Eligible LE Components into 31-bit SVA

```
IESADMDSV1              SHARED VIRTUAL AREA LAYOUT

--------------------------------------------------------------------------------
¢026FFFFF¢X SYSTEM GETVIS(31) USED:   (HWM: 1248K ) |   660K |        |
¢02612000¢X                   FREE:                 |   292K |   952K |
--------------------------------------------------------------------|  4096K
            PROGRAM AREA(31)  USED:                 |  3015K |        |
¢02300000¢X                   AVAILABLE:            |   129K |  3144K |
================================================================================
¢004FFFFF¢X V-POOL                                  |   128K |        |
¢004C5000¢X SYSTEM LABEL AREA (SLA)                 |   108K |   236K |
--------------------------------------------------------------------|
            SYSTEM GETVIS(24) USED:   (HWM:  580K ) |   560K |        |
¢00353000¢X                   FREE:                 |   920K |  1480K |
--------------------------------------------------------------------|  4268K
            PROGRAM AREA(24)  USED:                 |  2375K |        |
¢000E2000¢X                   AVAILABLE:            |   125K |        |
-----------------------------------------------------------|  2552K |
            SYSTEM DIRECTORY LIST (SDL)             |    52K |        |
¢000D5000¢X                                         |        |        |
--------------------------------------------------------------------------------
PF1=HELP      2=REFRESH   3=END       4=RETURN                6=PARTITION
```

Figure 20. Display of the SVA after Loading all LE and PL/I Phases. Note the highlighted values for both Program Areas.

2.5 Summary

We must now look at the values we arrived at. Clearly the increases made are almost completely used up when all eligible phases are loaded, and we are therefore back to the original decision - can we sensibly run so little spare space in the SVAs. If in fact other optional programs with SVA eligible code are to be used you should probably limit SVA residence to the recommended modules.

Table 1. SVA Usage for Recommended Phases

Loaded	**24-bit**	**31-bit**
Recommended LE	962KB	486KB
Recommended PL/I	27KB	5KB
Total	**989KB**	**491KB**
Still available	**212KB**	**386KB**

Table 2. SVA Usage for All Eligible Phases

Loaded	**24-bit**	**31-bit**
All LE	967KB	683KB
All PL/I	109KB	65KB
Total	**1076KB**	**748KB**
Still available	**125KB**	**129KB**

Table 3. Difference Between Space Requirement of Recommended and Eligible Phases		
Loaded	**24-bit**	**31-bit**
Delta LE	5KB	197KB
Delta PL/I	82KB	60KB
Total Delta	**87KB**	**257KB**
Still available	**125KB**	**129KB**

2.5.1 How About Some Simple Rules?

These rules are for initial setup and can be refined over time:

1. All recommended LE and PL/I 24-bit phases should be in the SVA

   CICS based components may not be loadable if the CICS DSA is constrained, so these at least **MUST** be in the SVA

2. All recommended LE and PL/I 31-bit phases should be in the SVA

   31-bit storage is usually not a constrained resource

3. All eligible (but not recommended) LE 31-bit phases should be in the SVA

   31-bit storage is usually not a constrained resource

4. All eligible (but not recommended) LE 24-bit phases should be in the SVA

   These only require 5K, but you may be more selective if storage is critical

5. All eligible (but not recommended) PL/I 31-bit phases should be in the SVA

   31-bit storage is usually not a constrained resource, but these may be tuned down at leisure

6. Selected eligible (but not recommended) PL/I 24-bit phases should be in the SVA

   Careful selection of heavily used components should be carried out - since the total requirement for these is about 80K, and some are rarely used, it is best to be selective.

Chapter 3. Run-time Options and Their Effects

LE/VSE provides run-time options with which you can control certain aspects of the environment set up for your application program and its access to resources. Only those LE/VSE options are covered which have meaning for programs compiled with the PL/I VSE compiler and affect the allocation of system resources and consequently the behavior of the application programs.

The following will be addressed in this chapter:

- Program Residence
- Stack Residence
- Heap Residence
- Effect of Report Options
- ALL31 Option
- Resource Usage Compared with DOS PL/I

3.1 Program Residence

Programs compiled with the PL/I for VSE compiler can take full advantage of extended addressability. A program′s ability to use 31-bit addressing depends on the linkage editor options specified at link edit time.

```
// EXEC LNKEDT,PARM=′AMODE=addr_mode,RMODE=res_mode′
```

The AMODE (Addressing Mode) identifies the addressing mode that is expected to be in effect when the program receives control.

addr_mode can be 24, 31 or ANY. A specification of AMODE ANY implies that the final decision about 24-bit or 31-bit mode is open until the program receives control. Depending on AMODE, the processor treats virtual addresses either as 24-bit or 31-bit values.

The RMODE (Residency Mode) identifies the location in virtual storage where the program resides.

res_mode can be either 24 or ANY. Specification of RMODE ANY implies that the program can reside anywhere in storage, while RMODE 24 ensures that the program can reside only below the 16MB boundary.

3.1.1 Batch Considerations

Programs meant for batch execution should have the compile-time option SYSTEM(VSE) specified. This means that the parameter list passed to the program is in the form of a single varying character string, or there are no parameters.

© Copyright IBM Corp. 1995

3.1.2 CICS Considerations

Programs running in the CICS environment should have the option SYSTEM(CICS) specified during compilation. This option dictates that the parameter list passed to the program is in the form of a pointer or list of pointers. To gain all the benefits of extended addressability AMODE 31 and RMODE ANY are recommended.

3.2 Stack Residence

LE/VSE provides services that control *stack storage* used at run time. PL/I for VSE uses these services for all automatic internal storage needed at run time. This section provides a more detailed description of *stack storage*.

3.2.1 Stack Storage Overview

Stack storage is the storage provided by LE/VSE that is needed for routine linkage and any automatic storage. It is automatically created on entry to a routine or a block, and freed on the subsequent return. Stack storage is provided at thread initialization and is available in *user stack*, which can be allocated above or below the 16MB line. Stack storage is also available in *library stack* which is allocated below the 16MB line and is used only for library routines. Each thread has a separate and distinct stack. Stack storage is divided into large stack segments, which are further divided into smaller units called *stack frames*, which are also known as dynamic storage areas (DSAs). Stack frames are added to the user stack when a routine is entered and removed upon exit following last-in first-out (LIFO) rules.

3.2.2 Tuning Stack Storage

For best performance the initial stack segment should be large enough to satisfy all requests for stack storage. The procedure is described in 3.2.4, “STACK Run-time Option.” An initial stack segment which is too large can waste storage and degrade overall system performance, especially under CICS where storage is limited.

3.2.3 PL/I Usage of Stack Storage

PL/I automatic storage is provided from the LE/VSE user stack. When the user stack is above the 16MB line, PL/I temporaries and parameter lists also reside above 16MB. The stack frame size for an individual block is constrained to 16 MB, which means that the size for an automatic aggregate, variable or dummy argument can not exceed 16 MB.

3.2.4 STACK Run-time Option

STACK controls the allocation of a thread′s stack storage and additional stack storage, and also specifies from where storage is allocated and how it is managed.

STACK(init_size, incr_size, ANYWHERE or BELOW, KEEP or FREE)

init_size determines the size of the initial stack segment.

ANYWHERE or BELOW determines the placement of stack storage.

BELOW stack storage is acquired below the 16MB line.

ANYWHERE stack storage is acquired anywhere in storage, preferably above the 16MB line.

KEEP and FREE determines the management of stack storage.

KEEP specifies that storage allocated to stack increments is not released when the last of storage is freed.

FREE specifies that storage allocated to stack increments is released when the last of storage is freed, and is the recommended option especially in the CICS environment.

3.2.4.1 Batch Considerations

The default value for *init_size* is 512KB (specified in the CEEDOPT macro). The recommendation for init_size is the value given in the RPTSTG listing under the heading *Stack statistics - Total stack storage used (suggested initial size).* The same value is recommended also for *incr_size*. The parameters ANYWHERE and FREE should be used whenever possible.

3.2.4.2 CICS Considerations

The default value for *init_size* is 4KB (specified in the CEECOPT macro). This value can be changed according to *RPTSTG's suggested initial size*, but care should be taken if BELOW is specified for stack storage, because in that case CICS DSA storage is used. Also the ANYWHERE and FREE parameters will improve performance.

3.3 Heap Residence

As for *stack storage* LE/VSE also provides *heap storage* for PL/I for VSE programs and library routines. This section describes *heap storage* definition and usage.

3.3.1 Heap Storage Overview

Heap storage is used to allocate storage that has a lifetime independent of the execution of the current routine. It remains allocated until it is explicitly freed or until the enclave terminates. Heap storage is shared among all program units and threads in an enclave. It is a collection of large *heap segments*, comprising the initial heap segment, which is dynamically allocated at the first request for heap storage, and increment heap segments which are obtained when the initial one is exhausted. Heap storage is further subdivided into *heap elements*, which are controlled by LE/VSE and delivered to the program upon request. There are two additional heaps; BELOWHEAP storage is allocated below the 16MB line and ANYHEAP storage is allocated anywhere in storage. Those two heaps are reserved for run-time library usage only. Do not confuse them with LE/VSE run-time options for user heap storage.

3.3.2 Tuning Heap Storage

For best performance, the initial heap segment should be large enough to satisfy all requests for heap storage. The main objective of tuning heap storage is to minimize the number of VSE/ESA GETVIS requests, which are used by LE/VSE for obtaining storage from the operating system. When running under CICS, HEAP BELOW is obtained from the CICS DSA, but HEAP ANYWHERE is obtained using VSE GETVIS services.

3.3.3 PL/I Usage of Heap Storage

PL/I storage allocated as a result of the PL/I ALLOCATE statement is within the PL/I area. If it has the storage class CONTROLLED or if the REFER option is in effect, then it must be released by the PL/I FREE statement. Allocated storage with the storage class BASED can be released by the PL/I FREE statement or by LE/VSE routines only when the structure is completely declared, does not contain the REFER option and requires no pad bytes to be added by the compiler.

3.3.4 HEAP Run-time Option

The HEAP run-time option controls the allocation of initial heap (non-LIFO) storage, controls additional heap storage allocation, and specifies from where heap storage is allocated and how it is managed. It also controls heap storage below the 16MB line in the special case where ANYWHERE is specified in the HEAP run-time option, but the program is running in AMODE 24.

HEAP (init_size,incr_size,ANYWHERE or BELOW,KEEP or FREE, initsz24,incrsz24).

init_size determines the initial allocation of heap storage.

incr_size determines the increments of additional heap storage acquisition when the initial heap is exhausted.

ANYWHERE or BELOW determines the placement of heap storage.

BELOW heap storage is acquired below the 16MB line.

ANYWHERE heap storage is acquired anywhere in storage, preferably above the 16MB line.

KEEP and FREE determines the management of heap storage.

KEEP specifies that storage allocated to heap increments is not released when the last storage is freed.

FREE specifies that storage allocated to heap increments is released when the last storage is freed, and is recommended especially in the CICS environment.

initsz24 determines the initial allocation of heap storage below the 16MB line for applications running with ALL31(OFF) which also have the parameter ANYWHERE in the HEAP run-time option.

3.3.4.1 Batch Considerations

The default value for *init_size* is 64KB (specified in the CEEDOPT macro). The recommended value for init_size is given in the RPTSTG listing under the heading *Heap statistics - Total heap storage used(suggested initial size).* For *incr_size* the same value is recommended. It is also recommended that the parameters ANYWHERE and FREE should be coded.

3.3.4.2 CICS Considerations

The default value for *init_size* is 4KB (specified in the CEECOPT macro). This value can be changed according to the suggested initial size, but care should be taken if BELOW is specified for heap storage placement, because in that case CICS DSA storage is used. Heap storage should have the parameters ANYWHERE and FREE specified to get the best placement and usage of heap storage.

3.3.5 Increment Rounding

In both batch and CICS, and for both HEAP and STACK, any initial or incremental allocations will be rounded up to the next page (4KB) boundary.

3.4 Effects of Other Run-time Options

There are three other run-time options which have a significant effect on the execution of an application program. These options are meant to provide information about resource usage during the execution of the program and for diagnosis purposes. Those options **must** be excluded when the program is in production.

3.4.1 STORAGE Run-time Option

STORAGE controls the initial content of storage when allocated and freed, and the amount of storage reserved for the out-of-storage condition.

STORAGE(heap_alloc_value,heap_free_value,dsa_alloc_value,reser_size)

Use of the STORAGE option increases the elapsed time for the program's execution, and may also affect other resource usage, so for performance reasons we strongly recommend that you specify:

```
STORAGE(NONE,NONE,NONE,8K)
```

When debugging your program, you will get a hex dump of different storage areas as a part of the LE/VSE CEE5DMP dump facility. To distinguish between different storage area types more easily, the STORAGE run-time option with initial values can be specified.

3.4.2 RPTOPTS Run-time Option

It generates, after an application has run, a report of run-time options in effect for the enclave. For performance reasons it is recommended **not** to specify the RPTOPTS option after a program is tuned, because there is a significant performance impact for CICS programs as well as for batch programs. The impact for CICS comes not only from the fact that the report is created, but also from the fact that it is written to an indirect transient data destination. We observed a significant increase in CPU time and dispatch count, and consequently also in response time.

Following is an example of an output listing resulting from specifying RPTOPTS(ON).

```
Options Report for Enclave TEST1 12/07/94 6:03:43 PM

LAST WHERE SET          OPTION
-----------------------------------------------------------------------
Installation default      ABPERC(NONE)
Installation default      ABTERMENC(RETCODE)
Installation default    NOAIXBLD
Installation default      ALL31(OFF)
Installation default      ANYHEAP(32768,16384,ANYWHERE,FREE)
Installation default      BELOWHEAP(32768,16384,FREE)
Installation default      CBLOPTS(ON)
Installation default      CBLPSHPOP(ON)
Installation default      CBLQDA(ON)
Installation default      CHECK(ON)
Installation default      COUNTRY(US)
Installation default      DEBUG
Installation default      DEPTHCONDLMT(10)
Installation default      ERRCOUNT(20)
Default setting         NOFLOW(99)
Invocation command        HEAP(65536,65536,ANYWHERE,FREE,16384,16384)
Installation default      INTERRUPT(OFF)
Default setting           ISAINC(0,0)
Default setting           ISASIZE(0,0,0)
Installation default      LIBSTACK(32768,16384,FREE)
Installation default      MSGFILE(SYSLST)
Installation default      MSGQ(15)
Installation default      NATLANG(UEN)
Invocation command        RPTOPTS(ON)
Invocation command        RPTSTG(ON)
Installation default    NORTEREUS
Installation default    NOSIMVRD
Invocation command        STACK(524288,524288,ANYWHERE,FREE)
Installation default      STORAGE(NONE,NONE,NONE,8192)
Installation default      TERMTHDACT(TRACE)
Installation default    NOTEST(NONE,†*†,†NOPROMPT†,†*†)
Installation default      TRAP(ON)
Installation default      UPSI(00000000)
Installation default      VCTRSAVE(OFF)
Installation default      XUFLOW(AUTO)
```

Figure 21. Example of RPTOPTS Run-time Option Report

3.4.3 RPTSTG Run-time Option

This generates, after an application has run, a report of the storage used for the enclave. For performance reasons it is recommended that you do **not** specify the RPTSTG option after the program is tuned and in production, especially for CICS applications. The impact for CICS comes not only from the fact that the report is created, but also from the fact that it is written to an indirect transient data destination. We observed a significant increase in CPU time, dispatch count, and consequently also in response time.

The RPTSTG run-time option is mainly used to tune HEAP and STACK options, but its report can also be used to tune other storage related options such as ANYHEAP, BELOWHEAP and LIBSTACK (used for run-time library routines only).

Following is an example of an output listing resulting from specifying RPTSTG(ON).

```
Storage Report for Enclave TEST1 12/08/94 8:06:14 AM

    STACK statistics:
      Initial size:                                      131072
      Increment size:                                    131072
      Total stack storage used (sugg. initial size):     271040
      Successful GETMAINs issued:                             2
      Successful FREEMAINs issued:                            0
    LIBSTACK statistics:
      Initial size:                                       32768
      Increment size:                                     16384
      Total stack storage used (sugg. initial size):          0
      Successful GETMAINs issued:                             1
      Successful FREEMAINs issued:                            0
    HEAP statistics:
      Initial size:                                       65536
      Increment size:                                     65536
      Total heap storage used (sugg. initial size):     4162080
      Successful Get Heap requests:                        2000
      Successful Free Heap requests:                          0
      Successful GETMAINs issued:                            65
      Successful FREEMAINs issued:                            0
    ANYHEAP statistics:
      Initial size:                                       32768
      Increment size:                                     16384
      Total heap storage used (sugg. initial size):         100
      Successful Get Heap requests:                           1
      Successful Free Heap requests:                          0
      Successful GETMAINs issued:                             1
      Successful FREEMAINs issued:                            0
    BELOWHEAP statistics:
      Initial size:                                       32768
      Increment size:                                     16384
      Total heap storage used (sugg. initial size):        2295
      Successful Get Heap requests:                           6
      Successful Free Heap requests:                          4
      Successful GETMAINs issued:                             1
      Successful FREEMAINs issued:                            0
    Additional Heap statistics:
      Successful Create Heap requests:                        1
      Successful Discard Heap requests:                       0
      Total heap storage used:                                0
      Successful Get Heap requests:                           0
      Successful Free Heap requests:                          0
      Successful GETMAINs issued:                             0
      Successful FREEMAINs issued:                            0
End of Storage Report
```

Figure 22. Example of RPTSTG Report

From this example you can see the report of storage usage for a program which does not have its run-time options tuned. For stack storage LE/VSE must request storage from VSE/ESA twice instead once. The case is even worse with heap storage, for which there are 2000 get heap requests to LE/VSE, which resulted in 65 requests when LE had to get storage from VSE/ESA.

3.5 ALL31 Run-time Option

ALL31 specifies whether an application can run entirely in AMODE(31) or whether an application has one or more AMODE(24) routines. This option has no effect on storage usage, because storage is managed with STACK and HEAP options. However you must be aware of the application's storage requirements and request storage accordingly when you specify ALL31(OFF) and the program is running in AMODE(24).

3.5.1.1 ALL31(OFF)

Indicates that application has AMODE(24) routines. It is the default setting and you must also use the default setting STACK(,,,BELOW) for stack storage. AMODE switching is performed on calls to LE/VSE common run-time routines and callable services.

3.5.1.2 ALL31(ON)

Indicates that no user routines of an application are AMODE(24). It results in minimized AMODE switching across calls to common run-time library routines.

3.5.1.3 CICS Considerations

The default specification under CICS is ALL31(ON) to allow program placement and storage acquisitions above the 16MB line.

3.5.1.4 Batch Considerations

If an application consists entirely of AMODE(31) routines it might run faster with ALL31(ON) specified because AMODE switching is not required.

Table 4. ALL31, AMODE and RMODE vs HEAP and STACK Parameters ANY and BELOW

			partition LE 16MB		partition GT 16MB	
ALL31	**AMODE**	**RMODE**	**BELOW**	**ANY**	**BELOW**	**ANY**
ON	24	24	NO	NO	NO	NO
ON	31	24	YES	YES	YES	YES
ON	31	ANY	YES	YES	YES	YES
OFF	24	24	YES	NO	YES	NO
OFF	31	24	YES	YES	YES	YES
OFF	31	ANY	YES	YES	YES	YES

Notes:

1. The combination AMODE(24) and RMODE(ANY) is invalid and is not present in the table.
2. *partition LE 16MB* means that the partition upper boundary is below the 16MB boundary.
3. *partition GT 16MB* means that the partition upper boundary has an address above the 16MB line.

3.6 PL/I VSE Resource Usage Compared with DOS PL/I

This section describes the resource usage of PL/I VSE programs running under LE/VSE compared with the resource usage of DOS PL/I programs.

3.6.1 Executable Modules

LE/VSE does not support DOS PL/I phases, but they can coexist with PL/I VSE phases in your VSE/ESA system, as long as they have different names. Thus old programs compiled with the DOS PL/I compiler will continue to run, provided you retain your DOS PL/I Transient Library. LE/VSE also does not support load modules which are a mixture of PL/I for VSE and DOS PL/I object modules. All programs running under LE/VSE must be recompiled with the PL/I VSE compiler and then re-linked with the LE/VSE libraries in the search chain.

The size of the executable phase of a program compiled and linked with PL/I VSE is much smaller than the size of a phase compiled and link edited with the old compiler. In some cases its length is less than half that of a comparable phase built with the old compiler. The main reason for this is that link-editing object modules compiled by PL/I for VSE causes the inclusion of stubs for library routine dynamic calls, instead of complete resident library routines as is done with DOS PL/I. Those routines are now loaded at execution time from the LE/VSE run-time library. Programs compiled and link edited under LE/VSE can not FETCH modules compiled with DOS PL/I.

3.6.2 Object Modules

Object modules compiled with the DOS PL/I compiler can not be linkage edited with modules compiled with PL/I VSE. You must ensure before link edit that all source code for routines which are to be linked together has been recompiled with the PL/I for VSE compiler, and placed in an object library.

3.6.3 Virtual Storage Use

Programs compiled and executed under LE/VSE use considerably more storage than programs compiled and executed with the DOS PL/I compiler. Most of that additional storage is used by LE/VSE routines. For better understanding we will first discuss the usage of virtual storage by DOS PL/I programs.

3.6.3.1 DOS PL/I Use of Virtual Storage

Programs compiled with an old compiler are loaded into the partition at the low address end of the partition. The DOS PL/I initialization routine then sets up an environment which consists of:

- an executable program phase
- the program management area
- last-in/first-out (LIFO) storage
- non-LIFO storage

All storage apart from the executable program phase is called the *initial storage area* and is controlled by the ISASIZE run-time option. From that storage the program management area is allocated first, followed by LIFO storage. Non-LIFO storage is allocated from the top of the partition down. The area in between is called the major free area and is used for all storage acquisitions. When the ISA storage area is exhausted the program abends.

You can find more detailed information in *SC33-0019 DOS PL/I Execution Logic*.

3.6.3.2 PL/I VSE Use of Virtual Storage - Batch Considerations

A program compiled and link edited under PL/I VSE is loaded at the beginning of the partition, then the LE/VSE initialization/termination routine CEEBINIT is loaded, which sets up the LE/VSE environment. Then the PL/I event handler routine CEEEV010 is loaded together with some other routines. LE/VSE then requests initial stack storage from VSE/ESA, as specified in the STACK option. This storage is under the control of the LE/VSE storage manager which handles all requests for storage from the application program. When the application allocates heap storage for the first time, initial heap storage is requested from VSE/ESA, but after that it is used by the LE/VSE routines to satisfy requests from the program for heap storage. When the initial stack or heap storage is completely used, LE/VSE acquires storage increments from VSE/ESA. The following is the sequence followed by LE/VSE storage management for allocating areas for any program running in the LE/VSE environment:

- executable program phase
- program management area
- LE/VSE routines
- LE/VSE managed storage
- GETVIS controlled storage

The following is a table showing storage use for the same sample programs compiled with the PL/I for VSE compiler (IEL1AA) and also compiled using the DOS PL/I compiler (PLIOPT). The table shows in the first column some values when the program was compiled with the PLIOPT compiler, in the second column the corresponding values with the new IEL1AA compiler but without any PL/I or LE phases loaded in the SVA, and in the last column the corresponding values when all recommended PL/I and LE phases were loaded into the SVA.

Table 5. Storage Use DOS PL/I vs PL/I VSE without and with Phases in SVA

	PLIOPT	**IEL1AA**	**IEL1AA / SVA**
PLIOPT sample program			
Loaded phase length	11KB	5KB	5KB
Execution size	16KB	17KB	17KB
GETVIS used	20KB	776KB	292KB
IEL1AA sample program			
Loaded phase length	23KB	5KB	5KB
Execution size	28KB	17KB	17KB
GETVIS used	32KB	776KB	280KB

This comparison clearly shows that the recommended LE/VSE and PL/I phases should be loaded in the SVA.

3.6.3.3 PL/I VSE Use of Virtual Storage - CICS Considerations

At CICS startup time the LE/VSE environment is established by including the CEECCICS routine into the CICS nucleus area, if this routine does not already reside in the SVA. Other initialization routines are then loaded. If these are not pre-loaded into the SVA, then they are loaded into the CICS DSA if they are linked with RMODE(24), and into 31-bit GETVIS, if they are linked with RMODE(ANY) or RMODE(31).

When a program is executed, all storage requests for stack and heap storage are handled by the LE/VSE storage management routines. If there is no more storage available, an increment is requested by LE/VSE from CICS.

The optimum storage usage is accomplished when all LE/VSE and PL/I VSE SVA-eligible phases are loaded into the SVA. The following is an example of storage usage by the same transactions first compiled with the DOS PL/I compiler and then with the PL/I for VSE compiler. The PMNU, PINQ and PUPD transactions are sample applications provided by CICS for VSE.

Table 6. Storage Use DOS PL/I vs PL/I VSE with Phases in SVA

	PLIOPT	**IEL1AA**
PMNU - menu transaction	41152B	10008B
PINQ - inquire transaction	8152B	10936B
PUPD - update transaction	8408B	10936B

Chapter 4. Interfacing with Other Subsystems

4.1 CICS/VSE

Since the scope of this document covers both batch and CICS/VSE execution of PL/I for VSE programs, there is no separate discussion of CICS/VSE in this chapter.

4.2 DL/I DOS/VS

The scope of this document is to discuss storage, for a discussion of DL/I issues that are not specific to storage, please see the PL/I for VSE Migration Guide.

DL/I DOS/VS 1.10 initially imposed the restriction that DL/I parameter lists and parameters need to be below the 16MB line. This restriction is removed by PTF. The PTF is incorporated in VSE/ESA 2.1, so that if DL/I DOS/VS is installed from the VSE/ESA 2.1 Optional Products tape, **or** it was migrated from VSE/ESA 1.3 but has the PTF applied, then full 31-bit residence and exploitation is achieved. However, if DL/I DOS/VS was migrated from a VSE/ESA 1.3 system, and the PTF has not been applied, then parameter lists and parameters must be below 16MB. Under these circumstances you must

either install the PTF

or make the parameter lists and parameters reside below 16MB, which is done as follows

- **static storage** PL/I static storage is held as part of the link-edited phase. If your program has DL/I parameters in static storage, you should link edit the program with RMODE(24).
- **automatic storage** The allocation of automatic storage is done by the LE/VSE STACK run-time option. To ensure that automatic storage is allocated below the line, specify STACK(,,BELOW).
- **controlled storage** The allocation of controlled storage, and AREA variables, is done by the LE/VSE HEAP run-time option. To ensure that controlled storage is allocated below the line, specify HEAP(,,BELOW).
- **based storage** The allocation of based storage is done by the LE/VSE HEAP run-time option. To ensure that based storage is allocated below the line, specify HEAP(,,BELOW).

  If the parameter list is in based storage that is never allocated, it is not treated as based in this context. Its actual storage class will be that of the base variable to which its locator is pointing.

© Copyright IBM Corp. 1995

4.3 DB2 for VM and VSE (SQL/DS)

There are no special considerations for 31-bit addressing by PL/I for VSE programs which access SQL/DS databases, either in batch or on-line.

For a discussion of SQL/DS issues that are not specific to storage, please see the PL/I for VSE Migration Guide.

4.4 IBM MQSeries for VSE/ESA

This product was formerly known as ezBRIDGE Transact on VSE/ESA. It runs in a CICS/VSE environment, and provides MQSeries access for VSE/ESA applications. The access is provided by a call interface to VS COBOL II subprograms. The interfaces provided for VS COBOL II in CICS/VSE meant that the only access was from VS COBOL II applications.

LE for VSE supports:

compatibility interface This permits you to re-link VS COBOL II object modules with the LE for VSE library, and take advantage of LE services.

inter-language communication (ILC) This allows a program written in an LE conforming language to issue calls to another program written in an LE conforming language. This includes calls not only to PL/I for VSE and COBOL for VSE modules, but also calls to earlier COBOL programs linked with the compatibility library.

Sample programs and copy books are only provided for VS COBOL II and time did not permit the creation and testing of PL/I applications, but ILC should allow PL/I for VSE applications to be written to make use of the Messaging and Queuing Interface of IBM MQSeries for VSE/ESA. This needs to be formally tested.

Chapter 5. How to Determine Compile-time and Run-time Options

A number of different methods exist for specifying compile-time and run-time options for your program. This chapter gives you a short description of how to define them and how to decide which options to specify.

The following is a list of sections:

- minimum compiler storage requirements
- defining compile-time options
- how to decide on your compile-time options
- defining run-time options globally
- defining run-time options by application
- how to decide on your storage related run-time options

5.1 Minimum Storage Requirements for PL/I VSE Compiler

The compiler itself needs a partition with the minimum size of 640K, when running in a native VSE/ESA environment, where the partition′s size allocation is done in 64K increments. A second requirement is that at least 588K of GETVIS storage is available for the compiler. The third requirement is that it needs a minimum 16K of EXEC size. To illustrate this: the compiler can run in a 640K partition with EXEC sizes from 16K up to 52K which will still leave the minimum of 588K for the partition′s GETVIS.

The EXEC size value included in the Interactive Interface is 256K.

5.2 Defining Compile Time Options

When compiling your program you can use the following mechanisms for setting compile-time options. They are listed in order of precedence:

1. the installation-wide defaults
2. options specified in your compile JCL
3. options specified in %PROCESS or *PROCESS statement in your program

Compile-time options and their default values are described in *SC26-8057 PL/I VSE Installation and Customization Guide*.

5.2.1 Customizing Installation Default Compile-time Options

When the PL/I for VSE compiler is installed, you can change some compile-time options if the supplied defaults are not suitable for your installation. The sample job supplied in the IEL1OPTV.Z library member is illustrated in Figure 23 on page 38.

```
// JOB     IEL1OPTV
*
*  Assemble PL/I VSE Default Options
*
* ------------------------------------------------------------
// SETPARM VOLUME=¢VSE111¢         * Volume to use
// SETPARM START=¢????¢            * Starting extent trk/blk
// SETPARM LENGTH=¢0020¢           * Length of work file
* ------------------------------------------------------------
*
// LIBDEF  *,SEARCH=(PRD2.PROD)
// DLBL    IJSYSPH,¢ASSEMBLE.OUTPUT¢,0
// EXTENT  SYSPCH,&VOLUME,1,0,&START,&LENGTH   <=== change values
ASSGN      SYSPCH,DISK,VOL=&VOLUME,SHR
// OPTION  DECK
// EXEC    ASMA90
         PUNCH ¢CATALOG IEL1AV.OBJ REPLACE=YES¢
         PRINT ON,GEN
         IEL1COPV AGGREGA=NO,ATTRIBU=NO,COMPILE=NOS,CONTROL=¢OPTIMIZE¢,X
               DELETE=,ESD=NO,FLAG=I,FMARGIN=(2,72),FSEQUEN=(73,80),   X
               GONUMBE=NO,GOSTMT=NO,GRAPHIC=NO,INCLUDE=NO,INSOURC=YES, X
               LANGLVL=(OS,NOSPROG),LINECOU=55,LIST=NO,LMESSAG=YES,    X
               MACRO=NO,MAP=NO,MARGINI=NO,MDECK=NO,NEST=NO,NOT=,       X
               NUMBER=NO,OFFSET=NO,OPTIMIZ=NO,OPTION=YES,OR=,SIZE=MAX, X
               SOURCE=YES,STMT=YES,STORAGE=NO,SYNTAX=NOS,SYSTEM=VSE,   X
               TEST=(NO,NONE,SYM),TSTAMP=YES,XREF=(NO,F)
         END
/*
CLOSE      SYSPCH,PUNCH
// DLBL    IJSYSIN,¢ASSEMBLE.OUTPUT¢,0
// EXTENT  SYSIPT
ASSGN      SYSIPT,DISK,VOL=&VOLUME,SHR
// EXEC    LIBR,PARM=¢MSHP;ACCESS SUBLIB=LIB.USERLIB¢
/*
CLOSE      SYSIPT,SYSRDR
/*
/&
```

Figure 23. Sample JCL to Assemble Installation Default Options

Before executing this job, you must change:

- the SETPARM START value for SYSPCH output
- the LIBDEF statement

  if PL/I for VSE does not reside in the PRD2.PROD sub-library
- the target library

  where you want the output object module cataloged

Before changes can take effect, the phase IEL1AV must be relinked. Sample JCL is provided in the library member IEL1OPTL.Z shown in Figure 24 on page 39.

```
// JOB    IEL1OPTL
*
*  Linkedit PL/I VSE default options phase IEL1AV
*
// LIBDEF *,SEARCH=(LIB.USERLIB,PRD2.PROD)
// LIBDEF PHASE,CATALOG=LIB.USERLIB
// OPTION CATAL
 INCLUDE IEL$AV
// EXEC   LNKEDT,PARM=¢MSHP¢
/*
/&
```

Figure 24. Sample JCL to Link Edit Installation Default Options Phase

Before executing this job you must change both LIBDEF statements to ensure that the proper libraries are specified for SEARCH and CATALOG access. The search library must be consistent with the PARM statement in job IEL1OPTV, and the catalog library must be the one you will use during linkage editing.

5.2.2 Specifying Compile-time Options in JCL

Compile-time options can be specified in the JCL you use for invoking the compiler. They are provided in the parameter list on the EXEC statement as in the following example:

```
// EXEC IEL1AA,PARM='OPTIMIZE(2),MACRO,TEST(ALL,SYM)'
```

This method and the following one should normally be used for any temporary overrides. It is obviously not sensible to keep changing the defaults or standards once they have been established, except in exceptional cases.

5.2.3 Specifying Compile-time Options in %PROCESS or *PROCESS Statement

Compile-time options can be specified, as in the past, in the PL/I source program statement %PROCESS for the preprocessor run:

```
%PROCESS OPTIMIZE(2) MACRO TEST(ALL,SYM);
```

or in the *PROCESS statement for the compiler run:

```
*PROCESS OPTIMIZE(2) MACRO TEST(ALL,SYM);
```

5.2.4 Compile-time Option Effects on Storage Requirements and Run Time

The following compile-time options have an effect on your program's run-time storage requirements and run time:

GOSTMT does not increase execution time but increases storage requirements by approximately 4 bytes per PL/I statement.

GONUMBER does not increase execution time but increases storage requirements by approximately 6 bytes per PL/I statement.

OPTIMIZE(TIME) decreases run time and can also reduce the amount of main storage used for the program.

TEST increases execution time and storage requirements and should be used only for tuning and testing your program.

5.2.5 CICS Compile-time Considerations

When coding a CICS transaction, prior to compiling it, you must invoke the CICS Command Language Translator, which is described in *CICS Application Programmer's Reference Manual*. After the CICS translator step, compile your program with the SYSTEM(CICS) compile-time option. An example of translating and compiling the CICS sample program DFH$PMNU is given in Figure 25. This example uses a customized version of the Interactive Interface supplied procedure.

```
* $$ JOB JNM=COMPCICS,DISP=D,CLASS=A,NTFY=YES
* $$ LST DISP=D,CLASS=Q,PRI=3
* $$ PUN DISP=I,PRI=9,CLASS=A
// JOB COMPCICS TRANSLATE PROGRAM DFH$PMNU
// ASSGN SYSIPT,SYSRDR
// EXEC IESINSRT
$ $$ LST DISP=D,CLASS=Q,PRI=3
// JOB COMPCICS COMPILE PROGRAM DFH$PMNU
// SETPARM CATALOG=2
// IF CATALOG = 1 THEN
// GOTO CAT
// OPTION ERRS,SXREF,SYM,LIST,NODECK
// GOTO ENDCAT
/. CAT
// LIBDEF *,SEARCH=(PRD2.SCEECICS,PRD2.SCEEBASE)
// LIBDEF PHASE,CATALOG=PRD2.CONFIG
// OPTION ERRS,SXREF,SYM,CATAL
   PHASE DFH$PMNU,*
/. ENDCAT
// EXEC IEL1AA,SIZE=256K
*PROCESS INCLUDE;
* $$ END
// ON $CANCEL OR $ABEND GOTO ENDJ2
// OPTION NOLIST,NODUMP,DECK
// EXEC DFHEPP1$,SIZE=512K
*PROCESS XOPTS(CICS DEBUG);
* $$ SLI ICCF=(DFH$PMNU),LIB=(0013)
/*
/. ENDJ2
// EXEC IESINSRT
/*
// IF CATALOG NE 1 OR $MRC GT 4 THEN
// GOTO NOLNK
// EXEC LNKEDT,SIZE=256K
/. NOLNK
#&
$ $$ EOJ
* $$ END
/&
* $$ EOJ
```

Figure 25. JCL to Translate and Compile CICS Sample Program DFH$PMNU

5.3 How to Decide on your Compile-time Options

The appropriate compile-time options depend on the stage at which program testing is performed. In the early stages you should include as many options as needed, but execute the program with the NOOPTIMIZE option. When the program runs error free, the OPTIMIZE(TIME) option should be included to tell the compiler to carry out optimization. If any optimization messages are received, they must, of course, be resolved. For a list of messages see Table 9 on page 58.

When tuning is completed, any compile-time options which increase the storage requirements at execution time of the program should be removed. Those options are GOSTMT, GONUMBER and TEST and are described in 5.2.4, "Compile-time Option Effects on Storage Requirements and Run Time" on page 39.

5.4 Defining Run-time Options

Since LE/VSE establishes and controls the program's run-time environment, the run-time options are specified by LE/VSE macros. You can specify run-time options in the following different ways:

1. in your PL/I source code
2. in JCL
3. as application default
4. in the assembler user exit
5. as installation defaults

Those options are listed in order of precedence, from highest to lowest. You should note however, that an option may be specified in the installation defaults as ineligible to be overridden, in which case it can clearly not be changed.

5.4.1 Defining Run-time Options Globally

5.4.1.1 Batch Considerations

The CEEDOPT assembler source member is used as a sample to establish installation-wide default run-time options using the CEEXOPT macro. This source member can be edited and changed either during LE/VSE installation time or later. When assembled, it produces a CEEDOPT object module. All applications which run in the LE/VSE common run-time environment can use these default values for their run-time options by link-editing the CEEDOPT object module with the application. When the CEEDOPT module is link edited with CEEBINIT, installation-wide default values are established. The sample of the CEEDOPT source member supplied with LE/VSE is shown in Figure 26.

```
CEEDOPT  CSECT
CEEDOPT  AMODE ANY
CEEDOPT  RMODE ANY
         CEEXOPT ABPERC=((NONE),OVR),                                  X
               ABTERMENC=((RETCODE),OVR),                              X
               AIXBLD=((OFF),OVR),                                     X
               ALL31=((OFF),OVR),                                      X
               ANYHEAP=((32K,16K,ANYWHERE,FREE),OVR),                  X
               BELOWHEAP=((32K,16K,FREE),OVR),                         X
               CBLOPTS=((ON),OVR),                                     X
               CBLPSHPOP=((ON),OVR),                                   X
               CBLQDA=((ON),OVR),                                      X
               CHECK=((ON),OVR),                                       X
               COUNTRY=((US),OVR),                                     X
               DEBUG=((ON),OVR),                                       X
               DEPTHCONDLMT=((10),OVR),                                X
               ERRCOUNT=((20),OVR),                                    X
               HEAP=((64K,64K,ANYWHERE,KEEP,16K,16K),OVR),             X
               INTERRUPT=((OFF),OVR),                                  X
               LIBSTACK=((32K,16K,FREE),OVR),                          X
               MSGFILE=((SYSLST),OVR),                                 X
               MSGQ=((15),OVR),                                        X
               NATLANG=((UEN),OVR),                                    X
               NOTEST=((NONE,*,NOPROMPT,*),OVR),                       X
               RPTOPTS=((OFF),OVR),                                    X
               RPTSTG=((OFF),OVR),                                     X
               RTEREUS=((OFF),OVR),                                    X
               SIMVRD=((OFF),OVR),                                     X
               STACK=((512K,512K,BELOW,KEEP),OVR),                     X
               STORAGE=((NONE,NONE,NONE,8K),OVR),                      X
               TERMTHDACT=((TRACE),OVR),                               X
               TRAP=((ON),OVR),                                        X
               UPSI=((00000000),OVR),                                  X
               VCTRSAVE=((OFF),OVR),                                   X
               XUFLOW=((AUTO),OVR)
         DC    C¢5686-067 (C) COPYRIGHT IBM CORP. 1991, 1995¢
         DC    C¢LICENSED MATERIAL - PROGRAM PROPERTY OF IBM¢
         END
```

Figure 26. Sample CEEDOPT Source Program

The CEEWDOPT assembler source member supplied with LE/VSE, provides JCL which when executed results in installation-wide run-time options. In that job, the CEEDOPT module is link edited into phases CEEBINIT and CEEBPIPI.

5.4.1.2 CICS Considerations

When you want to establish installation-wide CICS run-time options the CEECOPT source member is used. The sample as supplied in the LE/VSE SCEECICS sublibrary is shown in Figure 27.

```
CEECOPT  CSECT
CEECOPT  AMODE ANY
CEECOPT  RMODE ANY
         CEEXOPT ABTERMENC=((ABEND),OVR),                              X
               ALL31=((ON),OVR),                                       X
               ANYHEAP=((4K,4K,ANYWHERE,FREE),OVR),                    X
               BELOWHEAP=((4K,4K,FREE),OVR),                           X
               CBLOPTS=((ON),OVR),                                     X
               CBLPSHPOP=((ON),OVR),                                   X
               CHECK=((ON),OVR),                                       X
               COUNTRY=((US),OVR),                                     X
               DEBUG=((ON),OVR),                                       X
               DEPTHCONDLMT=((10),OVR),                                X
               ERRCOUNT=((20),OVR),                                    X
               HEAP=((4K,4K,ANYWHERE,KEEP,4K,4K),OVR),                 X
               INTERRUPT=((OFF),OVR),                                  X
               LIBSTACK=((4K,4K,FREE),OVR),                            X
               MSGFILE=((SYSLST),OVR),                                 X
               MSGQ=((15),OVR),                                        X
               NATLANG=((UEN),OVR),                                    X
               NOTEST=((NONE,*,NOPROMPT,*),OVR),                       X
               RPTOPTS=((OFF),OVR),                                    X
               RPTSTG=((OFF),OVR),                                     X
               STACK=((4K,4K,ANYWHERE,KEEP),OVR),                      X
               STORAGE=((NONE,NONE,NONE,0K),OVR),                      X
               TERMTHDACT=((TRACE),OVR),                               X
               TRAP=((ON),OVR),                                        X
               UPSI=((00000000),OVR),                                  X
               VCTRSAVE=((OFF),OVR),                                   X
               XUFLOW=((AUTO),OVR)
         DC    C¢5686-067 (C) COPYRIGHT IBM CORP. 1991, 1995¢
         DC    C¢LICENSED MATERIAL - PROGRAM PROPERTY OF IBM¢
         END
```

Figure 27. Sample CEECOPT Source Program

The CEEWCOPT member which is also supplied with LE/VSE, provides JCL to establish installation-wide CICS run-time options. In this job the CEEDOPT module is link edited into the phase CEECCICS, which is the initialization/termination routine for the CICS environment and is loaded at CICS startup time as part of the CICS nucleus.

5.5 Defining Run-time Options by Application

You can also specify run-time options which apply to an individual application. The possibilities are shown below:

- as application default options established in CEEUOPT
- as a parameter list in JCL
- in the assembler user exit
- in the PL/I application source code

5.5.1 Batch Considerations

5.5.1.1 Application Default Options Establishment in CEEUOPT Source File

The CEEUOPT assembler source file is used as a sample of how to establish run-time options for a specific application. The CEEXOPT macro creates the CEEUOPT object module, when it is assembled. When the CEEUOPT module is link edited with an application program, it establishes user default options for that application. The CEEUOPT source member is cataloged in the LE/VSE base sublibrary and is the same as the CEEDOPT member. For performance reasons we recommend that you specify in the CEEUOPT assembly only those options which are different from your default settings. The sample job supplied in source member CEEWUOPT and used to assemble and catalog the application default object module is shown in Figure 28.

```
// JOB CEEWUOPT
/* *******************************************************************
/* *                                                                 *
/* * IBM LANGUAGE ENVIRONMENT/VSE                                    *
/* *                    VERSION 1 RELEASE 1 MODIFICATION 0           *
/* *                                                                 *
/* * JOB TO CREATE APPLICATION-SPECIFIC RUN-TIME OPTIONS DEFAULTS    *
/* *                                                                 *
/* * NOTE: This sample JCL uses the sublibrary that the LE/VSE       *
/* *       install defaults to. If you have installed LE/VSE into    *
/* *       a different library/sublibrary then this JCL will         *
/* *       require modification.                                     *
/* *                                                                 *
/* *       Lines indicated with ¢<--¢ may need to be changed         *
/* *       to suit the environment at your site.                     *
/* *                                                                 *
/* *******************************************************************
// OPTION DECK
// LIBDEF *,SEARCH=PRD2.SCEEBASE                       <-- LE Run Time Lib
// DLBL IJSYSPH,¢PUNCH.OUTPUT¢,0,SD
// EXTENT SYSPCH,<volid>,1,0,<start>,<length>  <-- change volid,extents
ASSGN SYSPCH,DISK,VOL=<volid>,SHR                       <-- change volid
// EXEC ASMA90,SIZE=512K,PARM=¢EXIT(LIBEXIT(EDECKXIT))¢
     PUNCH ¢ CATALOG CEEUOPT.OBJ REPLACE=YES¢
CEEUOPT  CSECT
CEEUOPT  AMODE ANY
CEEUOPT  RMODE ANY
         CEEXOPT ALL31=((ON),OVR),                                     X
               HEAP=((16K,8K,ANYWHERE,FREE),OVR),                      X
               STACK=((32K,32K,ANYWHERE,FREE),OVR)
         DC    C¢5686-067 (C) COPYRIGHT IBM CORP. 1991, 1995¢
         DC    C¢LICENSED MATERIAL - PROGRAM PROPERTY OF IBM¢
         END
/*
CLOSE SYSPCH,PUNCH
// DLBL IJSYSIN,¢PUNCH.OUTPUT¢,0,SD
// EXTENT SYSIPT
ASSGN SYSIPT,DISK,VOL=<volid>,SHR                       <-- change volid
// EXEC LIBR,PARM=¢ACCESS SUBLIB=YOURLIB.YOURSUB¢          <-- Your lib
CLOSE SYSIPT,SYSRDR
/&
```

Figure 28. Sample CEEWUOPT Source Program

5.5.1.2 Application Default Options Establishment in JCL Parameter List

Run-time options can be passed to an application program by a parameter list specified in the JCL EXEC statement. Besides run-time options, program arguments can also be specified in the parameter list. The delimiter between them is "/" and the following rules apply:

Table 7 (Page 1 of 2). Formats for Specifying Run-time Options and Program Arguments	
Possible combinations	**Format**
Run-time options and program arguments both present	'run-time options/program arguments'
Only run-time options are present	'run-time options/'
Only program arguments are present	'/program arguments' or 'program arguments'

We also show an example of passing run-time options without program arguments in the JCL PARM attribute in Figure 29.

```
// EXEC  ,SIZE=128K,PARM=¢RPTOPTS(ON),RPTSTG(ON),STACK(512K,64K,BELOW),-
               HEAP(64K,64K,BELOW,FREE),ALL31(OFF)/¢
```

Figure 29. Run-time Options Specified in JCL

5.5.1.3 Application Default Options Establishment in PL/I Source Program

Run-time options can also be passed to an application program by the PLIXOPT character string in your source code. Those options override the installation defaults, but are overridden by options in the JCL PARM operand on the EXEC statement.

```
        DCL PLIXOPT CHAR(22) VAR INIT(¢RPTOPTS(ON),RPTSTG(ON)¢);
```

Figure 30. Run-time Options Specified in PL/I Source Program

5.5.2 CICS Considerations

Under CICS you can not pass run-time options as parameters when the application is invoked, but they can be specified by using one of the following methods:

- as CICS default options established in CEECOPT
- as application default options established in CEEUOPT
- in the assembler user exit
- in the PL/I application code

5.5.2.1 CICS Default Options Establishment in CEECOPT

Default options for CICS are specified in the CEECOPT source file, then assembled and link edited into CEECCICS to become the installation-wide CICS default run-time options. The process is described in the previous section.

5.5.2.2 CICS Default Options Establishment in CEEUOPT

The default options are specified in the CEEUOPT source file, then assembled and link edited with your application program. The process is described in the previous section, but there is also the possibility to include the assembly step into the JCL for the compilation and link edit of your CICS program. An example of this implementation is given in Figure 31.

```
* $$ JOB JNM=COMCICS,DISP=D,CLASS=A,NTFY=YES
* $$ LST DISP=D,CLASS=Q,PRI=3
* $$ PUN DISP=I,PRI=9,CLASS=A
// JOB COMCICS TRANSLATE PROGRAM DFH$PMNU
// ASSGN SYSIPT,SYSRDR
// EXEC IESINSRT
$ $$ LST DISP=D,CLASS=Q,PRI=3
// JOB COMPILE COMPILE PROGRAM DFH$PMNU
// SETPARM CATALOG=1
// IF CATALOG = 1 THEN
// GOTO CAT
// OPTION ERRS,SXREF,SYM,LIST,NODECK
// GOTO ENDCAT
/. CAT
// LIBDEF *,SEARCH=(PRD2.SCEECICS,PRD2.SCEEBASE)
// LIBDEF PHASE,CATALOG=PRD2.CONFIG
// OPTION ERRS,SXREF,SYM,CATAL
   PHASE DFH$PMNU,*
   INCLUDE DFHELII
/. ENDCAT
// EXEC IEL1AA,SIZE=256K
*PROCESS INCLUDE SYSTEM(CICS) STORAGE ;
* $$ END
// ON $CANCEL OR $ABEND GOTO ENDJ2
// OPTION NOLIST,NODUMP,DECK
// EXEC DFHEPP1$,SIZE=512K
*PROCESS XOPTS(CICS DEBUG);
 /* $SEG(DFH$PMNU),COMP(SAMPLES),PROD(CICS/VS):                     */
 /*******************************************************************/
 /*      DFH$PMNU   CICS/VS SAMPLE FILEA OPERATOR INSTRUCTION MENU   */
 /*******************************************************************/
    MENU::PROC OPTIONS(MAIN);
      EXEC CICS SEND MAP(¢DFH$PGA¢) MAPONLY ERASE;
      EXEC CICS RETURN;
      END;
/*
/. ENDJ2
// EXEC IESINSRT
/*
// IF CATALOG NE 1 OR $MRC GT 4 THEN
// GOTO NOLNK
// EXEC ASMA90,SIZE=(ASMA90,50K),PARM=¢EXIT(LIBEXIT(EDECKXIT))¢
CEEUOPT  CSECT
CEEUOPT  AMODE ANY
CEEUOPT  RMODE ANY
         CEEXOPT ALL31=(ON),                                        X
               HEAP=(4K,4K,ANY,KEEP,4K,4K),                         X
               RPTOPTS=(OFF),                                       X
               RPTSTG=(OFF),                                        X
               STACK=(12K,4K,ANYWHERE,KEEP),                        X
               STORAGE=(NONE,NONE,NONE,0K),                         X
               TRAP=(ON)
         DC    C¢5686-067 (C) COPYRIGHT IBM CORP. 1991, 1995¢
         DC    C¢LICENSED MATERIAL - PROGRAM PROPERTY OF IBM¢
         END
/*
// EXEC LNKEDT,SIZE=256K,PARM=¢AMODE=31,RMODE=ANY¢
/. NOLNK
#&
$ $$ EOJ
* $$ END
/&
* $$ EOJ
```

Figure 31. Sample CICS Compilation Job with User Run-time Options Included

5.5.2.3 CICS Default Options Establishment in Assembler User Exit

When you run your application, LE/VSE runs the assembler user exit CEEBXITA before the enclave is initialized. A sample of CEEBXITA is delivered with LE/VSE as a source member. The run-time options can be specified in that exit and then link edited with an application.

5.5.2.4 CICS Default Options Establishment in PL/I Source Program

Run-time options can be passed to an application program also by the PLIXOPT string in your source code. Those options override the installation-wide options specified in CEECOPT.

```
DCL PLIXOPT CHAR(22) VAR INIT(¢RPTOPTS(ON),RPTSTG(ON)¢);
```

Figure 32. Run-time Options Specified in PL/I Source Program

5.6 How to Decide on your Storage Related Run-time Options

Tuning run-time storage involves the allocation of suitable initial amounts for both heap and stack storage areas. This allocation is done with the HEAP and STACK run-time options.

When unsuitable values are used, it is possible that a program may be unable to make allocations of storage, especially if the increment values are too large. This is especially important for CICS programs, with the parameter BELOW specified in run-time options HEAP and STACK, because in that case storage is acquired from CICS DSA and can affect the overall performance of the CICS subsystem.

5.6.1 Common Run-time Storage Usage Considerations

A program′s use of storage at run time can seriously affect performance. The storage class attribute and run-time options HEAP and STACK are key elements in understanding and controlling run-time storage use.

5.6.1.1 Storage Class Attributes

Following is a list of possible classes of storage, which can be defined in declarations of variables.

STATIC Variables declared with the STATIC attribute have storage allocated in the static CSECT which is a part of your program.

AUTOMATIC Variables declared with the AUTOMATIC attribute (default value) have storage allocated on block entry (PROCEDURE or BEGIN block) from *stack* storage.

CONTROLLED Variables declared with the CONTROLLED attribute have storage allocated at run time when an ALLOCATE statement is executed. That storage is allocated from *heap* storage.

BASED Variables declared with the BASED attribute have storage allocated at run time when an ALLOCATE statement is executed. That storage is allocated from *heap* storage.

5.6.2 Using the STORAGE Compile-time Option

The STORAGE compile-time option specifies whether a compiler listing should include a table that gives the storage requirements for the object module. Two examples of a storage table output are given in Figure 33 and Figure 34.

```
                          STORAGE REQUIREMENTS
BLOCK, SECTION OR STATEMENT TYPE                LENGTH (HEX) DSA SIZE (HEX)
*TEST1                      PROGRAM CSECT         848   350
*TEST2                      STATIC CSECT          340   154
TEST                        PROCEDURE BLOCK       430   1AE    512    200
IJKEXHC                     PROCEDURE BLOCK       300   12C    240     F0
BLOCK 3       STMT 6        ON UNIT               116    74    192     C0
```

Figure 33. Storage Requirements Table for Program TEST

Calculation for this example shows stack storage requirements of 846 bytes for procedure blocks and 944 bytes for DSA storage, but the RPTSTG report showed a value of 11552 bytes for suggested initial stack segment which is far in excess of the predicted requirements.

```
                          STORAGE REQUIREMENTS
BLOCK, SECTION OR STATEMENT TYPE              LENGTH (HEX)  DSA SIZE (HEX)
*SAMPLE1                    PROGRAM CSECT   4540   11BC
*SAMPLE2                    STATIC CSECT    3044    BE4
SAMPLE                      PROCEDURE BLOCK 2428    97C  81264     13D70
BLOCK 2       STMT 23       ON UNIT         170      AA    208        D0
BLOCK 3       STMT 27       ON UNIT         162      A2    216        D8
BLOCK 4       STMT 33       ON UNIT         304     130    288       120
NEXT_WORD                  PROCEDURE BLOCK 1022     3FE    170       368
LOOKUP_WORD                PROCEDURE BLOCK  448     1C0    256       100
```

Figure 34. Storage Requirements Table for Sample Program IEL1ESO

The calculation for this second example shows stack storage requirements of 96936 bytes for procedure blocks and for DSA storage, but in this case the RPTSTG report showed value of 96488 bytes for suggested initial stack segment which is quite close to the predicted requirements.

The previous two examples showed us that the compile-time storage report is not a reliable basis for calculating storage requirements. Instead of using the STORAGE prediction, you should use the RPTSTG report obtained during final testing.

5.6.3 Using RPTSTG Run-time Option

When the RPTSTG run-time option is set on, it produces a report of the storage used by the enclave during its execution. You should use the RPTSTG report to adjust the ANYHEAP, BELOWHEAP, HEAP and STACK run-time options. From the storage report use *suggested initial value* to set initial and increment values for your run-time options. This will reduce the number of times the LE/VSE storage manager makes requests to acquire storage for your application. More information about those run-time options is given in Chapter 3, “Run-time Options and Their Effects.”

5.6.4 Recommended Initial and Increment Values

The following table gives recommended storage values for *batch* and *CICS* usage.

Table 8. Recommended Run-time Storage Values

STORAGE TYPE	BATCH	CICS	ENVIRONMENT
Stack initial size	128K	4K	Thread
Stack increment size	64K	4K	Thread
Libstack initial size	16K	4K	Thread
Libstack increment size	8K	4K	Thread
Heap initial size	32K	4K	Enclave
Heap increment size	32K	4K	Enclave
Anyheap initial size	16K	4K	Enclave
Anyheap increment size	8K	4K	Enclave
Belowheap initial size	8K	4K	Enclave
Belowheap increment size	4K	4K	Enclave

Chapter 6. A Check-list for Initial Implementation

The measurements and analysis on which these recommendations are based were given earlier in this publication, so the list will be kept as simple as possible.

6.1 The Shared Virtual Areas

1. All recommended LE and PL/I 24-bit phases should be in the SVA

   CICS based components may not be loadable if the DSA is constrained, so these at least **MUST** be in the SVA

2. All recommended LE and PL/I 31-bit phases should be in the SVA

   31-bit storage is usually not a constrained resource

3. All eligible (but not recommended) LE 31-bit phases should be in the SVA

   31-bit storage is usually not a constrained resource

4. All eligible (but not recommended) LE 24-bit phases should be in the SVA

   These only require 5K, but you may be more selective if storage is critical

5. All eligible (but not recommended) PL/I 31-bit phases should be in the SVA

   31-bit storage is usually not a constrained resource, but these may be tuned down at leisure

6. Selected eligible (but not recommended) PL/I 24-bit phases should be in the SVA

   Careful selection of heavily used components should be carried out - since the total requirement for these is about 80K, and some are rarely used, it is best to be selective.

You will certainly need to increase both SVAs. If the recommendations cause the shared area to go over a megabyte boundary, then instead of trying to reduce to the boundary by removing components from the SVA, you should consider what components of other sub-systems (especially CICS) can in fact also be put into the SVA.

6.2 Run-time Options

1. Tune HEAP and STACK following the guidelines in Chapter 5, “How to Determine Compile-time and Run-time Options”
2. Do **not** use RPTOPTNS or RPTSTG in a production system
3. Run all programs possible in 31-bit mode

   Calling and called programs within the same load-module **must** be migrated together

4. Use application specific run-time options for storage
   a. PARM on the EXEC statement for batch
   b. link-edit with CEEUOPT for CICS
5. Use installation standard run-time options for non-storage functions

© Copyright IBM Corp. 1995

6.3 Compilation

Pl/I for VSE is fully supported by the Interactive Interface Compile dialogs. The SIZE operand used to execute the compiler is 256K. Our tests indicate that in the region of 600KB GETVIS is needed. Not all VSE/ESA 2.1 Environment B static and dynamic partitions are large enough. If you intend to have multiple concurrent compilations, you may need to change either the dynamic class table, or the BG allocation procedure at IPL time, or both.

6.4 Differences from DOS PL/I

For discussion of migration issues and differences from DOS PL/I, you should see:

PL/I for VSE Migration Guide

LE Concepts and Facilities

PL/I Installation and Customization

LE Installation and Customization

Appendix A. Performance Considerations

This appendix describes various methods of improving the efficiency of PL/I programs. It also provides a checklist which identifies some of the factors that can affect the performance of PL/I programs when using the PLI/VSE Version 1 compiler under LE/VSE Version 1.

This appendix will discuss the following topics:

- compile time optimization
- global optimization features
- storage efficiency
- tuning checklist

A.1 Compile Time Optimization

You can write a program as a single external procedure, but a better approach is to write your program as a number of smaller external procedures, or modules. One of the advantages of modular programming is that the compiler will need less time for the compilation of small programs. Program size affects the time and space required for compilation, but the increase in compile time is not directly proportional to the program size, especially if the compiler has to spill its text onto auxiliary storage. Also maintenance of and changes to the program are easier when modular techniques are used when writing the program.

The compiler is loaded into the partition via the JCL statement

```
// EXEC IEL1AA,SIZE=128K,PARM='various compile time options'
```

Although the compiler runs below the 16MB line, PL/I applications created by the compiler can use VSE/ESA's 31-bit extended addressing.

The SIZE specified in the EXEC statement determines the amount of virtual storage directly available to the compiler in the program area. All other partition storage is considered as partition GETVIS storage and is used by the compiler as working storage. The minimum value for SIZE is 16KB, but SIZE=256K, which is the default value used by the Interactive Interface COMPILE option, gave the same performance results. Don't forget that the minimum partition size is 640KB and the minimum GETVIS area is 588KB as described in Chapter 5, "How to Determine Compile-time and Run-time Options."

Avoid specifying SIZE=AUTO because in this case the value taken for SIZE is the length of the largest IEL1 phase (IEL1AE with length of 198KB). The compiler dynamically allocates storage during execution for better performance, but specifying a program area size which is more than the minimum has reduced GETVIS, thus reducing the working storage available to the compiler.

Always use as much working storage as possible to obtain maximum performance from the compiler. In general, the larger the source program, the greater the amount of storage necessary for maximum performance. The default compile-time option SIZE(MAX) is recommended.

© Copyright IBM Corp. 1995

The NOOPTIMIZE compile-time option will result in fast compilation speed but bypasses optimization of the program, so it should normally be avoided, except in the early stages of development.

Consider using the NOSYNTAX and NOCOMPILE compile-time options to avoid unnecessary compilations of your program in the early stages of testing.

A.1.1 VDISK Usage

The compiler requires a sequential data set for its own use as a temporary work-file, which is also known as a spill file. This data set is used when working storage in the partition is exhausted. It must reside on a direct access device which is assigned to SYS001. Performance will be improved if SYS001 is assigned to a VSE Virtual Disk.

You should refer to *VSE/ESA System Control Statements* for full details of the implementation of VDISK.

A.2 The Global Optimization Features

The PL/I compiler attempts to generate object programs that run fast at execution time. In most cases the compiler generates efficient code for statements in the same sequence as was used by the programmer. However, in some cases the compiler might alter the sequence of statements or operations to improve execution performance.

The compiler carries out the following types of optimization:

- Expressions:
  - Common expression elimination
  - Redundant expression elimination
  - Simplification of expressions
- Loops:
  - Transfer of expressions out of loops
  - Special-case code for DO statements
- Arrays and structures:
  - Initialization
  - Assignments
  - Elimination of common control data
- In-line code for:
  - Conversions
  - RECORD I/O
  - String manipulation
  - Built-in functions
- Input/output:
  - Key handling for REGIONAL data sets
  - Matching format lists with data lists
- Other:
  - Library subroutines
  - Use of registers
  - Analyzing run-time options during compile time

A.2.1 Limitations

There are some limitations to the optimization which the compiler can perform.

- PL/I attempts full optimization only when the compile-time option OPTIMIZE(TIME) is in effect
- the compiler has a limitation on the number of variables that it can consider for global optimization
- it has a limitation on the number of flow units that it can consider for flow analysis
- if the static CSECT or DSA exceeds 4096 bytes, the compiler has to generate additional code to address more storage
- if the object code for a procedure exceeds 4096 bytes, the compiler has to repeatedly reset base registers

The use of modular programming techniques will avoid almost all of the limitations listed above.

A.3 Storage Efficiency

The output of the compiler is well suited for requirements of VSE, but there are some points which should be considered when tuning your program.

Use the run-time option RPTSTG when tuning your program
The output report of the RPTSTG option helps you in specifying proper storage related run-time options. When you adjust values for the first time, repeat the whole procedure at least once to get properly tuned values. You can find more information about tuning run-time storage in Chapter 3, "Run-time Options and Their Effects" and Chapter 5, "How to Determine Compile-time and Run-time Options."

Use modular programming techniques
Place statements frequently executed together in the same section of the source program.

Avoid large branches within the program

Put values used together in the same part of storage.
Take care that items within the aggregate which are accessed together are held together. The choice between using arrays of structures or structures of arrays can be critical.

Control the linkage editor to ensure that CSECTs used together are placed together within the same storage pages, if possible.

A.4 Performance Tuning Checklist

This section is based on the *PL/I Performance Tuning* document written by J.D.Brownsmith and R.J.Arellanes. All examples were adapted and tested with the PL/I VSE compiler. New options not included in the original document have been added.

A.4.1 Obtaining the Listings Needed for Tuning

The first step is to collect data by compiling and executing the program with different options specified.

Obtain Source Listings: Compile the program selected for tuning with the following compile-time options specified:

SOURCE
STMT
NEST
OPTIONS
LIST
OPT(TIME)

Those options will produce a source listing with indicated nesting levels as a result of the NEST compile-time option. You will get a generated code listing as a result of the LIST compile-time option. This listing can be used to find library calls.

The compiler can also produce some messages which are tuning related and these will be covered later.

Obtain the RPTSTG Report Listing Compile the program a second time and this time also run it. Use the normal run-time options with the exception of the addition of the RPTSTG option which will produce a report of storage used at run time. This report will be used for tuning the HEAP and STACK run-time options.

A.4.2 Selection of Compile-time Options

There are several compile-time options which have a performance impact during the execution of application programs. The most significant ones are listed here:

OPTIMIZE(TIME) and NOOPTIMIZE
The default value is NOOPTIMIZE and means that the compiler will not optimize your application program for time and consequently that the compile time will be shorter. OPTIMIZE(0) means exactly the same as NOOPTIMIZE. This option is recommended during program development when frequent compilations are necessary.

OPTIMIZE(2) is equivalent to OPTIMIZE(TIME) and has the effect that the compiler will attempt optimization as described in A.2, "The Global Optimization Features" on page 54. The major effect of this option is that the compiler moves any eligible code out of the loops. You are strongly advised to use OPTIMIZE(2) for production runs.

TEST and NOTEST The TEST option specifies the level of testing capability that the compiler generates as a part of the object code. This option increases the size of the object code and so affects the performance of your program. When the program is considered ready for production you may want to limit the number and placement of hooks or to remove them altogether. You may consider compiling the program

with the minimum number of hooks and performing your tuning tasks on the block boundary. This can be done by specifying TEST(BLOCK).

ORDER and REORDER
ORDER and REORDER are optimization options that are specified for a procedure or begin block. If neither is specified the default is ORDER for external procedures. Internal procedures or begin blocks have this option inherited from the containing external block.

ORDER Use the ORDER option only when you must be sure that the most recently assigned values of variables are needed for ON-units which can be entered during execution of statements and expressions in the block. Be aware that the ORDER option is the default and you must specify REORDER to avoid it.

REORDER is the optimization option which allows the compiler to generate optimized code by excluding from a loop all invariant expression statements which are executed only once. An example of this is:

```
EXAMPLE: PROCEDURE OPTIONS(MAIN) REORDER;
```

A.4.3 Tuning the Run-time Storage Use

The following are brief descriptions of run-time options which affect the execution performance of your application program.

HEAP run-time option This option and how to tune it are described in Chapter 3, "Run-time Options and Their Effects" on page 23.

STACK run-time option This option and how to tune it are described in Chapter 3, "Run-time Options and Their Effects" on page 23.

RPTOPTS run-time option is described in Chapter 3, "Run-time Options and Their Effects" on page 23.

RPTSTG run-time option is described in Chapter 3, "Run-time Options and Their Effects" on page 23.

The program′s use of storage at run time can seriously affect performance, so it is important to be able to predict and influence that use. Chapter 3, "Run-time Options and Their Effects" and Chapter 5, "How to Determine Compile-time and Run-time Options" describe in detail how to predict run-time storage requirements and placement accurately, and how to tune it effectively.

A.4.4 Analysis of Compiler Messages Affecting Performance

There are several compiler warning messages provided to assist you in tuning your program. If performance is a problem, then the program should be restructured so that optimization can be performed over the entire external block. When your program is properly structured these messages will not appear in your compilation listing.

Message	Message description
IEL0910I W	Compiler restriction. Too many ″call″ and function references. Optimization is inhibited for the program.
IEL0911I W	Compiler restriction. Too many locator label or entry variable assignments.Optimization is inhibited for the program.
IEL0912I W	Compiler restriction. Too many assignments with base locators label or entry variables. Optimization is inhibited for the program.
IEL0913I W	Compiler restriction. Too many locator temporaries active for optimization. Optimization is inhibited for the program.
IEL0915I W	Compiler restriction. Too many statement label constants. Optimization is inhibited for the program.
IEL0917I W	Block contains n flow units. Global optimization performed only in DO groups.
IEL0919I W	N variables in program. Global optimization performed for 255 variables. Local optimization performed on remainder.
IEL0920I W	DO group contains N flow units. Global optimization is restricted. Optimization is inhibited for the program.
IEL0925I W	Flow within block or DO group is too complex. Global optimization is restricted.
IEL0940I W	The INTERRUPT option and the STRINGRANGE, SUBSCRIPRANGE, SIZE and STRINGSIZE conditions conflicts with the OPTIMIZE option. Execution time may be increased.

Table 9. Some Compiler Warning Messages Related to Tuning

A.4.5 Analysis of the Attributes of Procedures and Begin Blocks

First examine the coding style of the program. Ensure that the program is well structured. The compiler can optimize code better and over larger blocks of code if the program is well structured and has a straight flow of control. Additionally structuring will also ease maintenance of the program.

REORDER The REORDER option can have a major beneficial impact on run-time performance. It allows the compiler to remove all allowable code from the loop.

REENTRANT The REENTRANT option causes the compiler to generate additional code to ensure that the program is reentrant. This allows the program to be placed into the shared virtual area (SVA), which provides performance benefits for all frequently used programs, but especially those used in CICS.

BEGIN blocks BEGIN blocks can be used to sequence code in the same way as DO statements. However, BEGIN - END defines a block prologue and epilogue code is generated, which a DO group does not have. From a performance point of view use DO groups rather then BEGIN blocks.

GOTO statements The use of the GOTO statement inhibits the compiler from moving code around the target of the GOTO, due to the fixed presence of a label there.

ON UNITS
: Use on-units for executing condition handling strictly according to the language rules. Using on-units for your own CONDITION handling may be slower in execution and is not recommended.

PROCEDURE SIZE
: Check the size of the procedure to ensure that you are using a sensible program size. If the external procedure is too big then optimization can be inhibited. If the procedure is small and frequently executed then too much execution time may be spent on executing the procedure's prologue and epilogue.

A.4.6 Tuning Statement Prefixes and Labels

Statement prefixes and labels can inhibit optimization. Exclude from your program all unnecessary prefixes and labels.

CONDITION PREFIX
: A condition prefix specifies the enabling or disabling of various PL/I conditions.

LABEL PREFIX
: A label prefix identifies a statement. It will inhibit optimization. When the compiler can't determine all the code that can branch to a label, code cannot be moved past the label.

NOFIXEDOVERFLOW condition
: The effect can vary greatly, depending on data types involved. The shortest instruction sequence is for the FIXED DECIMAL data type.

SUBSCRIPTRANGE condition
: This generates additional code to verify if all subscripts are within the proper bounds. This code is executed each time a subscripted data item is referenced and may result in some degradation at execution time. For performance sensitive applications NOSUBSCRIPTRANGE is recommended.

SIZE condition
: SIZE checking involves some overhead in storage usage and in execution time. Consider removing this option.

A.4.7 Tuning Expressions

The execution time of the program can be improved by factoring expressions and being careful about the precision and scale factor of variables used in expressions. Be aware when intermediate results and library calls are required and avoid this if possible, by rearranging the expression.

- Avoid the use of mixed mode arithmetic, especially the use of character strings in arithmetic operations. In the following example there is a character variable which has to be regularly incremented by 1.

  ```
  DCL CTLNO   CHAR(8);
  CTLNO = CTLNO + 1 ;
  ```

  This example require two conversions and one library call, while

  ```
  DCL CTLNO   CHAR(8);
  DCL DCTLNO  DECIMAL FIXED(9,0);
  DCTLNO = DCTLNO + 1 ;
  CTLNO = DCTLNO ;
  ```

requires only one conversion and no library calls.

- Do not use operator expressions, variables or function references where constants can be substituted.

  For example:

  ```
  DCL A(8) CHAR(80);
  ```

  is more efficient than

  ```
  DCL A(5+3) CHAR(80);
  ```

  and

  ```
  VERIFY(DATA,'0123456789')....;
  ```

  is more efficient than

  ```
  DCL NUMBERS CHAR(10) STATIC INIT('0123456789');
  VERIFY(DATA,NUMBERS).... ;
  ```

- Avoid declaring strings with a length close to the limit which is 32767 bytes for CHARACTER strings and 32767 bits for BIT strings. Special care should be taken for strings with the VARYING attribute, because the length used in intermediate results is the maximum possible length for a VARYING string, so you may suffer intermediate truncation.

- Avoid concatenation operations on bit strings, because they are time consuming.

- Array arithmetic is a convenient way of specifying an iterative computation. For example:

  ```
  DCL A(10,20);
  A = A / A(1,1);
  ```

  has the same effect as

  ```
  DCL A(10,20);
  DO I = 1 TO 10;
    DO J = 1 TO 20;
    A(I,J) = A(I,J) / A(1,1);
    END;
  END;
  ```

- The compiler is bound by the left to right evaluation rules for arithmetic expressions. In order to get better optimization move all constants and duplicate expressions to the left of the expression or group them in parentheses.

Examine Loops for the Five Inhibitors to Code Optimization: The following example shows an arithmetic expression executed within a loop. EXP and SQRT are PL/I Built-In functions.

```
TEST1: DCL PI DECIMAL FIXED(5,4) INIT(3.1416);
       DCL X(5000) DECIMAL FIXED(5,0);
       N = 10;
       DO I = 1 TO N;
         X(I) = X(I) * EXP(SQRT(PI/2);
         END;
```

We expect the compiler to move the constant part of the expression out of the loop, so the above example will appear as:

```
TEST2:  DCL PI DECIMAL FIXED(5,4) INIT(3.1416);
        DCL X(5000) DECIMAL FIXED(5,0);
        TEMP1 = EXP(SQRT(PI/2);
        N = 10;
        DO I = 1 TO N;
           X(I) = X(I) * TEMP1;
           END;
```

The compiler will move expressions or parts of expressions out of the loop only when all the following conditions are met:

1. The OPTIMIZE(TIME) compile-time option is coded as described in Chapter 3, "Run-time Options and Their Effects"
2. The REORDER option is specified for the block containing loop (described in previous sections).
3. Variables in the expression do not have the BASED attribute as described in Chapter 3, "Run-time Options and Their Effects"
4. The expression does not reference user-defined functions. The compiler will not remove from loops either user defined functions or irreducible PL/I Built-In functions. A function is reducible when all repeated evaluations with the same argument values produce exactly the same result. In the above example the Built-In functions EXP and SQRT are reducible. An example of an irreducible Built-In function is TIME. All user defined functions are irreducible, because the optimizer has no way of predicting any of their side effects.
5. The expression is moveable. Lines of code are moveable when the following criteria are met:
   a. they are always executed in the loop
   b. the result is not set elsewhere in the loop
   c. the code involves only loop invariants

To illustrate the performance gain by moving code out of the loop the results immediately following were obtained for the previous examples.

	NOOPTIMIZE		OPTIMIZE(TIME)	
EXAMPLE	ORDER	REORDER	ORDER	REORDER
TEST1	6.5	6.5	6.2	1.0
TEST2	1.3	1.3	1.0	1.0

Figure 35. A Comparison of Performance Options. The values are ratios relative to TEST1 with OPT(2) and REORDER.

A.4.8 Tuning Data Types and Data Structures

You should examine data structures and data types and ensure that they are both appropriate for the selected calculation. If the wrong data structure or type is used then performance can degrade significantly.

Structures and Arrays

```
EXAMPL1:DCL 1 A,
              2 NAME(1000) CHARACTER(15),
              2 NUMBER(1000) FIXED DECIMAL(5,0);
        DO I = 1 TO N ;
           IF (A.NAME(I) = ... & A.NUMBER(I) > ...) THEN ... ;
           END;

EXAMPL2:DCL 1 A(1000),
              2 NAME   CHARACTER(15),
              2 NUMBER FIXED DECIMAL(5,0);
        DO I = 1 TO N ;
           IF (A.NAME(I) = ... & A.NUMBER(I) > ...) THEN ... ;
           END;
```

Figure 36. A Structure Optimization Example. In EXAMPL1 A.NAME and A.NUMBER are not adjacent in storage.

Care should be taken that items which are accessed together are also stored together as is shown in EXAMPL2.

Based and Defined Attributes

The next example shows how different data attributes can affect performance at execution time.

```
.
EXAMPL3::DCL A(10)  CHAR(100)STATIC,
            B(10)  CHAR(100)STATIC,
            C(10)  CHAR(100)BASED(ADDR(b));
            D(10)  CHAR(100)DEFINED B ;
        DO I = 1 TO 10;
            A = ¢¢;          <===  6 instructions in loop
            B = ¢¢;          <=== 10 instructions in loop
            C = ¢¢;          <=== 11 instructions in loop
            D = ¢¢;          <===  6 instructions in loop
            END;
```

Figure 37. The Effect of Data Attributes on Performance. EXAMPL3 was executed with the LIST and NOOPTIMIZE compiler options.

The previous example showed that using the DEFINED attribute generates less instructions than the BASED attribute. Consequently use of the DEFINED attribute is recommended.

A characteristic of the hardware architecture is that some data types have different performance in computations than others.

Picture Data Type

You should avoid using PICTURE data items for computations, because additional overhead is required to convert data item before and after the computation. In some cases conversion is done by a call to a library subroutine which is obviously more costly in resource use than using in-line instructions. On the other hand PICTURE data items are better for computational operations than CHARACTER data items.

Fixed Decimal and Fixed Binary Data Type

You should ensure that loop indexes, counters and subscripts are defined as being the FIXED BINARY data type. Normally variables with names beginning with the letters I to N are given the attributes REAL FIXED BINARY(15,0) and those with names beginning with any other alphabetic character are given the attributes REAL FLOAT DECIMAL(6).

```
EXAMPL4::DCL  A FIXED ;      ==>  defaults to DECIMAL FIXED(5,0)
        DCL  B ;            ==>  defaults to DECIMAL FLOAT(6)
        DCL  I ;            ==>  defaults to BINARY FIXED(15,0)
        DCL  J BINARY(15);  ==>  defaults to BINARY FLOAT(15)
        DCL  K BINARY(15,0);==>  defaults to BINARY FIXED(15,0)
```

Figure 38. Default Attributes for Various Data Types

Automatic and Static Attributes

Variables with the AUTOMATIC attribute are allocated on each entry to the block where they are declared. You should check whether initialization is necessary and if so that it is done efficiently. A variable gets the AUTOMATIC attribute by default.

Variables with the STATIC attribute specified are allocated prior to running a program and form part of the STATIC CSECT of the compiled program. The storage they require increases the cataloged program length.

Aligned and Unaligned Attributes

The ALIGNED attribute specifies that data is aligned on a storage boundary depending on the data type.

The UNALIGNED attribute specifies that data is aligned on the next byte boundary (fixed length bit strings are mapped on next bit).

The default for bit, character, graphic and character numeric data is UNALIGNED. For all other types it is ALIGNED.

If bit string data whose length is not a multiple of 8 is to be held in structures, you should declare the structures ALIGNED.

From the performance view, operations with ALIGNED data items result in less instructions and no library calls, but need more storage for alignment of data items. UNALIGNED data uses less storage, but for operations

upon it the compiler generates extra instructions and even library calls.

A.4.9 Tuning the Type and Use of Data Files

- When using VSAM files, increase the number of data buffers for sequential access by increasing BUFND, and increase the number of index buffers for random access by increasing the BUFNI value.
- Select the appropriate control interval size (CISZ) for the application. A smaller CISZ results in faster retrieval for random processing at the expense of inserts, while a larger CISZ is more efficient for sequential processing.
- For better performance, access records sequentially and avoid using alternate indexes when possible.

  The ENVIRONMENT options BUFND, BUFNI and the SKIP option which specifies that the VSAM OPTCD ″SKP″ is to be used wherever possible are options used to optimize VSAM′s performance.

  BUFSP can also be used for controlling the total buffer space allocation.
- For record oriented transmission you should allocate sufficient buffers to prevent the program to become I/O bound, and use blocked output records.
- Consider using the BUFFERED attribute for DIRECT files (it is the default for SEQUENTIAL data files) which results in the overlapping of data transmission with processing. The BUFFERS(n) option can also speed up processing of SEQUENTIAL files with ″n″ defaulting to 2, which means two buffers are reserved for the file.
- For stream oriented transmission use edit-directed input/output in preference to list- or data-directed input/output.
- In STREAM input/output, do as much as possible in each input/output statement. For example:

  ```
  PUT EDIT ((A(I) DO I = 1 TO 10)) (... ;
  ```

  is more efficient than:

  ```
  DO I = 1 TO 10 ;
    PUT EDIT ((A(I) DO I = 1 TO 10)) (... ;
    END ;
  ```
- Care should be taken when using the PUT DATA statement without a data-list, because in that case all variables known in the block are printed and the CONVERSION condition can be raised for FIXED DECIMAL data.
- Avoid using the TITLE option, because it requires closing and reopening the file whenever the statement, which deals with multiple files, accesses a new data set. A file variable should be used instead.
- The OPEN statement is executed by library routines that are loaded dynamically. Execution time can be improved if more than one file is specified in single OPEN statement. The same is valid also for the CLOSE statement.

A.4.10 Examining Program′s Run-time Behavior

To monitor a program′s run-time behavior the CEEBINT user exit is used. CEEBINT is a HLL user exit (written in PL/I or LE/VSE conforming assembler) called during enclave initialization which lets you perform tasks such as recording accounting information or calling other user exits. LE provides an object module dummy version of CEEBINT that consists of an immediate return to the application. The CEEBINT module can be linked with initialization/termination library routines or with an application program. However if you want it included with an application program you must explicitly include it at link edit time.

The following callable services are provided to assist you in tuning:

IBMBHKS can be used to turn hooks on and off prior to execution

IBMBSIR is a Static Information Retrieval module which lets you determine static information about PL/I modules compiled with the TEST option

IBMBHIR is a HOOK Information Retrieval module which lets you determine static information about hooks executed in modules compiled with the TEST option

The TEST compile time option specifies the level of testing capability that the compiler generates as part of the object code and is described in previous sections.

The data you obtain by this means can be used to further tune your program. A detailed description of the process can be found in:

SC26-8053 PL/I VSE Programming Guide (chapter 9: Examining and Tuning)

and

SC26-8065 LE/VSE Programming Guide, which gives a detailed description of user exits and callable services.

Appendix B. Full Text of SDL Load Jobs

Where the size of the job stream made it necessary to simply list the phases loaded, the actual jobstream is included here to allow you to copy from the softcopy documentation to an ICCF or CMS library member.

B.1 Job to Load All SVA-Eligible 24-bit PL/I Modules

```
* $$ JOB JNM,SDLPL24,DISP=D,CLASS=0
// JOB LOAD Pl/I 24-BIT MODULES INTO SVA
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.SCEECICS
SET SDL
IBMRCCLA,SVA
IBMRCCRA,SVA
IBMRCOMP,SVA
IBMRKMRA,SVA
IBMRLANA,SVA
IBMRLANN,SVA
IBMRLANU,SVA
IBMRLIB1,SVA
IBMRMCTA,SVA
IBMROCAA,SVA
IBMROPAA,SVA
IBMROPEA,SVA
IBMROPZA,SVA
IBMRPDBA,SVA
IBMRRAAA,SVA
IBMRRABA,SVA
IBMRRACA,SVA
IBMRRADA,SVA
IBMRRAEA,SVA
IBMRRAFA,SVA
IBMRRAGA,SVA
IBMRRAHA,SVA
IBMRRBAA,SVA
IBMRRBBA,SVA
IBMRRBEA,SVA
IBMRRBFA,SVA
IBMRRCAA,SVA
IBMRRCBA,SVA
IBMRRCCA,SVA
IBMRRDAA,SVA
IBMRRDBA,SVA
IBMRRDCA,SVA
IBMRRDDA,SVA
IBMRREAA,SVA
IBMRRECA,SVA
IBMRREEA,SVA
IBMRREFA,SVA
IBMRRQAA,SVA
IBMRRQBA,SVA
IBMRRQCA,SVA
IBMRRQEA,SVA
IBMRRQFA,SVA
IBMRRQGA,SVA
IBMRRVAA,SVA
```

```
IBMRRVGA,SVA
IBMRRVHA,SVA
IBMRRVIA,SVA
IBMRSOFA,SVA
IBMRSOUA,SVA
IBMRSOVA,SVA
IBMRSTFA,SVA
IBMRSTIA,SVA
IBMRSTUA,SVA
IBMRSTVA,SVA
/*
/&
* $$ EOJ
```

B.2 Job to Load All SVA-Eligible 31-bit LE Modules

```
* $$ JOB JNM=SDLPL24,DISP=D,CLASS=0
// JOB LOAD PL/I 31-BIT MODULES INTO SVA
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.SCEECICS
SET SDL,SVA
CEE5ABD,SVA
CEE5ADDM,SVA
CEE5CTY,SVA
CEE5DMP,SVA
CEE5GRC,SVA
CEE5GRN,SVA
CEE5LNG,SVA
CEE5MCS,SVA
CEE5MDS,SVA
CEE5MTS,SVA
CEE5PRM,SVA
CEE5RPH,SVA
CEE5SPM,SVA
CEE5SRC,SVA
CEE5USR,SVA
CEECMI,SVA
CEECNTY,SVA
CEECOPP,SVA
CEECRHP,SVA
CEECZST,SVA
CEEDATE,SVA
CEEDATM,SVA
CEEDAYS,SVA
CEEDCOD,SVA
CEEDSHP,SVA
CEEDYWK,SVA
CEEFMDA,SVA
CEEFMDT,SVA
CEEFMTM,SVA
CEEFRST,SVA
CEEGMT,SVA
CEEGMTO,SVA
CEEGPID,SVA
CEEGQDT,SVA
CEEGTST,SVA
CEEHDLR,SVA
CEEHDLU,SVA
CEEISEC,SVA
CEEITOK,SVA
CEEKDS,SVA
CEELOCT,SVA
CEEMENU0,SVA
CEEMENU2,SVA
CEEMGET,SVA
CEEMMS,SVA
CEEMOUT,SVA
CEEMRCR,SVA
CEEMSG,SVA
CEEMUEN0,SVA
CEEMUEN2,SVA
CEEMUEN3,SVA
CEENCOD,SVA
CEEOPMF,SVA
```

```
CEEPARM,SVA
CEEPLPKA,SVA
CEEQCEN,SVA
CEEQMATH,SVA
CEEQUMF,SVA
CEERAN0,SVA
CEESCEN,SVA
CEESDABS,SVA
CEESDACS,SVA
CEESDASN,SVA
CEESDAT2,SVA
CEESDATH,SVA
CEESDATN,SVA
CEESDCOS,SVA
CEESDCSH,SVA
CEESDCTN,SVA
CEESDDIM,SVA
CEESDERC,SVA
CEESDERF,SVA
CEESDEXP,SVA
CEESDGMA,SVA
CEESDINT,SVA
CEESDLG1,SVA
CEESDLG2,SVA
CEESDLGM,SVA
CEESDLOG,SVA
CEESDMOD,SVA
CEESDNIN,SVA
CEESDNWN,SVA
CEESDSGN,SVA
CEESDSIN,SVA
CEESDSNH,SVA
CEESDSQT,SVA
CEESDTAN,SVA
CEESDTNH,SVA
CEESDXPD,SVA
CEESDXPI,SVA
CEESEABS,SVA
CEESEATH,SVA
CEESEATN,SVA
CEESECI,SVA
CEESECJG,SVA
CEESECOS,SVA
CEESECS,SVA
CEESECSH,SVA
CEESEDVD,SVA
CEESEEXP,SVA
CEESEIMG,SVA
CEESELOG,SVA
CEESEMLT,SVA
CEESESIN,SVA
CEESESNH,SVA
CEESESQT,SVA
CEESETAN,SVA
CEESETNH,SVA
CEESEXPE,SVA
CEESEXPI,SVA
CEESGL,SVA
CEESGLT,SVA
```

```
CEESIABS,SVA
CEESIDIM,SVA
CEESIMOD,SVA
CEESISGN,SVA
CEESIXPI,SVA
CEESQABS,SVA
CEESQACS,SVA
CEESQASN,SVA
CEESQAT2,SVA
CEESQATH,SVA
CEESQATN,SVA
CEESQCOS,SVA
CEESQCSH,SVA
CEESQCTN,SVA
CEESQDIM,SVA
CEESQERC,SVA
CEESQERF,SVA
CEESQEXP,SVA
CEESQINT,SVA
CEESQLG1,SVA
CEESQLG2,SVA
CEESQLOG,SVA
CEESQMOD,SVA
CEESQSGN,SVA
CEESQSIN,SVA
CEESQSNH,SVA
CEESQSQT,SVA
CEESQTAN,SVA
CEESQTNH,SVA
CEESQXP2,SVA
CEESQXPI,SVA
CEESQXPQ,SVA
CEESRABS,SVA
CEESRATH,SVA
CEESRATN,SVA
CEESRCJG,SVA
CEESRCOS,SVA
CEESRCSH,SVA
CEESRDVD,SVA
CEESREXP,SVA
CEESRIMG,SVA
CEESRLOG,SVA
CEESRMLT,SVA
CEESRSIN,SVA
CEESRSNH,SVA
CEESRSQT,SVA
CEESRTAN,SVA
CEESRTNH,SVA
CEESRXPI,SVA
CEESRXPR,SVA
CEESSABS,SVA
CEESSACS,SVA
CEESSASN,SVA
CEESSAT2,SVA
CEESSATH,SVA
CEESSATN,SVA
CEESSCOS,SVA
CEESSCSH,SVA
CEESSCTN,SVA
```

```
CEESSDIM,SVA
CEESSERC,SVA
CEESSERF,SVA
CEESSEXP,SVA
CEESSGMA,SVA
CEESSINT,SVA
CEESSLG1,SVA
CEESSLG2,SVA
CEESSLGM,SVA
CEESSLOG,SVA
CEESSMOD,SVA
CEESSNIN,SVA
CEESSNWN,SVA
CEESSSGN,SVA
CEESSSIN,SVA
CEESSSNH,SVA
CEESSSQT,SVA
CEESSTAN,SVA
CEESSTNH,SVA
CEESSXPI,SVA
CEESSXPS,SVA
CEESTABS,SVA
CEESTATH,SVA
CEESTATN,SVA
CEESTCJG,SVA
CEESTCOS,SVA
CEESTCSH,SVA
CEESTDVD,SVA
CEESTEXP,SVA
CEESTIMG,SVA
CEESTLOG,SVA
CEESTMLT,SVA
CEESTSIN,SVA
CEESTSNH,SVA
CEESTSQT,SVA
CEESTTAN,SVA
CEESTTNH,SVA
CEESTXPI,SVA
CEESTXPT,SVA
CEEURTB,SVA
CEEUTC,SVA
CEEYDTS,SVA
/*
/&
* $$ EOJ
```

Glossary

abend. Abnormal end of application.

active routine. The currently executing routine.

additional heap. A Language Environment heap created and controlled by a call to CEECRHP. See also below heap, anywhere heap, and initial heap.

American National Standard Code for Information Interchange (ASCII). The code developed by the American National Standards Institute (ANSI) for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

anywhere heap. The LE/VSE heap controlled by the ANYHEAP run-time option. It contains library data, such as LE/VSE control blocks and data structures not normally accessible from user code. The anywhere heap may reside above 16MB. See also below heap, additional heap, initial heap.

application development life cycle. The sequence of activities performed during application development, from enterprise modeling and validation, requirements analysis and application design, to system development, test, production, and maintenance.

application generator. An application development tool that creates applications, application components (panels, data, databases, logic, interfaces to system services), or complete application systems from design specifications.

application program. A collection of software components used to perform specific types of work on a computer, such as a program that does inventory control or payroll.

argument. An expression used at the point of a call to specify a data item or aggregate to be passed to the called routine.

ASCII. American National Standard Code for Information Interchange.

assembler. see High Level Assembler.

automatic data. Data that does not persist across calls to other routines. Automatic data may be automatically initialized to a certain value upon entry and reentry to a routine.

automatic storage. Storage that is allocated on entry to a routine or block and is freed on the subsequent return. Sometimes referred to as stack storage or dynamic storage.

below heap. The LE/VSE heap controlled by the BELOWHEAP run-time option, which contains library data, such as LE/VSE control block and data structures not normally accessible from user code. Below heap always resides below 16MB. See also anywhere heap, initial heap, additional heap. breakpoint. A place in a program, usually specified by a command or a condition, where execution may be interrupted and control given to the workstation user or to a specified debug tool program.

by reference. See pass by reference.

by value. See pass by value.

byte. The basic unit of storage addressability, usually with a length of 8 bits.

callable services. A set of services that can be invoked by a Language Environment-conforming high level language using the conventional Language Environment-defined call interface, and usable by all programs sharing the Language Environment conventions. Use of these services helps to decrease an application§s dependence on the specific form and content of the services delivered by any single operating system.

CASE. Computer-aided software engineering.

CICS. Customer Information Control System.

COBOL. COmmon Business-Oriented Language. A high level language, based on English, that is primarily used for business applications.

COBOL run unit. A COBOL-specific term that defines the scope of language semantics. Equivalent to a Language Environment enclave.

compilation unit. An independently compilable sequence of HLL statements. Each HLL product has different rules for what makes up a compilation unit. Synonymous with program unit.

computer-aided software engineering (CASE). A software engineering discipline for automating the application development process and thereby improving the quality of application and the productivity of application developers.

condition. An exception that has been enabled, or recognized, by Language Environment and thus is eligible to activate user and language condition handlers. Any alteration to the normal programmed flow of an application. Conditions can be detected by

© Copyright IBM Corp. 1995

the hardware/operating system and result in an interrupt. They can also be detected by language-specific generated code or language library code.

condition handler. A user-written condition handler or language-specific condition handler (such as a PL/I ON-unit) invoked by the LE/VSE condition manager to respond to conditions.

condition manager. Manages conditions in the common execution environment by invoking various user-written and language-specific condition handlers. broad range of callable services.

condition token. In Language Environment, a data type consisting of 96 bits (12 bytes). The condition token contains structured fields that indicate various aspects of a condition including the severity, the associated message number, and information that is specific to a given instance of the condition.

cross-system consistency. Consistency of interfaces across different systems. Cross-system consistency relates to portability of applications to different platforms; that is, the application writer sees consistent support on all of the supported platforms relative to standard HLL source statements and a broad range of callable services.

Customer Information Control System (CICS). CICS is an OnLine Transaction Processing (OLTP) system that provides specialized interfaces to databases, files and terminals in support of business and commercial applications.

data type. The properties and internal representation that characterize data.

DBCS. Double-byte character set.

default. A value that is used when no alternative is specified.

DOS PL/I. See PL/I.

double-byte character set (DBCS). A collection of characters represented by a two-byte code.

DSA. Dynamic storage area.

dynamic call. A call that results in the resolution of the called routine at run time. Contrast with static call.

dynamic storage. Storage acquired as needed at run time. Contrast with static storage.

EBCDIC. Extended binary-coded decimal interchange code.

enablement. The determination by a language at run time that an exception should be processed as a condition. This is the capability to intercept an exception and to determine whether it should be ignored or not; unrecognized exceptions are always defined to be enabled. Normally, enablement is used to supplement the hardware for capabilities that it does not have and for language enforcement of the language§s semantics. An example of supplementing the hardware is the specialized handling of floating-point overflow exceptions based on language specifications (on some machines this can be achieved through masking the exception).

enclave. In Language Environment, an independent collection of routines, one of which is designated as the main routine. An enclave is roughly analogous to a program or run unit.

enterprise. The composite of all operational entities, functions, and resources that form the total business concern.

environment. A set of services and data available to a program during execution. In Language Environment, environment is normally a reference to the run-time environment of HLLs at the enclave level.

exception. The original event such as a hardware signal, software detected event, or user-signaled event which is a potential condition. This action may or may not include an alteration in a program′s normal flow. See also condition.

execution time. Synonym for run time.

execution environment. Synonym for run-time environment.

(EBCDIC)

extended binary-coded decimal interchange code. A set of 256 eight-bit characters.

external data. Data that persists over the lifetime of an enclave and maintains last-used values whenever a routine within the enclave is reentered. Within an enclave consisting of a single phase, it is equivalent to COBOL external data.

external routine. A procedure or function that may be invoked from outside the program in which the routine is defined.

feedback code (fc). A condition token value. If you specify fc in a call to a callable service, a condition token indicating whether the service completed successfully is returned to the calling routine.

file. A named collection of related data records that is stored and retrieved by an assigned name.

filename. A 1- to 7-character name used within an application and in JCL to identify a file. The filename

provides the means for the logical file to be connected to the physical file.

fix-up and resume. The correction of a condition by changing the argument or parameter and running the routine again.

Fortran. A high level language primarily designed for applications involving numeric computations.

function. A routine that is invoked by coding its name in an expression. The routine passes a result back to the invoker through the routine name.

handler. See condition handler.

heap. An area of storage used for allocation of storage whose lifetime is not related to the execution of the current routine. The heap consists of the initial heap segment and zero or more increments. See anywhere heap, below heap, initial heap, and additional heap.

heap element. A contiguous area of storage allocated by a call to the CEEGTST service. Heap elements are always allocated within a single heap segment.

heap increment. See increment.

heap segment. A contiguous area of storage obtained directly from the operating system. The Language Environment storage management scheme subdivides heap segments into individual heap elements. If the initial heap segment becomes full, Language Environment obtains a second segment, or increment, from the operating system.

heap storage. See heap.

High Level Assembler. An IBM licensed program. Translates symbolic assembler language into binary machine language.

high level language (HLL). A programming language above the level of assembler language and below that of program generators and query languages.

HLL. High level language.

ILC. Interlanguage communication.

increment. The second and subsequent segments of storage allocated to the stack or heap.

indirect parameter passing. Placing an address in a parameter list. In other words, passing a pointer to a value instead of passing the value itself.

initial heap. The Language Environment heap controlled by the HEAP run-time option and designated by a heap_id of 0. The initial heap contains dynamically allocated user data. See also additional heap.

initial heap segment. The first heap segment. A heap consists of the initial heap segment and zero or more additional segments or increments.

initial stack segment. The first stack segment. A stack consists of the initial stack segment and zero or more additional segments or increments.

instance specific information (ISI). Located within the LE/VSE condition token, the ISI contains information used by the condition manager to identify and react to a specific occurrence of a condition. interlanguage communication (ILC). The ability of routines written in different programming languages to communicate. ILC support allows the application writer to readily build applications from component routines written in a variety of languages. interrupt. A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.

ISI. Instance specific information.

KSDS.. Key sequenced data sets. See VSAM.

Language Environment. A set of architectural constructs and interfaces that provides a common run-time environment and run-time services to applications compiled by Language Environment-conforming compilers.

Language Environment/VSE. An IBM software product that provides a common run-time environment and common run-time services for conforming language compilers.

library. A collection of functions, subroutines, or other data.

LIFO. Last in, first out method of access. A queuing technique in which the next item to be retrieved is the item most recently placed in the queue.

local data. Data that is known only to the routine in which it is declared. Equivalent to working storage in COBOL.

main program. The first routine in an enclave to gain control from the invoker.

multitasking. See multithreading.

multithreading. Mode of operation that provides for the concurrent, or interleaved, execution of two or more tasks, or threads.

national language support. Translation requirements affecting parts of licensed programs; for example, translation of message text and conversion of symbols specific to countries.

object code. Output from a compiler or assembler which is itself executable machine code or is suitable for processing to produce executable machine code.

object deck. Synonymous with object module.

object module. A portion of an object program suitable as input to a linkage editor. Synonymous with object deck.

online. (1) Pertaining to a user's ability to interact with a computer. (2) Pertaining to a user's access to a computer via a terminal.

operating system.. Software that controls the running of programs; in addition, an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

parameter. Data items that are received by a routine.

Pascal. A high level language for general purpose use. Programs written in Pascal are block structured, consisting of independent routines.

pass by reference. In programming languages, one of the basic argument passing semantics. The address of the object is passed. Any changes made by the callee to the argument value will be reflected in the calling routine at the time the change is made. pass by value. In programming languages, one of the basic argument passing semantics. The value of the object is passed. Any changes made by the callee to the argument value will not be reflected in the calling routine.

percolate. The action taken by the condition manager when the returned value from a condition handler indicates that the handler could not handle the condition, and the condition will be transferred to the next handler.

phase. An application or routine in a form suitable for execution. The application or routine has been compiled and link-edited; that is, address constants have been resolved.

PL/I. A general purpose scientific/business high level language. It is a high-powered procedure-oriented language especially well suited for solving complex scientific problems or running lengthy and complicated business transactions and record-keeping applications.

pointer. A data element that indicates the location of another data element.

procedure. A named block of code that can be invoked, usually via a call. In Language Environment, the term routine is used as generic for a procedure or a function.

process. The highest level of the Language Environment program management model. A process is a collection of resources, both program code and data, and consists of at least one enclave.

program. See application program.

program management. The functions within the system that provide for establishing the necessary activation and invocation for a program to run in the applicable run-time environment when it is called.

program unit. Synonym for compilation unit.

programmable workstation (PWS). A workstation that has some degree of processing capability and that allows a user to change its functions. promote. To change a condition. A condition is promoted when a condition handling routine changes the condition to a different one. A condition handling routine promotes a condition because the error needs to be handled in a way other than that suggested by the original condition.

PWS. Programmable workstation.

register. To specify formally. In Language Environment, to register a condition handler means to add a user-written condition handler onto a routine's stack frame.

resume. To begin execution in an application at the point immediately after which a condition occurred. A resume occurs when the condition manager determines that a condition has been handled and normal application execution should continue.

resume cursor. Designates the point in the application where a condition occurred when it is first reported to the condition manager. The resume cursor also designates the point where execution resumes after a condition is handled, usually at the instruction in the application immediately following the point at which the error occurred. The resume cursor can be moved with the CEEMRCR callable service.

return code. A code produced by a routine to indicate its success. It may be used to influence the execution of succeeding instructions.

routine. In Language Environment, refers to a procedure, function, or subroutine.

run. To cause a program, utility, or other machine function to be performed.

run time. Any instant at which a program is being executed. Synonymous with execution time.

run-time environment. A set of resources that are used to support the execution of a program. Synonymous with execution environment.

run unit. One or more object programs that are executed together. In Language Environment, a run unit is the equivalent of an enclave.

safe condition. Any condition having a severity of 0 or 1. Such conditions are ignored if no condition handler handles the condition.

SBCS. Single-byte character set.

scope. A term used to describe the effective range of the enablement of a condition and/or the establishment of a user-generated routine to handle a condition. Scope can be both statically and dynamically defined. The portion of an application within which the definition of a variable remains unchanged.

segment. See stack segment.

single-byte character set (SBCS). A collection of characters represented by a 1-byte code.

source code. The input to a compiler or assembler, written in a source language.

source program. A set of instructions written in a programming language that must be translated to machine language before the program can be run.

stack. An area of storage used for suballocation of stack frames. Such suballocations are allocated and freed on a LIFO (last in, first out) basis. A stack is a collection of one or more stack segments consisting of an initial stack segment and zero or more increments.

stack frame. The physical representation of the activation of a routine. The stack frame is allocated on a LIFO stack and contains various pieces of information including a save area, condition handling routines, fields to assist the acquisition of a stack frame from the stack, and the local, automatic variables for the routine. In LE/VSE, a stack frame is synonymous with DSA.

stack increment. See increment.

stack segment. A contiguous area of storage obtained directly from the operating system. The Language Environment storage management scheme subdivides stack segments into individual DSAs. If the initial stack segment becomes full, a second segment or increment is obtained from the operating system.

stack storage. See stack and automatic storage.

static call. A call that results in the resolution of the called program statically at link-edit time. Contrast with dynamic call.

static data. Data that retains its last-used state across calls.

static storage. Storage that persists and retains its value across calls. Contrast with dynamic storage.

subsystem. A secondary or subordinate system, or programming support, usually capable of operating independently of or asynchronously with a controlling system. Example: CICS.

syntax. The rules governing the structure of a programming language and the construction of a statement in a programming language.

thread. The basic run-time path within the Language Environment program management model. It is dispatched by the system with its own instruction counter and registers. The thread is where actual code resides.

token. See condition token.

user-written condition handler. A routine established by the CEEHDLR callable service to handle a condition or conditions when they occur in the common run-time environment. A queue of user-written condition handlers established by CEEHDLR may be associated with each stack frame in which they are established.

vendor. A person or company that provides a service or product to another person or company.

VSAM. Virtual storage access method. A high-performance mass storage access method. Three types of data organization are available: entry sequenced data sets (ESDS), key sequenced data sets (KSDS), and relative record data sets (RRDS).

workstation. One or more programmable or nonprogrammable devices that allow a user to do work on a computer. See also programmable workstation.

List of Abbreviations

IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
CEL	common execution library
CICS	Customer Information Control System
CSD	CICS system definition
CSECT	control section
DB2	Data Base 2
DCL	declare
DL/I	Data Language 1
DOS	Disk Operating System
DOS/VS	Disk Operating System/Virtual Storage
DSA	dynamic storage area
GT	greater than
HLASM	High Level Assembler
HLL	high level language
HWM	high water mark
ISA	initial storage area
JCL	job control language
LE	Language Environment
LIFO	last in first out
MB	megabyte(s)
PTF	program temporary fix
SDL	system directory list
SQL	Structured Query Language
SQL/DS	Structured Query Language/Data System
SVA	shared virtual area
UK	United Kingdom
VDISK	virtual disk
VM	virtual machine
VSAM	Virtual Storage Access Method
VSCR	virtual storage constraint relief
VSE	Virtual Storage Extended
VSE/ESA	Virtual Storage Extended/Enterprise System Architecture

© Copyright IBM Corp. 1995

Index

Special Characters

*PROCESS 39
%PROCESS 39

A

abbreviations 79
acronyms 79
ALIGNED 63
ALL31 30
AMODE 23
ANYHEAP 25
ANYWHERE
  HEAP 26
  STACK 24
attributes
  ALIGNED 63
  automatic 63
  based 62
  defined 62
  REENTRANT 58
  REORDER 58
  static 63
  tuning 58
  UNALIGNED 63
AUTOMATIC storage 48

B

BASED storage 48
BEGIN blocks 58
BELOW
  HEAP 26
  STACK 24
BELOWHEAP 25

C

CEECOPT 42
CEEDOPT 41
CEEUOPT 43
checklist 51
  run-time options 51
  SVA 51
  tuning 55
compile-time
  optimization 53
compiler
  EXEC SIZE 53
  partition size 53
  use of VDISK 54
condition
  NOFIXEDOVERFLOW 59
  SIZE 59
condition *(continued)*
  SUBSCRIPTRANGE 59
CONTROLLED storage 48

D

data tuning 62
data type
  binary fixed 63
  decimal fixed 63
  PICTURE 63
DB2 36
definitions
  data 3
    automatic 3
    external 3
    local 3
  enclave 2
  heap 5
  heap element 5
  heap increment 5
  heap segment 5
  process 3
  routine 2
  stack 4
  stack frame 4
  thread 2
DL/I 35

E

expressions 59
ezBRIDGE 36

F

fixed binary 63
fixed decimal 63
FREE
  HEAP 26
  STACK 25

G

global
  optimization 54
glossary 73
GOTO statements 58

H

Heap
  ANYWHERE 26
  BELOW 26
  FREE 26

© Copyright IBM Corp. 1995

Heap *(continued)*
KEEP 26
Residence 25
run-time option
tuning 25
use by PL/I 26

I

IEL1OPTL 39
IEL1OPTV 37

K

KEEP
HEAP 26
STACK 25

L

limitations
load modules 31
object modules 31
load modules
limitations 31

M

management models
program 2
storage 4
messages 57
monitoring 65
MQSeries 36

N

NOFIXEDOVERFLOW 59

O

object modules
limitations 31
ON UNITS 59
optimization
global 54
limitations 55
options
compile-time 54, 56, 57
default 37
defining 37, 41
determining 37
effect on run-time 39
effect on storage 39
how to decide 41
in *PROCESS 39
in %PROCESS 39
in JCL 39
OPTIMIZE 54, 56
ORDER 57
selection 56
options *(continued)*
compile-time *(continued)*
STORAGE 49
SYSTEM 23
TEST 56
run-time 51
ALL31 30
by assembler exit 48
by CEECOPT 46
by CEEUOPT 43, 46
by PARM 45
by PLIXOPT 46, 48
checklist 51
deciding 48
determining 37
global 41, 42
HEAP 26
ISASIZE 31
RPTOPTS 27
RPTSTG 28, 49
STACK 24
STORAGE 27
other subsystems 35

P

PARM
compiler 39
run-time 45
performance
checklist 55
monitoring 65
non-storage 53
PICTURE 63
PL/I
heap 26
stack 24
prefix
condition 59
label 59
prefixes and labels 59
pricing 5
PROCEDURE size 59
Program
Residence 23

R

REENTRANT 58
REORDER 58
Residence
heap 25
Program 23
Stack 24
RMODE 23
RPTOPTS 27
RPTSTG 28
run-time
PARM 45

Run-time options 23

S

shared virtual area
  initial 7
size 59
  partition 53
  PROCEDURE 59
  program area 53
SIZE condition 59
SQL/DS 36
Stack
  ANYWHERE 24
  BELOW 24
  FREE 25
  KEEP 25
  Residence 24
  run-time option
  Tuning 24
  use by PL/I 24
STATIC storage 48
storage
  ANYHEAP 25
  AUTOMATIC 48
  BASED 48
  BELOWHEAP 25
  CONTROLLED 48
  efficient use 55
  run-time 57
    tuning 57
  STATIC 48
storage use
  compiler 37
  load modules 31, 32, 33
    batch 32
    CICS 33
sub-systems
  DB2 36
  DL/I 35
  ezBRIDGE 36
  MQSeries 36
  other 35
  SQL/DS 36
SUBSCRIPTRANGE 59
SVA
  all eligible modules
    LE 24-bit 15
    LE 31-bit 17
    PL/I 24-bit 19
    PL/I 31-bit 20
  checklist 51
  eligibility
    compiler 7
    LE services 8
    PL/I services 8
  enlargement 9
  guidance 22
  initial 7
SVA *(continued)*
  recommended modules
    LE 24-bit 10
    LE 31-bit 11
    PL/I 24-bit 14
    PL/I 31-bit 12
  sizes 21, 22
    all eligible phases 21
    differences 22
    recommended 21
SYSTEM
  CICS 24
  VSE 23

T

Tuning
  arrays 62
  attributes 58
  based 62
  data 62
  expressions 59
  file types 64
  files 64
  heap 25
  Stack 24
  storage 57
  structures 62

U

UNALIGNED 63

V

VDISK 54

W

warnings 57

ITSO Technical Bulletin Evaluation RED000

International Technical Support Organization
31-bit Addressing in PL/I for VSE
Getting Started
January 1995

Publication No. GG24-4271-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____

Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

____

____

What other topics would you like to see in this Bulletin?

____

____

What other Technical Bulletins would you like to see published?

____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

______________________ ______________________
Name Address

______________________ ______________________
Company or Organization

______________________ ______________________
Phone No.

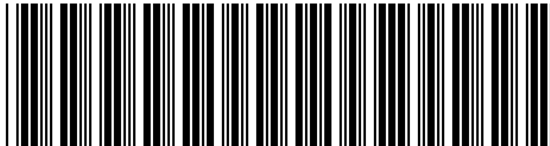
Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM International Technical Support Organization
Department 3222, Building 71032-02
POSTFACH 1380
71032 BOEBLINGEN
GERMANY

Fold and Tape **Please do not staple** Fold and Tape

Printed in U.S.A.



Artwork Definitions

id	File	Page	References
ITSLOGO	4271SU	i	i

Table Definitions

id	File	Page	References
TBLALL0	4271CH03	30	30

Figures

id	File	Page	References
RESOWN	4271CH01	4	1
PRICE	4271CH01	6	2 5
DEFSVA	4271CH02	8	3 7, 15
BIGSVA	4271CH02	9	4 15
JOBL24R	4271CH02	10	5 10, 11, 12, 14
L24RSVA	4271CH02	10	6 10
JOBL31R	4271CH02	11	7 11, 12, 14
L31RSVA	4271CH02	11	8 11
JOBP31R	4271CH02	12	9 12, 14
P31RSVA	4271CH02	13	10 12
JOBP24R	4271CH02	14	11 14
P24RSVA	4271CH02	14	12 14
JOBL24	4271CH02	15	13 15, 17, 19, 20
L24SVA	4271CH02	16	14 15
JOBL31	4271CH02	17	15 17, 19, 20
L31SVA	4271CH02	18	16 17
JOBP24	4271CH02	19	17 19, 20
P24SVA	4271CH02	19	18 19
JOBP31	4271CH02	20	19 20
P31SVA	4271CH02	21	20 20
OPTV	4271CH05	38	23 37

id	File	Page	References
OPTL	4271CH05	39	24 38
COMPSAM	4271CH05	40	25 40
CEEDOPT	4271CH05	42	26 41
CEECOPT	4271CH05	43	27 42
CEWUOPT	4271CH05	45	28 44
PARMEX	4271CH05	46	29 46
PLIXOPT	4271CH05	46	30
COMCICS	4271CH05	47	31 46
STOR1	4271CH05	49	33 49
STOR2	4271CH05	49	34 49

Headings

id	File	Page	References
NOTICES	4271FM	xiii	Special Notices ii
BIBL	4271PREF	xvi	Related Publications
INTRO	4271CH01	1	Chapter 1, Introduction xv
SVAUSE	4271CH02	7	Chapter 2, SVA Eligibility and Residence xv
RUNTIME	4271CH03	23	Chapter 3, Run-time Options and Their Effects xv, 49, 55, 57, 57, 57, 57, 57, 61, 61
STACKO	4271CH03	24	3.2.4, STACK Run-time Option 24
OTHSS	4271CH04	35	Chapter 4, Interfacing with Other Subsystems xv
WHATRT	4271CH05	37	Chapter 5, How to Determine Compile-time and Run-time Options xv, 51, 53, 55, 57
CTOE	4271CH05	39	5.2.4, Compile-time Option Effects on Storage Requirements and Run Time 41
SUMMARY	4271CH06	51	Chapter 6, A Check-list for Initial Implementation xv
PERF	4271AX01	53	Appendix A, Performance Considerations xv
GLOBOPT	4271AX01	54	A.2, The Global Optimization Features 56
HILIT2	4271AX01	55	A.3, Storage Efficiency
HILIT3	4271AX01	55	A.4, Performance Tuning Checklist
HILIT4	4271AX01	56	A.4.1, Obtaining the Listings Needed for Tuning
HILIT5	4271AX01	56	A.4.2, Selection of Compile-time Options
HILIT6	4271AX01	57	A.4.4, Analysis of Compiler Messages Affecting Performance
HILIT7	4271AX01	59	A.4.6, Tuning Statement Prefixes and Labels
HILIT8	4271AX01		

		59	A.4.7, Tuning Expressions
HILIT9	4271AX01	62	A.4.8, Tuning Data Types and Data Structures
HILIT0	4271AX01	64	A.4.9, Tuning the Type and Use of Data Files
HILITA	4271AX01	65	A.4.10, Examining Program's Run-time Behavior
SOURCE	4271AX02	67	Appendix B, Full Text of SDL Load Jobs xv
B1	4271AX02	67	B.1, Job to Load All SVA-Eligible 24-bit PL/I Modules 19, 20
B2	4271AX02	69	B.2, Job to Load All SVA-Eligible 31-bit LE Modules 17, 19, 20

Tables

id	File	Page	References
RECTAB	4271CH02	21	1
TOTTAB	4271CH02	21	2
DELTAB	4271CH02	22	3
TBLALL	4271CH03	30	4
COMTBL	4271CH03	32	5
COMTBLC	4271CH03	33	6
PARMRT	4271CH05	45	7
RECTBL	4271CH05	50	8
MSGTAB	4271AX01	58	9 41

Processing Options

Runtime values:

- Document fileid .... GG244271 SCRIPT
- Document type .... USERDOC
- Document style .... IBMXAGD
- Profile EDFPRF30
- Service Level 0029
- SCRIPT/VS Release 4.0.0
- Date 95.01.19
- Time 05:31:14
- Device 3820A
- Number of Passes 4
- Index .... YES
- SYSVAR D .... YES
- SYSVAR G .... INLINE
- SYSVAR V .... ITSCEVAL

Formatting values used:

- Annotation .... NO
- Cross reference listing .... YES
- Cross reference head prefix only .... NO
- Dialog .... LABEL
- Duplex .... YES
- DVCF conditions file .... (none)
- DVCF value 1 .... (none)
- DVCF value 2 .... (none)
- DVCF value 3 .... (none)
- DVCF value 4 .... (none)
- DVCF value 5 .... (none)
- DVCF value 6 .... (none)
- DVCF value 7 .... (none)
- DVCF value 8 .... (none)
- DVCF value 9 .... (none)
- Explode .... NO
- Figure list on new page .... YES
- Figure/table number separation .... YES
- Folio-by-chapter .... NO

Head 0 body text .... Part
Head 1 body text .... Chapter
Head 1 appendix text .... Appendix
Hyphenation .... NO
Justification .... NO
Language .... ENGL
Layout .... OFF
Leader dots .... YES
Master index .... (none)
Partial TOC (maximum level) 4
Partial TOC (new page after) .... INLINE
Print example id's .... NO
Print cross reference page numbers .... YES
Process value .... (none)
Punctuation move characters ....,
Read cross-reference file .... (none)
Running heading/footing rule .... NONE
Show index entries .... NO
Table of Contents (maximum level) 3
Table list on new page .... YES
Title page (draft) alignment .... RIGHT
Write cross-reference file .... (none)

Imbed Trace

Page	File
Page 0	4271SU
Page 0	4271VARS
Page 0	4271FM
Page i	4271EDNO
Page ii	4271ABST
Page xiii	4271SPEC
Page xiii	4271TMKS
Page xiv	4271PREF
Page xvii	4271ACKS
Page xviii	4271CH01
Page 6	4271CH02
Page 22	4271CH03
Page 33	4271CH04
Page 36	4271CH05
Page 50	4271CH06
Page 52	4271AX01
Page 65	4271AX02
Page 72	4271GLOS
Page 77	4271ABRV
Page 83	4271EVAL
Page 83	RCFADDR
Page 83	ITSCADDR FILE
Page 85	RCFADDR
Page 85	ITSCADDR FILE