IBM Language Environment for VSE/ESA

Concepts Guide

C33-6680-00

First Edition (December 1996)

This edition applies to Version 1 Release 4 ofIBM Language Environment for VSE/ESA , 5686-094, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

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

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation
Attn: Dept. ECJ - BP/003D
6300 Diagonal Highway
Boulder, CO 80301,
U.S.A.

or to:

IBM Deutschland Entwicklung GmbH
Department 3282
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of 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.

Contents

Figures v

Tables vii

Notices ix

Programming Interface Information xi

Trademarks xiii

About This Book xv
- What Is LE/VSE? xv
- LE/VSE-Conforming Languages xv
  - LE/VSE Compatibility with Previous Versions of COBOL xvi
  - Using Your Documentation xvi
  - Terms Used in This Book xvii

Summary of Changes xix
- Major Changes to the Product xix
  - Release 4, December 1996 xix

Chapter 1. Overview 1
- What You Can Do with LE/VSE 3
  - Common Use of System Resources Gives You Greater Control 3
  - Consistent Condition Handling Simplifies Error Recovery 3
  - LE/VSE Protects Your VS COBOL II Programming Investment 3
  - Enhanced Interlanguage Communication Ensures Application Flexibility 3
  - Common Dump Puts All Debugging Information in One Place 4
  - Locale Callable Services Enhance the Development of Internationalized Applications 4
  - Support For Advanced Debugging 4

Chapter 2. The Model for Language Environment 5
- Language Environment Program Management Model 5
  - Language Environment Program Management Model Terminology 5
- Program Management 6
- Processes 7
- Enclaves 8
  - Characteristics of the Enclave 8
- Threads 8
- Language Environment Storage Management Model 10
  - Stack Storage 10
  - Heap Storage 10
- Language Environment Condition Handling Model 11
  - Condition Handling Terminology 12
  - Condition Handling Model Description 12
  - How Conditions are Represented 14
  - How Condition Tokens are Created and Used 14
  - Condition Handling Responses 15

Run-Time Dump Service Provides Information in One Place....16
Language Environment Message Handling Model and National Language Support....16
National Language Support....16

Chapter 3. LE/VSE Callable Services....17
LE/VSE Calling Conventions....17
Invoking Callable Services from C....17
Invoking Callable Services from COBOL....18
Invoking Callable Services from PL/I....18
Invoking Callable Services from Assembler....18
LE/VSE Callable Services....19

Chapter 4. LE/VSE Run-Time Options....27

Chapter 5. Sample Routines....29
Sample Assembler Routine....29
Sample C Routine....29
Sample COBOL Routine....31
Sample PL/IPL/I Routine....33

Chapter 6. Product Requirements....35
Machine Requirements....35
Programming Requirements....35
Required Licensed Programs....35
Optional Licensed Programs....35
Compatibility Considerations....36

Bibliography....37
Language Environment Publications....37
LE/VSE-Conforming Language Product Publications....37
Softcopy Publications....38
Language EnvironmentLanguage Environment Glossary....39

Index....49

Figures

1. Components of LE/VSE....2
2. LE/VSE's Common Run-Time Environment.... 2
3. Language Environment Resource Ownership....7
4. Language Environment Program Management.... 9
5. Language Environment Heap Storage.... 11
6. Condition Handling Stack Configuration....13
7. Language Environment Condition Handling.... 15
8. Sample Invocation of a Callable Service from C.... 17
9. Omitting the Feedback Code when Calling a Service from C....18
10. Sample Invocation of a Callable Service from COBOL....18
11. Sample Invocation of a Callable Service from PL/I....18
12. Omitting the Feedback Code when Calling a Service from PL/I.... 18
13. Sample Invocation of a Callable Service from Assembler....19
14. Omitting the Feedback Code when Calling a Service from Assembler.... 19
15. A Simple Main Assembler Routine.... 29
16. Sample C Routine....30
17. Sample COBOL Routine.... 32
18. Sample PL/I Routine.... 34

Tables

1. LE/VSE-Conforming Languages..... xvi

2. How to Use LE/VSE and Language Publications.....xvi

3. LE/VSE Callable Services..... 19

4. LE/VSE Run-Time Options.....27

5. Required Licensed Programs for LE/VSE.....35

6. Optional Licensed Compiler Programs for LE/VSE.....35

7. Optional Licensed Programs for LE/VSE.....35

Notices

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 that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

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:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Programming Interface Information

This book is intended to help with application programming. This book documents General-Use Programming Interface and Associated Guidance Information provided by IBM Language Environment for VSE/ESA.

General-Use programming interfaces allow the customer to write programs that obtain the services of IBM Language Environment for VSE/ESA.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AD/Cycle	DFSORT	SAA
BookManager	IBM	SQL/DS
C/370	Integrated Language Environment	System/370
COBOL/370	Language Environment	VM/ESA
CICS	MVS/ESA	VSE/ESA
CICS/VSE	Operating System/400	

About This Book

This book introduces you to the Language Environment architecture as implemented on the VSE platform by IBM Language Environment for VSE/ESA (LE/VSE).

The book contains an overview of LE/VSE, descriptions ofLE/VSE 's full program model, callable services, run-time options, software and hardware requirements, and a glossary of LE/VSE terms. This is not a programming manual, but rather a conceptual introduction to LE/VSE.

Concepts Guide should be read by those who design systems installations and develop application programs. This high-level guide will show how best to plan for systems to support your enterprise.

Terms that may be new to you are *italicized* on their first use. Definitions of these terms can be found in “Language EnvironmentLanguage Environment Glossary” on page 39.

What Is LE/VSE?

LE/VSE is a set of common services and language-specific routines that provide a single run-time environment for applications written in *LE/VSE-conforming* versions of the C, COBOL, and PL/I high level languages (HLLs), and for many applications written in previous versions of COBOL. (For a list of LE/VSE-conforming languages, and a description of compatibility with previous versions of COBOL, see “LE/VSE Compatibility with Previous Versions of COBOL” on page xvi.) LE/VSE also supports applications written in assembler language using LE/VSE-provided macros and assembled using High Level Assembler (HLASM).

Prior to LE/VSE, each programming language provided its own separate run-time environment. LE/VSE combines essential and commonly-used run-time services—such as message handling, condition handling, storage management, date and time services, and math functions—and makes them available through a set of interfaces that are consistent across programming languages. With LE/VSE, you can use one run-time environment for your applications, regardless of the application's programming language or system resource needs, because most system dependencies have been removed.

Services that work with only one language are available within language-specific portions of LE/VSE.

LE/VSE consists of:

- Basic routines for starting and stopping programs, allocating storage, communicating with programs written in different languages, and indicating and handling error conditions.
- Common library services, such as math services and date and time services, that are commonly needed by programs running on the system. These functions are supported through a library of callable services.
- Language-specific portions of the common run-time library.

LE/VSE is the implementation of Language Environment on the VSE platform. Language Environment is offered on two other platforms: on MVS and VM as IBM Language Environment for MVS & VM, and on OS/400 as Integrated Language Environment.

LE/VSE-Conforming Languages

An LE/VSE-conforming language is any HLL that adheres to the LE/VSE common interface. Table 1 on page xvi lists the LE/VSE-conforming language compiler products you can use to generate applications that run with LE/VSE Release 4.

Table 1. LE/VSE-Conforming Languages

Language	LE/VSE-Conforming Language	Minimum Release
C	IBM C for VSE/ESA	Release 1
COBOL	IBM COBOL for VSE/ESA	Release 1
PL/I	IBM PL/I for VSE/ESA	Release 1

Any HLL not listed in Table 1 on page xvi is known as a *non-LE/VSE-conforming* or, alternatively, a *pre-LE/VSE-conforming* language. Some examples of non-LE/VSELE/VSE-conforming languages are: C/370,DOS/VS COBOL , VS COBOL II, and DOS PL/I.

Only the following products can generate applications that run withLE/VSE :

- LE/VSE-conforming languages
- HLASM using LE/VSE-provided macros (for details, see *LE/VSE Programming Guide*)
- DOS/VS COBOL and VS COBOL IIVS COBOL II, with some restrictions (see "LE/VSE Compatibility with Previous Versions of COBOL" on page xvi)

LE/VSE Compatibility with Previous Versions of COBOL

Although DOS/VS COBOL and VS COBOL II are non-LE/VSE-conforming languages, many applications generated with these compilers can run with LE/VSE without recompiling or relink-editing. For details about compatibility, see LE/VSE Run-Time Migration Guide.

VS COBOL II can also dynamically call some LE/VSE date and time callable services. For details, see LE/VSE Programming Reference.

Using Your Documentation

The publications provided with LE/VSELE/VSE are designed to help you:

- Manage the run-time environment for applications written inLE/VSE -conforming languages.
- Write applications that use the LE/VSE callable services.
- Develop interlanguage communication (ILC) applications.
- Plan for, install, customize, and maintain LE/VSE.
- Debug problems in your LE/VSE-conforming applications.
- Migrate your high-level language applications to LE/VSE.

Language programming information is provided in the high-level language programming manuals that provide language definition, library function syntax and semantics, and programming guidance information.

Each publication helps you perform a different task. For a complete list of publications you might need, see "Bibliography" on page 37.

Table 2. How to Use LE/VSE and Language Publications

To...	Use...	
Evaluate LE/VSE	*LE/VSE Fact Sheet* *LE/VSE Concepts Guide*	GC33-6679 GC33-6680
Plan for, install, customize, and maintain LE/VSE	*LE/VSE Installation and Customization Guide*	SC33-6682
Understand the LE/VSE program models and concepts	*LE/VSE Concepts Guide* *LE/VSE Programming Guide*	GC33-6680 SC33-6684

Table 2. How to Use LE/VSE and Language Publications (continued)

To...	Use...	
Find syntax for LE/VSE run-time options and callable services	LE/VSE Programming Reference	SC33-6685
Develop your LE/VSE-conforming applications	*LE/VSE Programming Guide* *LE/VSE C Run-Time Programming Guide* *LE/VSE C Run-Time Library Reference* *Your language programming guide*	SC33-6684 SC33-6688 SC33-6689
Develop interlanguage communication (ILC) applications	*LE/VSE Writing Interlanguage Communication Applications*	SC33-6686
Debug your LE/VSE-conforming application and get details on run-time messages	*lE/VSE Debugging Guide and Run-Time Messages*	SC33-6681
Migrate applications to LE/VSE	*LE/VSE Run-Time Migration Guide* *Your language migration guide*	SC33-6687
Diagnose problems that occur in your LE/VSE-conforming application	*LE/VSE Debugging Guide and Run-Time Messages*	SC33-6681
Understand warranty information	*LE/VSE Licensed Program Specifications*	GC33-6683

Terms Used in This Book

Unless otherwise stated, the following terms are used in this book to refer to the specified languages:

Term...
Refers to the language supported by...

C
The IBM C for VSE/ESA compiler

COBOL
The IBM COBOL for VSE/ESA and VS COBOL II compilers

PL/I
The IBM PL/I for VSE/ESA compilers

For a list of LE/VSE-conforming language compilers, see “LE/VSE-Conforming Languages” on page xv.

Summary of Changes

This section lists the major changes that have been made toLE/VSE since Release 1.

Major Changes to the Product

Release 4, December 1996

LE/VSE Release 4 is a major functional enhancement of LE/VSE Release 1 (program number 5686-067); there were no intermediate releases. This release number was chosen to match IBM Language Environment for MVS & VM Release 4, upon which LE/VSE Release 4 is based.

- C language run-time support has been added for C applications compiled with an LE/VSE-conforming C language compiler.
- Support has been added for interactive and batch-mode debugging of applications using a debug tool such as Debug Tool for VSE/ESA. The TEST run-time option specifies the conditions under which a debug tool assumes control when the user application is being initialized.
- The following run-time options have been added:

  ARGPARSE
  Specifies whether arguments on the command line are to be parsed in the usual C format.

  ENV
  Specifies the operating environment for a C application.

  ENVAR
  Sets the initial values for the environment variables specified. With ENVAR, you can pass into an application switches or tagged information that can then be accessed during application execution using the C functions `getenv()`, `setenv()`, and `clearenv()`.

  EXECOPS
  Specifies whether run-time options can be specified on the command line.

  PLIST
  Specifies the format of the invocation parameters received by your C application when it is invoked.

  REDIR
  Specifies whether redirections for `stdin`, `stderr`, and `stdout` are allowed from the command line.

  TRACE
  Determines whether LE/VSE run-time library tracing is active.
- The CEE5CIB callable service has been added. CEE5CIB returns a pointer to a condition information block (CIB) associated with a given condition token. CEE5CIB is used only during condition handling.
- The CEECBLDY callable service has been added. CEECBLDY converts a string representing a date into a COBOL Integer format that is compatible with ANSI COBOL intrinsic functions.
- LE/VSE locale callable services that access and manage locale information have been added:

  CEEFMON
  Formats monetary strings. CEEFMON corresponds to the C function `strfmon()`.

  CEEFTDS
  Formats time and date into a character string. CEEFTDS corresponds to the C function `strftime()`.

  CEELCNV
  Queries locale numeric conventions. CEELCNV corresponds to the C function `localeconv()`.

  CEEQDTC
  Queries locale date and time conventions and returns the specified format information. CEEQDTC corresponds to the C function `localdtconv()`.

CEEQRYL
Queries the active locale environment. CEEQRYL corresponds to the C function `setlocale()`.

CEESCOL
Compares the collation weight of two strings. CEESCOL corresponds to the C function `strcoll()`.

CEESETL
Sets the locale operating environment. CEESETL corresponds to the C function `setlocale()`.

CEESTXF
Transforms string characters into collation weights. CEESTXF corresponds to the C function `strxfrm()`.

- Predefined locales for specifying different national language and cultural conventions have been added.
- Locale definition utility for modifying and creating locales has been added (requires LE/VSELE/VSE-conforming C language compiler).
- Nested enclaves can now be created by the C `system()` function.

For more information about changes between LE/VSE Release 1 and Release 4, see LE/VSE Run-Time Migration Guide .

Chapter 1. Overview

Today, enterprises need efficient, consistent, and less complex ways to develop quality applications and to maintain their existing inventory of applications. The trend in application development is to modularize and share code, and to develop applications on a workstation-based front end. LE/VSE gives you a common environment for LE/VSE-conforming high-level language (HLL) products. An HLL is a programming language above the level of assembler language and below that of program generators and query languages.

In the past, programming languages have had limited ability to call each other. Typically, when a routine calls and other routine written in a different language, the called routine's environment must be initialized at the time of the call and terminated after the call—a very costly process. This has constrained those who want to use several languages in an application. Programming languages also have had different rules for implementing data structures and condition handling, and for interfacing with system services and library routines.

With LE/VSE, routines call one another within one common run-time environment, regardless of the LE/VSE-conforming HLL they are written in. Routines follow common calling conventions that standardize the way routines call one another and make interlanguage communication (ILC) in mixed-language applications easier, more efficient, and more consistent.

LE/VSE also combines essential run-time services, such as routines for run-time message handling, condition handling, and storage management. All of these services are available through a set of interfaces that are consistent across programming languages. You may either call these interfaces yourself, or use language-specific services that call the interfaces. With LE/VSE, you can use one run-time environment for your applications, regardless of the application's programming language or system resource needs.

LE/VSE consists of:

- Basic routines that support starting and stopping programs, allocating storage, communicating with programs written in different languages, and indicating and handling conditions.
- Common library services, such as math services and date and time services, that are commonly needed by programs running on the system. These functions are supported through a library of callable services.
- Language-specific portions of the run-time library. Because many language-specific routines call LE/VSE services, behavior is consistent across languages.

Figure 1 on page 2 shows the separate components that make up LE/VSE .

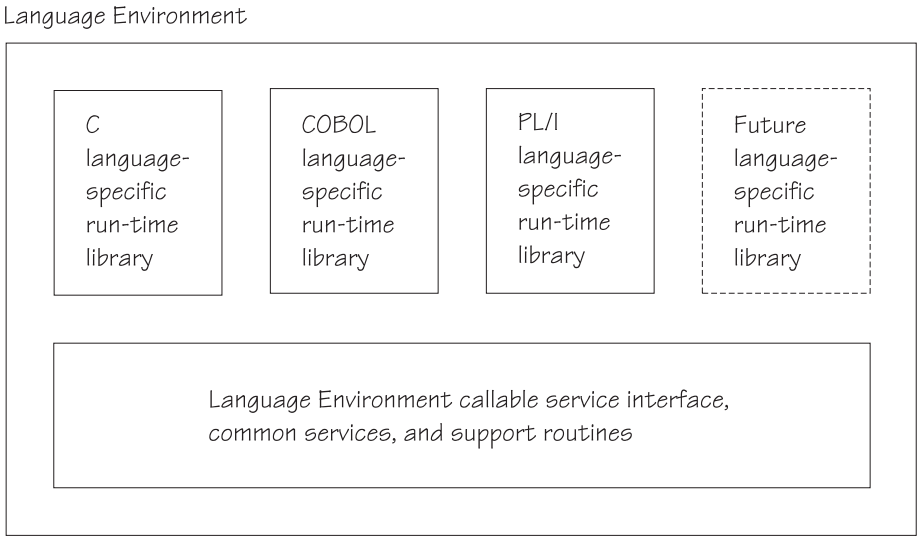

Figure 1. Components of LE/VSE

LE/VSE provides a single run-time environment for applications written in -conforming versions of the C, COBOL, and PL/I HLLs, and for many applications written in previous versions of COBOL. (For a list of LE/VSE-conforming languages, and a description of compatibility with previous versions of COBOL, see "LE/VSE-Conforming Languages" on page xv .) LE/VSE also supports applications written in assembler language usingLE/VSE -provided macros and assembled using HLASM.

For a complete list of operating systems and subsystems supported byLE/VSE , see Chapter 6, "Product Requirements," on page 35.

Figure 2 on page 2 illustrates the common environment that LE/VSE creates.

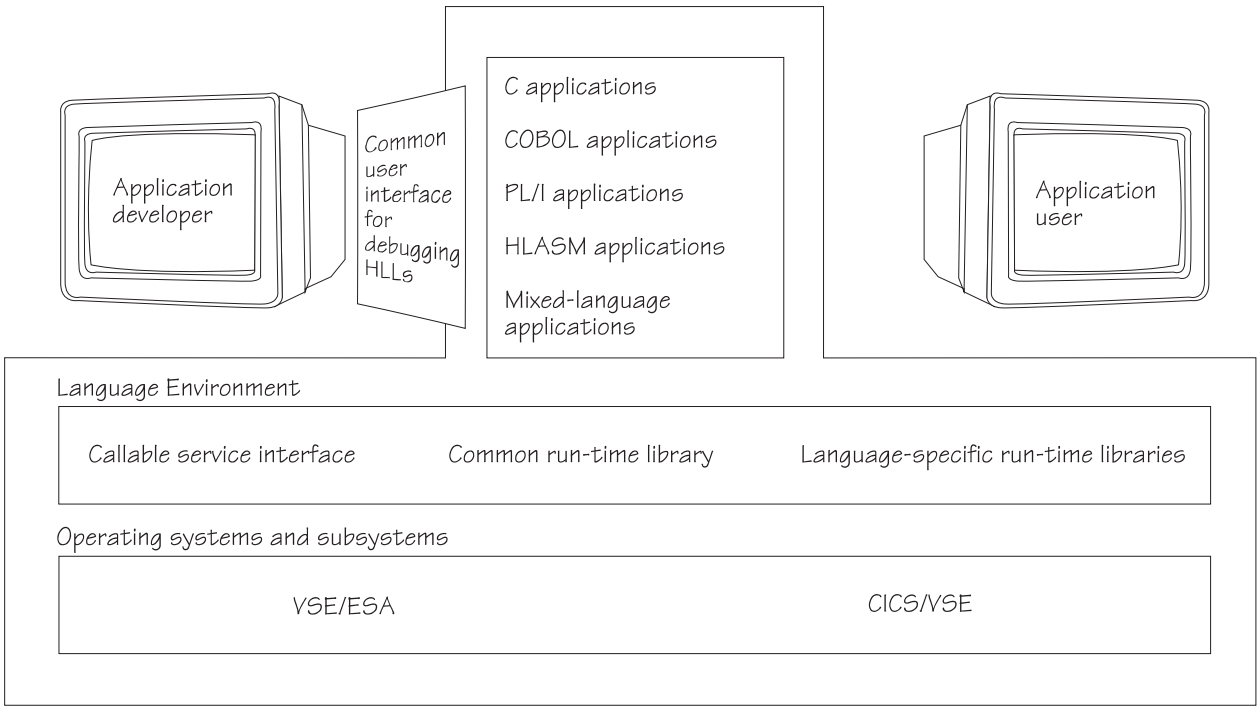

Figure 2. LE/VSE's Common Run-Time Environment

What You Can Do with LE/VSE

LE/VSE helps you create mixed-language applications and gives you a consistent method of accessing common, frequently used services (for example, date and time conversions). Building mixed-language applications is easier with LE/VSE-conforming routines because LE/VSE establishes a consistent environment for all languages in the application.

Common Use of System Resources Gives You Greater Control

LE/VSE provides the base for future IBM language library enhancements in the VSE environment.

Because LE/VSE provides a common library, with services that you can call through a common callable interface, the behavior of your applications will be easier to predict. Many system-dependent services have been removed from LE/VSE-conforming language products. LE/VSE 's common library includes common services such as messages, date and time functions, math functions, application utilities, system services, and subsystem support. The language-specific portions of LE/VSE provide language interfaces and specific services that are supported for each individual language.

LE/VSE services are accessed through defined common calling conventions, described in Chapter 3, “LE/VSE Callable Services,” on page 17.

Consistent Condition Handling Simplifies Error Recovery

LE/VSE establishes consistent condition handling for HLLs and assembler language routines. For languages with little or no condition handling function, like COBOL, LE/VSE provides a user-controlled method for predictable, robust error recovery.LE/VSE condition handling honors single- and mixed-language semantics and is integrated with message handling services to provide you with specific information about each condition.

LE/VSE's language-independent condition handler, unlike some existing HLL condition semantics, is stack frame-based and delivers predictable behavior at a given stack frame. This enables you to construct applications out of building blocks of modules and to control which applications will handle certain conditions.

A complete description of LE/VSE's condition handling model and message services is given in Chapter 2, “The Model for Language Environment,” on page 5.

LE/VSE Protects Your VS COBOL II Programming Investment

LE/VSE provides compatible support for existing VS COBOL II applications. Routines compiled with the new LE/VSE compilers can be mixed with oldVS COBOL II routines in an application. Thus, applications can be enhanced or maintained selectively, without recompiling the whole application when a change is made to a single routine. For mixed-language (COBOL—PL/I —C) applications, you must:

- Recompile any PL/I programs with an LE/VSE-conformingPL/I compiler
- Recompile any C programs with an LE/VSE-conforming C compiler
- Recompile any DOS/VS COBOL programs with an LE/VSE-conforming COBOL compiler
- Link-edit the applications with the LE/VSE run-time library

Some modifications of existing applications may be required. See “Compatibility Considerations” on page 36 for more information.

Enhanced Interlanguage Communication Ensures Application Flexibility

LE/VSE eliminates incompatibilities among language-specific run-time environments. Routines call one another within one common run-time environment, eliminating the need for initialization and termination of a language-specific run-time environment with each call. This makes interlanguage communication (ILC) in mixed-language applications easier, more efficient, and more consistent.

This ILC capability also means that you can share and reuse code easily. You can write a service routine in the language of your choice—C, COBOL, PL/I, or assembler—and allow that routine to be called from C, COBOL, PL/I, or assembler applications. Similarly, vendors can write one application package in the language of their choice, and allow the application package to be called from C, COBOL, PL/I, and assembler routines.

In addition, LE/VSE lets you use the best language for any task. Some programming languages are better suited for certain tasks. LE/VSE 's enhanced ILC allows the best language to be used for any given application task. Many programmers, each experienced in a different programming language, can work together to build applications with component routines written in a variety of languages. LE/VSE 's enhanced ILC allows you to build applications with component routines written in a variety of languages. The result is code that runs faster, is less prone to errors, and is easier to maintain.

Common Dump Puts All Debugging Information in One Place

LE/VSE provides a common dump for all conforming languages. The dump includes, in an easy-to-read format, a description of any relevant conditions and information on error location, variables, and storage.

With a common dump, you can locate precisely in which module an error occurred, saving you many hours spent debugging, especially if your module is built with several languages. A common dump also allows programmers of differing language skills to collaborate effectively in determining the location of a problem that involves modules of different languages.

Locale Callable Services Enhance the Development of Internationalized Applications

Demand is steadily increasing in global markets for software products, and application developers are seeking to make their products available in multiple countries. While marketing their products globally, however, programmers must also make their applications function with the specific language and cultural conventions of the individual user's locale. With locale callable services, application developers can build programs that can be marketed globally, and still meet end users' needs to work with specific languages, cultures, and conventions.

LE/VSE provides pre-defined locales that your C, COBOL, and PL/I routines can access at run time through the locale callable services. You can also create your own locales, or modify the IBM-supplied locales, using the locale definition utility supplied with LE/VSE (this utility requires an LE/VSE-conforming C compiler).

While C routines can use the locale callable services, it is recommended that they use the equivalent native C library services instead for portability across platforms.

For a complete description of LE/VSE locale support, see *C Run-Time Programming Guide* and *Programming Guide*.

Support For Advanced Debugging

Debug Tool for VSE/ESA (packaged with the Full Function Offerings ofC/VSE , COBOL/VSE, and PL/I VSE) is a new feature that requires LE/VSE Release 4. With Debug Tool for VSE/ESA, you can interactively:

- View a program listing while debugging
- Step through execution code
- Set dynamic break points
- Track variables
- Modify program and variable storage during debug
- Debug mixed-language applications
- Develop testing scripts for regression testing

You can use Debug Tool for VSE/ESA to debug applications written in anyLE/VSE -conforming language.

Chapter 2. The Model for Language Environment

This chapter describes the Language Environment architecture, a system of user conventions, product conventions, and processing models that, when followed by HLL application programmers, provides a common, consistent run-time environment. Models for program management, storage management, condition handling, and message services are outlined.

LE/VSE Implementation Information

This release of LE/VSE implements a subset of the Language Environment model. Features not supported in LE/VSE Version 1 Release 4 are clearly indicated in this chapter.

Language Environment Program Management Model

The Language Environment 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 Language Environment architecture. The program management model defines the effects of programming language semantics in mixed-language applications and integrates transaction processing and multithreading.

LE/VSE Implementation Information

Multithreading: Although the Language Environment model supports multithreading, LE/VSE Version 1 Release 4 supports only single threading.

Language Environment 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 a Language Environment context as compared to other contexts. For more detailed definitions of these and other Language Environment terms, please consult the “Language EnvironmentLanguage Environment Glossary” on page 39.

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 Language Environment, normally a reference to the run-time environment of HLLs at the enclave level.

Language Environment Terms and Their HLL Equivalents:

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.

Enclave
The enclave defines the scope of HLL semantics. In Language Environment, 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, C—program, consisting of a main C function and its subfunctions, PL/I—main procedure and its subroutines.

Thread
An execution construct that consists of synchronous invocations and terminations of routines. The thread is the basic run-time path within the Language Environment 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.

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

Equivalent HLL terms: COBOL—program; C—function; PL/I—procedure or begin block.

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 HLL terms:* COBOL—WORKING-STORAGE data items, C—local data, PL/I—data declared with the PL/I INTERNAL attribute.

Program Management

Program management defines the program execution constructs of an application, and the semantics associated with the integration of various components' management of such constructs.

Three entities—the *process, enclave, and thread*—are at the core of the Language Environment program management model. They are described below.

Please refer to Figure 3 on page 7 as you read the following discussion about processes, enclaves, and threads. This figure illustrates the simplest form of the Language Environment program management model, and how resources such as storage are managed.

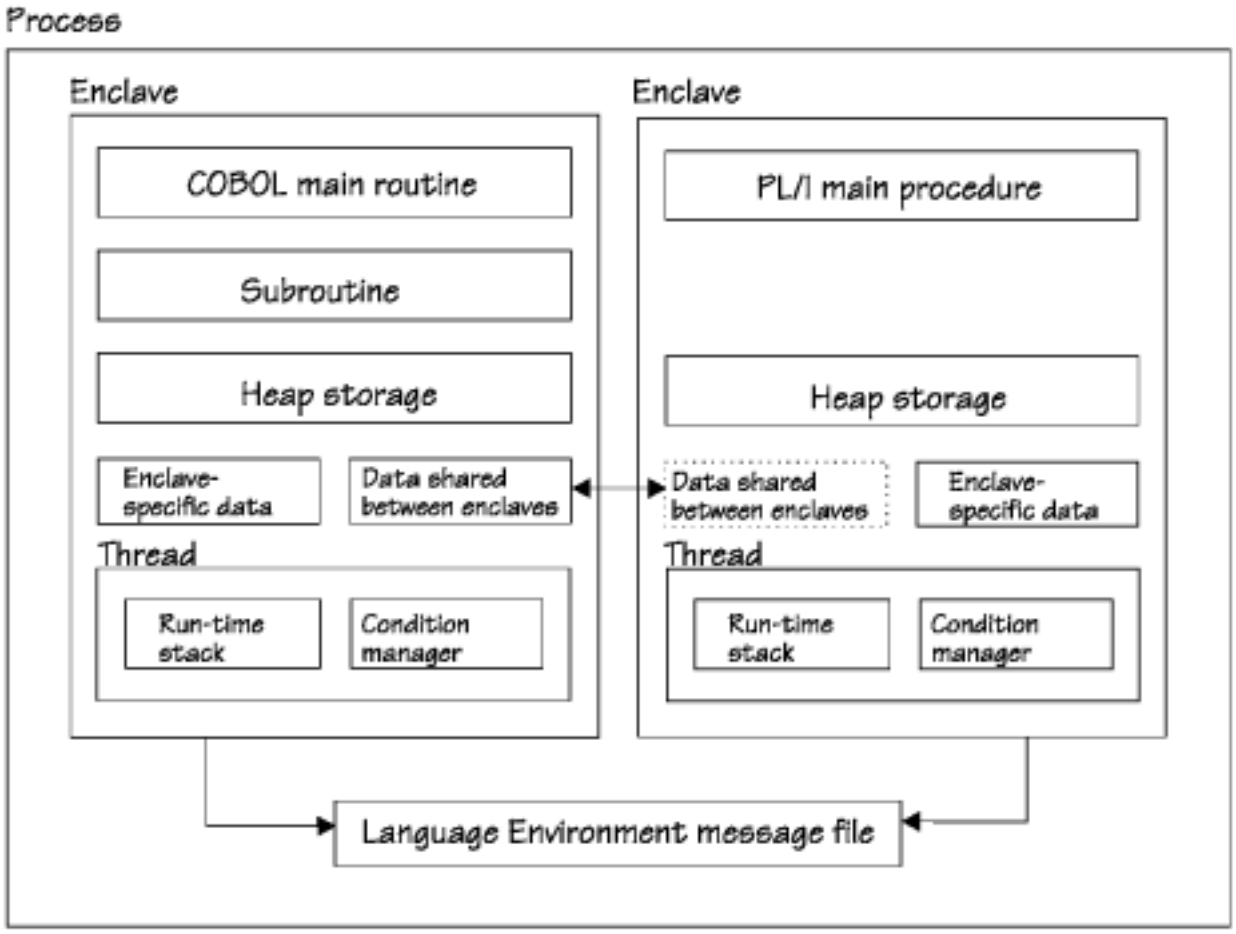

Figure 3. Language Environment Resource Ownership

Processes

The highest level component of the Language Environment program model is the process. A process consists of at least one enclave and is logically separate from other processes. Processes do not share storage and are independent of and equal to each other; they are not hierarchically related.

Language Environment generally does not allow language file sharing across enclaves nor does it provide the ability to access collections of externally stored data. However, the PL/I standard SYSPRINT file may be shared across enclaves. The Language Environment message file also may be shared across enclaves, since it is managed at the process level. The Language Environment message file contains messages from all routines running within a process, making it a useful central location for messages generated during run time.

Processes can create new processes and communicate to each other by usingLanguage Environment - defined communication, for such things as indicating when a created process has been terminated.

LE/VSE Implementation Information

Although the Language Environment model supports applications consisting of one of more processes, LE/VSE Version 1 Release 4 supports only a single process for each application that runs in the common run-time environment.

Enclaves

A key feature of the program management model is the enclave, a collection of the routines that make up an application. As mentioned in the terminology defined above, the enclave is the equivalent of a *run unit* in COBOL, a *program* in C (consisting of a `main()` function and its subfunctions) or a main procedure and all of its subroutines in PL/I.

The enclave consists of one main routine and zero or more subroutines. The main routine is the first to execute in an enclave; all subsequent routines are named as subroutines.

Characteristics of the Enclave

The enclave logically owns resources normally associated with the running of a program. Some resources are owned directly, such as heap storage; some are owned indirectly, such as the run-time stack, which is owned by a thread. Heap storage, the run-time stack, and threads are discussed in the following sections.

Except for the Language Environment message file and PL/I standard SYSPRINT file, the enclave does not share resources with other enclaves. Heap storage is shared among all routines in an enclave and can be allocated by a routine in one language and be freed by a routine in another language. For a discussion on stack and heap storage, see "Language Environment Program Management Model" on page 5.

The enclave defines the scope—how far the semantic effects of language statements reach—of the language semantics for its component routines, just as a COBOL run unit defines the scope of semantics of a COBOL routine.

The enclave defines the following in a Language Environment-conforming application:

- Scope of shared external data, such as COBOL and PL/I external data
- Scope of external files, such as COBOL external files[1]
- Scope of the effect of language statements, for example, STOP-like constructs, such as STOP RUN in COBOL or other terminating mechanisms
- Lifetime of heap storage, in its last-used state

LE/VSE Implementation Information

Multiple Enclaves: Although the Language Environment model supports multiple enclaves within a single process, LE/VSE Version 1 Release 4 provides explicit support for only a single enclave within a single process. Under some circumstances, however, multiple enclaves can exist within a single process. For information on creating multiple, or nested, enclaves, see *Programming Guide*.

Threads

Each enclave consists of at least one thread, the basic instance of a particular routine. A thread is created during enclave initialization with its own run-time stack, which keeps track of the thread's execution, as well as a unique instruction counter, registers, and condition handling mechanisms. Each thread represents an independent instance of a routine running under an enclave's resources.

Threads share all of the resources of an enclave. A thread can address all storage within an enclave. All threads are equal and independent of one another and are not related hierarchically. A thread can create a new enclave. Because threads operate with unique run-time stacks, they can run concurrently within an enclave and allocate and free their own storage. Because they may execute concurrently, threads can be used to implement parallel processing applications and event-driven applications.

Figure 4 on page 9 illustrates the full Language Environment program model, with its multiple processes, enclaves, and threads.

[1] Except for the Language Environment message file, Language Environment provides no support for files that are open under two languages at the same time. You must manage all such files to ensure that no conflicts arise.

As Figure 4 on page 9 shows, each process consists of one or more enclaves. An enclave consists of one main routine, with any number of subroutines.

External data is available only within the enclave where it resides; notice that even though the external data may have identical names in different enclaves, the external data is unique to the enclave. The scope of external data, as described earlier, is the enclave. The threads can create enclaves, which can create more threads, and so on.

LE/VSE Implementation Information

Multiple Threads: Although the Language EnvironmentLanguage Environment model supports multiple threads within an enclave, LE/VSE Version 1 Release 4 supports only a single thread within an enclave.

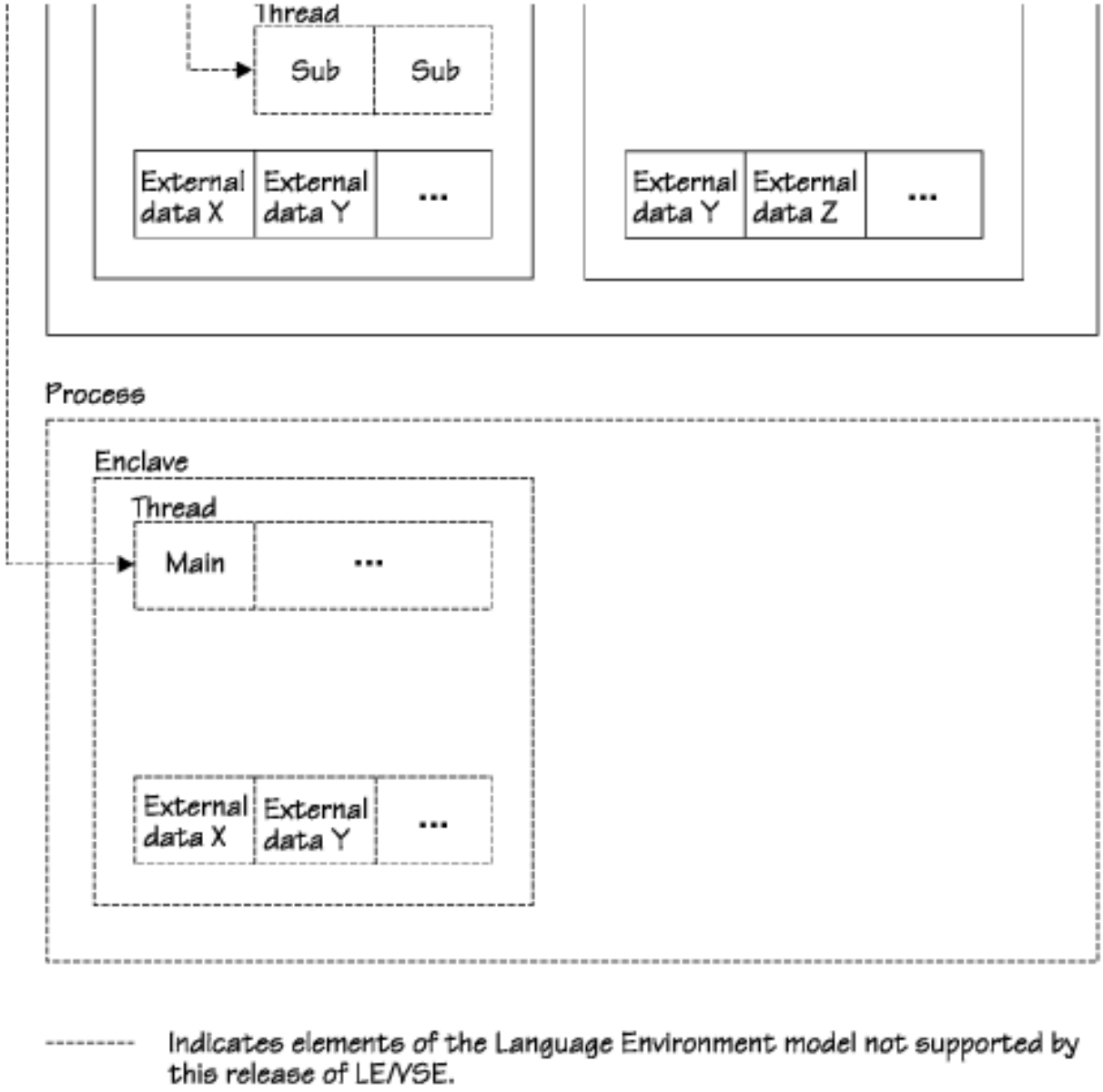

Figure 4. Language Environment Program Management

Language Environment Storage Management Model

Common storage management services are provided for all Language Environment-conforming programming languages;Language Environment 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 (see "Language Environment Condition Handling Model" on page 11 for an explanation of stack frames).

Heap
An area of storage used for allocation of storage whose lifetimes are 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, and storage acquired by `malloc()` and `calloc()` in C.

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.

Stack Storage

A run-time stack, or stack storage, is automatically created when a thread is created, and freed when the thread terminates. When a thread is created, Language Environment allocates an initial stack, which can have stack increments added to it as needed. Users can specify the sizes of the initial stack and additional stack increments; they can also tune the stack for better performance.

Heap Storage

Heap storage can be allocated and freed in any order. (Stack storage, in contrast, is allocated when a routine is entered and freed when the routine ends.)Language Environment provides multiple heaps that may be dynamically created and discarded by using Language Environment callable services.Language Environment 's heap storage is reliable because it provides a level of isolation and prevents common errors such as attempting to free a heap element that has already been freed.

Heap storage is shared among all program units and all threads in an enclave. Allocated heap storage remains allocated until it is explicitly freed by a thread or until the enclave terminates. Heap storage is typically controlled by the programmer through Language Environment run-time options and callable services.

Heap storage consists of an initial heap segment that is allocated when the first heap element is allocated (by a call to CEEGTST). The Language Environment storage manager allocates heap increments as previously allocated segments become full.

Figure 5 on page 11 illustrates heap storage.

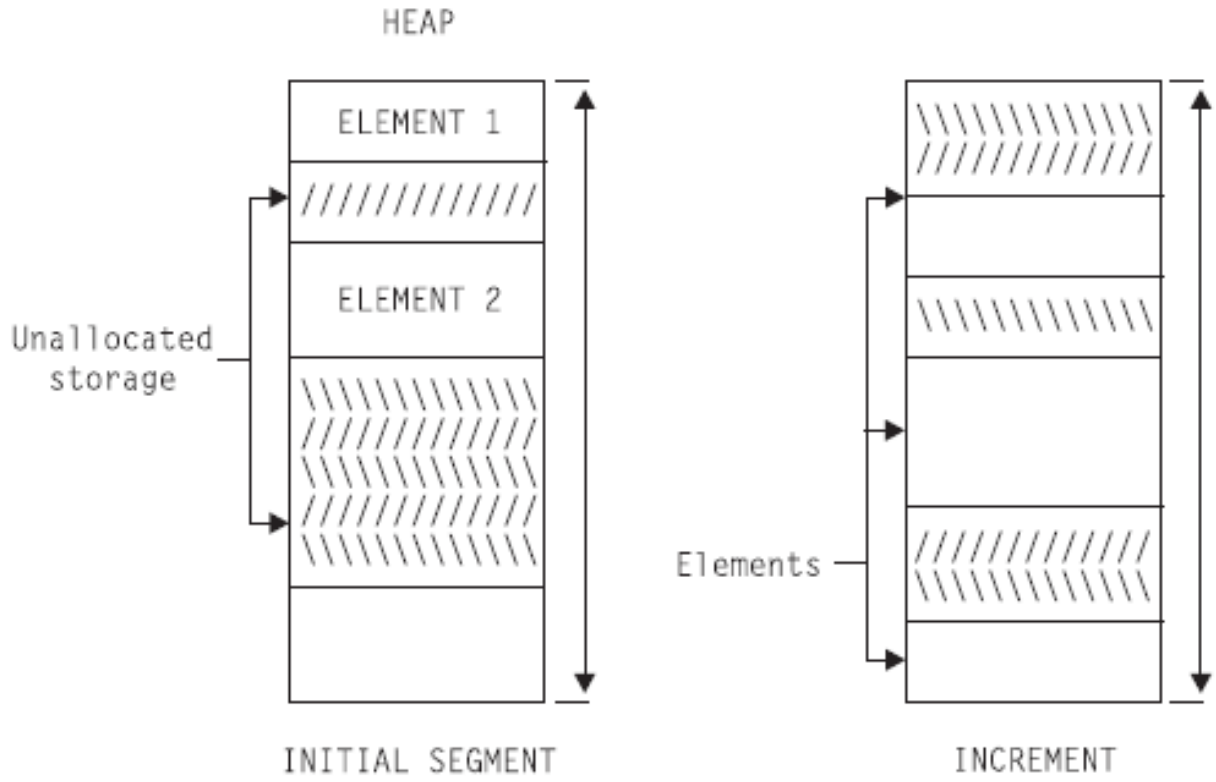

Figure 5. Language Environment Heap Storage

Storage Management Options

Storage Report

You can write a storage report using the run-time option RPTSTG. The report summarizes all heap and stack activity, including total amount of storage used, number of heap elements allocated and freed, number of operating system calls performed, and recommended heap and stack sizes. Proper setting of heap and stack sizes can significantly improve performance by reducing the number of operating system calls made to allocate and free storage.

Storage Option

In Language Environment, the run-time option STORAGE may be used to automatically initialize all heap and stack storage to a specified value. This is useful as a debugging aid to find references to uninitialized program variables.

Language Environment Condition Handling Model

For single- and mixed-language applications, the Language Environment run-time library provides a consistent and predictable condition handling facility. It does not replace current HLL condition handling, but instead allows each language to respond to its own unique environment as well as to a mixed-language environment.

Language Environment condition management gives you the flexibility to respond directly to conditions by providing callable services to signal conditions and to interrogate information about those conditions. It also provides functions for error diagnosis, reporting, and recovery.

Language Environment condition handling is based on the *stack frame*, an area of storage that is allocated when a routine runs and that represents the history of execution of that routine. It can contain automatic variables, information on program linkage and condition handling, and other information. Using the stack frame as the model for condition handling allows conditions to be handled in the stack frame in which

they occur. This allows you to tailor condition handling according to a specific routine, rather than handle every possible condition that could occur within one global condition handler.

A unique feature of Language Environment condition handling is the condition token. The token is a 12-byte data type that contains information about each condition. The information can be returned to the user as a feedback code when calling Language Environment callable services. It can also be used as a communication vehicle within the run-time environment.

Condition Handling Terminology

Below is a list of terms you need to understand while reading the discussion on Language Environment condition handling. For more detailed definitions of these and other Language Environment terms, please consult the “Language EnvironmentLanguage Environment Glossary” on page 39.

Condition
Any change to the normal programmed flow of a program. In Language Environment, a condition can be generated by an event that has historically been called an exception, interruption, or condition.

Condition handler
A routine invoked by Language Environment that responds to conditions in an application. Condition handlers are registered through the CEEHDLR callable service, or provided by the language libraries, by such constructs as PL/I ON statements.

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

Feedback code
A condition token value used to communicate information when using the Language Environment callable services.

Handle cursor
Points to the first condition handler within the stack frame.

Resume cursor
Names the point in the application where execution resumes after a condition is handled. Initially, the resume cursor is positioned after the instruction that caused or signaled the condition.

Stack frame
The physical representation of the activation of a routine. The stack frame is allocated on a last in, first out (LIFO) basis and contains program linkage information, automatic variables, and condition handling routines.

A stack frame is conceptually equivalent to a dynamic save area in PL/I, or a save area in assembler.

For a more detailed description of stack frames, see “Condition Handling Model Description” on page 12.

Condition Handling Model Description

The Language Environment condition handler is based on a stack frame model. A stack frame is an area of storage that can contain automatic variables, information on program linkage and condition handling, and other information. The stack frame is allocated usingLanguage Environment -managed stack storage. It is created through any of the following:

A function call in C
Entry into a compile unit in COBOL
Entry into a procedure or begin block in PL/I
Entry into an ON-unit in PL/I

Each routine adds a unique stack frame, in a LIFO manner, to theLanguage Environment -managed stack storage. User-written condition handlers (registered through CEEHDLR) are associated with each stack

frame. In addition, HLL handling semantics can affect the processing conditions at each stack frame. See Figure 6 on page 13 for an illustration of the Language Environment run-time stack and its divisions into stack frames.

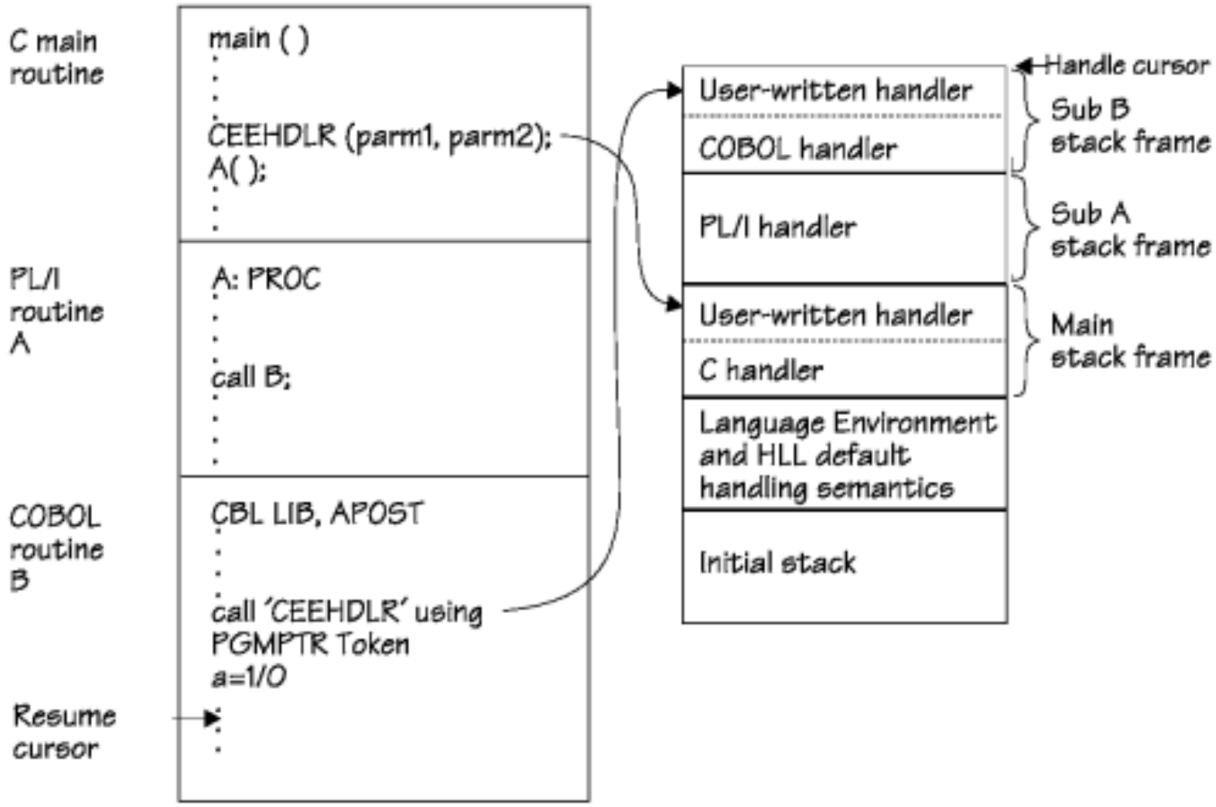

Figure 6. Condition Handling Stack Configuration

Each Language Environment user condition handler is explicitly registered through the callable service CEEHDLR. Language-defined handling mechanisms are registered through language-provided constructs, such as the PL/I ON statement or the C `signal()` function. When a routine returns to its caller, its stack frame is removed from the stack and the associated handlers are automatically unregistered. Semantics associated with a routine are honored; for example, PL/I semantics on a return specify that any ON-units within a routine will be unregistered.

A condition is signaled within Language Environment as a result of one of the following occurrences:

- A hardware-detected interrupt
- An operating system-detected exception
- A condition generated by Language Environment callable services
- A condition explicitly signaled within a routine

The first three types of conditions are managed by Language Environment and signaled if appropriate. The last may be signaled by user-written code through a call to the service CEESGL or signaled by HLL semantics such as SIGNAL in PL/I or `raise` in C.

When a condition is signaled, whether by a user routine, by Language Environment in response to an operating system- or hardware-detected error, or by a callable service, Language Environment directs the appropriate condition handlers in the stack frame to handle the condition. Condition handling proceeds first with user-written condition handlers in the queue, if present, then with any HLL-specific condition handlers, such as a PL/I ON-unit or a C signal handler, that may be established. The process continues for each frame in the stack, from the most recently allocated to the least recently allocated.

If a condition remains unhandled after the stack is traversed, the condition is handled by either Language Environment or by the default semantics of the language where the condition occurred.

How Conditions are Represented

A condition token is used to communicate information about a condition to Language Environment message services, callable services, and routines. The token is a 12-byte data type with fields that indicate the following information about a condition:

- Severity of a condition
- Associated message number
- Facility ID: This field identifies the owner of the condition (Language Environment, Language Environment component, or user-specified). It is also used to identify a file containing message text that is unique for the condition.
- Instance specific information: This field is created if the condition requires that data or text be inserted into a message, for example, a variable name. This field also contains qualifying data, which can be used to specify data (input or output) to be used when a routine resumes processing after a condition occurs.

How Condition Tokens are Created and Used

If the condition is detected by the operating system or by the hardware,Language EnvironmentLanguage Environment will automatically build the condition token and signal the condition. With Language Environment callable services, you can create a condition token with corresponding message or data inserts and then signal the condition to the application running within Language Environment by returning the token.

When used in Language Environment callable services, the entire condition token represents a value called the feedback code. You can include a feedback parameter to Language Environment callable services, and check the result of the call; or, in PL/I and C, you can omit the feedback parameter, and any errors in the call will be signaled to you.

See Figure 7 on page 15 for an illustration of the creation and use of condition tokens.

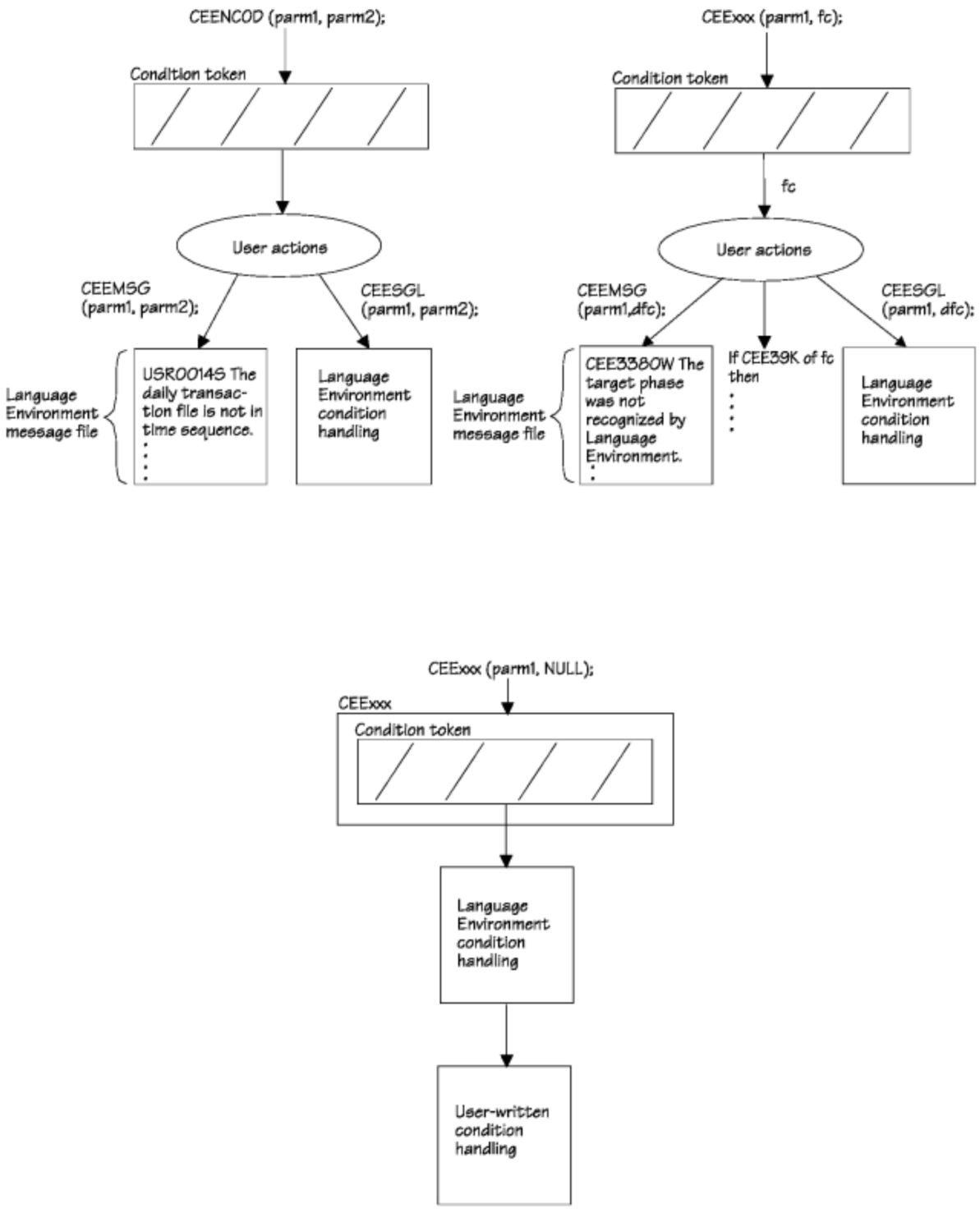

Figure 7. Language Environment Condition Handling

Condition Handling Responses

Conditions are responded to in one of the following ways:

- *Resume* terminates condition handling, and transfers control usually to the location immediately following the point where the condition occurred.

  A resume cursor points to the place where a routine should resume; it can be moved by the callable service CEEMRCR to point to another resume point.

- *Percolate* defers condition handling for an unchanged condition. Condition handling continues at the next condition handler.
- *Promote* is similar to percolate in that it passes the condition on to the next condition handler; however, it transforms a condition to another condition, one with a new meaning. Condition handling then continues, this time with a new type of condition.

Run-Time Dump Service Provides Information in One Place

The Language Environment callable service CEE5DMP dumps the run-time environment of Language Environment into one easily readable report. CEE5DMP can be called directly from an application to produce a dump that is formatted for printing. Depending on the options you choose, the dump report may contain information on conditions, tracebacks, variables, control blocks, stack and heap storage, file status and attributes, and language-specific information. The report can be requested upon program termination due to an unhandled condition by using the run-time option TERMTHDACT.

Language Environment Message Handling Model and National Language Support

A set of common message handling services that create and send run-time informational and diagnostic messages is provided by Language Environment.

With the message handling services, you can use the condition token that is returned from a callable service or from some other signaled condition, format it into a message, and deliver it to a defined output device or to a buffer.

National Language Support

Messages may be formatted according to national language support specifications for the following languages:

- Uppercase American English (UEN)
- Mixed-case American English (ENU)
- Japanese (JPN)

National language support callable services allow you to set a national language that affects the language of the error messages and the names of the day, week, and month. It also allows you to change the country setting, which affects the default date format, time format, currency symbol, decimal separator character, and thousands separator.

Chapter 3. LE/VSE Callable Services

This chapter gives an overview of LE/VSE callable services and the common calling procedure required to invoke them from C, COBOL, PL/I, and assembler.

This common set of callable services is designed to supplement your programming language's intrinsic capability. For example, COBOL application developers will find LE/VSE's consistent condition handling services especially useful. All languages can benefit from the rich set of LE/VSE common math services, as well as the date and time services.

LE/VSE callable services are divided into the following groups:

- Condition Handling Services
- Date and Time Services
- Dynamic Storage Services
- General Callable Services
- Initialization/Termination Services
- Locale Callable Services
- Math Services
- Message Handling Services
- National Language Support Services

Language-specific services, including those that callLE/VSE callable services, are documented in the language manuals.

LE/VSE Calling Conventions

LE/VSEservices can be invoked by HLL library routines, other LE/VSE services, and user-written HLL calls. In many cases, services will be invoked by HLL library routines, as a result of a user-specified function, such as a COBOL intrinsic function.

LE/VSE-conforming languages exhibit consistent behavior because language functions call LE/VSE services. For example, C `malloc()` and PL/I ALLOCATE each directly or indirectly call CEEGTST to obtain storage.

The sections below show examples of the syntax used to invoke LE/VSE callable services.

Invoking Callable Services from C

In C, invoke an LE/VSE callable service (with feedback code) using the syntax shown below:

```
#include &lt;leawi.h>
main ()
&lbrace.
 CEESERV(:pv.parm1, parm2, ... parmn, fc:epv.);
 &rbrace.
```

Figure 8. Sample Invocation of a Callable Service from C

`leawi.h` is a header file shipped withLE/VSE that contains declarations of LE/VSE callable services and OMIT_FC, which is used to explicitly omit the feedback code parameter, as shown below.

```
#include &lt;leawi.h>
main ()
&lbrace.
 CEESERV(:pv.parm1, parm2, ... parmn, OMIT&us.FC:epv.);
 &rbrace.
```

Figure 9. Omitting the Feedback Code when Calling a Service from C

Invoking Callable Services from COBOL

In COBOL, invoke an LE/VSE callable service using the syntax shown below:

```
COPY CEEIGZCT
&vellip.
CALL &odq.CEESERV&cdq. USING :pv.parm1 parm2 ... parmn fc:epv.
```

Figure 10. Sample Invocation of a Callable Service from COBOL

CEEIGZCT is a copy file file shipped with LE/VSE that contains COBOL declarations for symbolic LE/VSE feedback codes.

COBOL users may not omit the feedback code parameter.

Invoking Callable Services from PL/I

In PL/I, invoke an LE/VSE callable service (with feedback code) using the syntax shown below:

```
%INCLUDE CEEIBMCT;
%INCLUDE CEEIBMAW;
&vellip.
CALL CEESERV (:pv.parm1, parm2, ... parmn, fc:epv.);
```

Figure 11. Sample Invocation of a Callable Service from PL/I

CEEIBMAW and CEEIBMCT are include files shipped with LE/VSE. CEEIBMAW contains PL/I declarations of LE/VSE callable services. CEEIBMCT contains PL/I declarations of symbolic LE/VSE feedback codes.

PL/I allows you to omit arguments when invoking callable services. To do so, code an asterisk (*) in place of the argument, as shown below.

```
%INCLUDE CEEIBMAW
&vellip.
CALL CEESERV (:pv.parm1, parm2, ... parmn, *:epv.);
```

Figure 12. Omitting the Feedback Code when Calling a Service from PL/I

Invoking Callable Services from Assembler

In assembler, invoke an LE/VSE callable service (with feedback code) using the syntax shown below:

```
         LA    R1,PLIST
         L     R15,=V(CEESERV)
         BALR  R14,R15
         CLC   FC,CEE000       Check if feedback code is zero
         BNE   ER1             If not, branch to error routine
&vellip.
PLIST    DS    0D
         DC    A(:pv.PARM1:epv.)
&vellip.
*                              Parms 2 through :pv.n:epv.

         DC    A(FC+X'80000000')     Feedback code as last parm

:pv.PARM1:epv.    DC    F'5'            Parm 1
&vellip.
*                              Parms 2 through :pv.n:epv.

FC       DS    12C             Feedback code as last parm
CEE000   DC    12X'00'         Good feedback code
```

Figure 13. Sample Invocation of a Callable Service from Assembler

Assembler allows you to omit the *fc* parameter when invoking callable services. To do so, code X'80000000' in the *fc* parameter address slot, as shown in Figure 14 on page 19.

```
         LA    R1,PLIST
         L     R15,=V(CEESERV)
         BALR  R14,R15
&vellip.
PLIST    DS    0D
         DC    A(:pv.PARM1:epv.)
&vellip.
*                              Parms 2 through :pv.n:epv.
         DC    A(X'80000000')        Omitted feedback code in last slot

:pv.PARM1:epv.    DC    F'5'            Parm 1
&vellip.
*                              Parms 2 through :pv.n:epv.
```

Figure 14. Omitting the Feedback Code when Calling a Service from Assembler

LE/VSE Callable Services

Table 3 on page 19 lists LE/VSE callable services. Naming conventions of the callable services are as follows:

- Those services starting with CEE*x*, where *x* is not 5, are intended to be cross-system consistent; they operate on all platform-specific implementations ofLanguage Environment .
- Those services starting with CEE5 are services that exploit unique System/390 or VSE characteristics.

Table 3. LE/VSE Callable Services

Service Name	Description

Table 3. LE/VSE Callable Services

Condition Handling Services	

Table 3. LE/VSE Callable Services

CEE5CIB—Return Pointer to Condition Information Block	Given a condition token passed to a user-written condition handler, CEE5CIB returns a pointer to the condition information block associated with a condition. Allows access to detailed information about the subject condition during condition handling.

Table 3. LE/VSE Callable Services (continued)	
CEE5GRN—Get Name of Routine that Incurred Condition	Obtains the name of the routine that is currently running when a condition is raised. If there are nested conditions, the most recently signaled condition is used.
CEE5SPM—Query and Modify LE/VSE Hardware Condition Enablement	Allows the user to manipulate the program mask by enabling or masking hardware interrupts.
CEEDCOD—Decompose a Condition Token	Decomposes or changes an existing condition token.
CEEGPID—Retrieve the LE/VSE Version and Platform ID	Retrieves the LE/VSE version ID and platform ID currently in use.
CEEGQDT—Retrieve q_data_Token	Retrieves the q_data token from the ISI to be used by user condition handlers.
CEEHDLR—Register a User Condition Handler	Registers a user condition handler for the current stack frame. Currently, PL/I users can not call CEEHDLR.
CEEHDLU—Unregister a User Condition Handler	Unregisters a user condition handler for the current stack frame.
CEEITOK—Return Initial Condition Token	Returns the initial condition token for the current condition.
CEEMRCR—Move Resume Cursor Relative to Handle Cursor	Moves the resume cursor. You can either move the resume cursor to the call return point of the routine that registered the executing condition handler, or move the resume cursor to the caller of the routine that registered the executing condition handler.
CEENCOD—Construct a Condition Token	Dynamically constructs a condition token. The condition token communicates with message services, condition management,LE/VSE callable services, and user applications.
CEESGL—Signal a Condition	Signals a condition to the LE/VSE condition manager. It also may be used to provide qualifying data and create an instance specific information (ISI) field. The ISI contains information that is used by the LE/VSE condition manager to identify and react to conditions.

Table 3. LE/VSE Callable Services
Date and Time Services

Table 3. LE/VSE Callable Services	
CEECBLDY—Convert Date to COBOL Integer Format	Converts a string representing a date into a COBOL Integer format that is compatible with ANSI COBOL intrinsic functions.
CEEDATE—Convert Lilian Date to Character Format	Converts a number representing a Lilian date to a date written in character format. The output is a character string such as "1996/05/14".
CEEDATM—Convert Seconds to Character Timestamp	Converts a number representing the number of seconds since 00:00:00 14 October 1582 to a character format. The format of the output is a character string, such as "1996/05/14 20:37:00".
CEEDAYS—Convert Date to Lilian Format	Converts a string representing a date into a Lilian format. The Lilian format represents a date as the number of days since 14 October 1582, the beginning of the Gregorian calendar.
CEEDYWK—Calculate Day of Week from Lilian Date	Calculates the day of the week on which a Lilian date falls. The day of the week is returned to the calling routine as a number between 1 and 7.

Table 3. LE/VSE Callable Services (continued)	
CEEGMT—Get Current Greenwich Mean Time	Returns the current Greenwich Mean Time (GMT) as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582. These values are compatible with those generated and used by the other LE/VSE date and time services.
CEEGMTO—Get Offset from Greenwich Mean Time to Local Time	Returns values to the calling routine which represent the difference between the local system time and Greenwich Mean Time.
CEEISEC—Convert Integers to Seconds	Converts separate binary integers representing year, month, day, hour, minute, second, and millisecond to a number representing the number of seconds since 00:00:00 14 October 1582. Use CEEISEC instead of CEESECS when the input is in numeric format rather than character format.
CEELOCT—Get Current Local Time	Returns the current local time in three formats: • Lilian date (the number of days since 14 October 1582) • Lilian timestamp (the number of seconds since 00:00:00 14 October 1582) • Gregorian character string (in the form YYYYMMDDHHMISS999)
CEEQCEN—Query the Century Window	Queries the century within which LE/VSE assumes 2-digit year values lie. Use it in conjunction with CEESCEN when it is necessary to save and restore the current setting.
CEESCEN—Set the Century Window	Sets the century whereLE/VSE assumes 2-digit year values lie. Use it in conjunction with CEEDAYS or CEESECS when you process date values that contain 2-digit years (for example, in the YYMMDD format), or when the LE/VSE default century interval doesn't meet the requirements of a particular application.
CEESECI—Convert Seconds to Integers	Converts a number representing the number of seconds since 00:00:00 14 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond. Use CEESECI instead of CEEDATM when the output is needed in numeric format rather than character format.
CEESECS—Convert Timestamp to Number of Seconds	Converts a string representing a timestamp into a number representing the number of seconds since 00:00:00 14 October 1582. This service makes it easier to do time arithmetic, such as calculating the elapsed time between two timestamps.
CEEUTC—Get Coordinated Universal Time	CEEUTC is an alias of CEEGMT.

Table 3. LE/VSE Callable Services
Dynamic Storage Services

Table 3. LE/VSE Callable Services	
CEECRHP—Create New Additional Heap	Defines additional heaps. The heaps defined by CEECRHP can be used just like the LE/VSE initial heap (heap id of 0). However, the entire heap created by CEECRHP may be quickly freed with a single call to the CEEDSHP (discard heap) service.
CEECZST—Reallocate (Change Size of) Storage	Changes the size of a previously allocated storage element while preserving its contents. Reallocation of a storage element is accomplished by allocating a new storage element of a new size and copying the contents of the old element to the new element.

Table 3. LE/VSE Callable Services (continued)	
CEEDSHP—Discard Heap	Discards an entire heap previously created with a call to CEECRHP.
CEEFRST—Free Heap Storage	Frees storage previously allocated by CEEGTST. It can be used to free both large and small blocks of storage efficiently because freed storage is retained on a free chain instead of being returned to the operating system.
CEEGTST—Get Heap Storage	Allocates storage from a heap whose ID you specify. It can be used to efficiently acquire both large and small blocks of storage.

Table 3. LE/VSE Callable Services
General Services

Table 3. LE/VSE Callable Services	
CEE5DMP—Generate Dump	Generates a dump of the run-time environment of LE/VSE and of the member language libraries. The dump can be modified to selectively include such information as number and contents of enclaves and threads, traceback of all routines on a call chain, file attributes, and variable, register, and storage contents.
CEE5PRM—Query Parameter String	Returns to the calling routine the parameter string that was specified at invocation of the program. The returned parameter string contains only user parameters. If no user parameters are available, a blank string is returned.
CEE5RPH—Set Report Heading	Sets the heading displayed at the top of the storage or run-time options report.LE/VSE generates the storage report when the RPTSTG(ON) run-time option is specified, and the options report when the RPTOPTS(ON) run-time option is specified.
CEE5USR—Set or Query User Area Fields	Sets or queries one of two 4-byte fields in the enclave data block known as the user area fields. The user area fields are associated with an enclave and are maintained on an enclave basis. A user area might be used by vendor or applications to store a pointer to a global data area or keep a recursion counter.
CEERAN0—Calculate Uniform Random Numbers	Generates a sequence of uniform pseudo-random numbers between 0 and 1 using the multiplicative congruential method with a user-specified seed.
CEETDLI—Invoke DL/I	Provides an interface to DL/I DOS/VS.
CEETEST—Invoke Debug Tool	Invokes a debug tool, such as Debug Tool for VSE/ESA.

Table 3. LE/VSE Callable Services
Initialization/Termination Services

Table 3. LE/VSE Callable Services	
CEE5ABD—Terminate Enclave with an Abend	Requests LE/VSE to terminate the enclave via an abend. The abend can be issued either with or without cleanup.
CEE5GRC—Get the Enclave Return Code	Retrieves the current value of the user enclave return code.
CEE5SRC—Set the Enclave Return Code	Modifies the user enclave return code. The value set will be used in the calculation of the final enclave return code at enclave termination.

Table 3. LE/VSE Callable Services	
Locale Services	

Table 3. LE/VSE Callable Services	
CEEFMON—Format Monetary String	Converts numeric values to monetary strings.
CEEFTDS—Format Time and Date into Character String	Converts time and date specifications into a character string.
CEELCNV—Query Locale Numeric Conventions	Returns information about the LC_NUMERIC and LC_MONETARY categories of the locale.
CEEQDTC—Query Locale Date and Time Conventions	Queries the locale's date and time conventions.
CEEQRYL—Query Active Locale Environment	Allows the calling routine to query the current locale.
CEESCOL—Compare Collation Weight of Two Strings	Compares two character strings based on the collating sequence specified in the LC_COLLATE category of the locale.
CEESETL—Set Locale Operating Environment	Allows an enclave to establish a locale operating environment, which determines the behavior of character collation, character classification, date and time formatting, numeric punctuation, and message responses.
CEESTXF—Transform String Characters into Collation Weights	Transforms each character in a character string into its collation weight and returns the length of the transformed string.

Table 3. LE/VSE Callable Services
Mathematical Services **LE/VSE math services are scalar routines.*x* is a data type variable.**

Table 3. LE/VSE Callable Services	
CEESxABS	Absolute value
CEESxACS	Arccosine
CEESxASN	Arcsine
CEESxATH	Hyperbolic arctangent
CEESxATN	Arctangent
CEESxAT2	Arctangent x/y
CEESxCJG	Conjugate of complex
CEESxCOS	Cosine
CEESxCSH	Hyperbolic cosine
CEESxCTN	Cotangent
CEESxDIM	Positive difference
CEESxDVD	Floating complex divide
CEESxERF	Error function
CEESxEXP	Exponential (base e)

Table 3. LE/VSE Callable Services (continued)	
CEESxGMA	Gamma function
CEESxIMG	Imaginary part of complex
CEESxINT	Truncation
CEESxLGM	Log gamma function
CEESxLG1	Logarithm base 10
CEESxLG2	Logarithm base 2
CEESxLOG	Logarithm base e
CEESxMLT	Floating complex multiply
CEESxMOD	Modular arithmetic
CEESxNIN	Nearest integer
CEESxNWN	Nearest whole number
CEESxSGN	Transfer of sign
CEESxSIN	Sine
CEESxSNH	Hyperbolic sine
CEESxSQT	Square root
CEESxTAN	Tangent
CEESxTNH	Hyperbolic tangent
CEESxXPx	Exponentiation

Table 3. LE/VSE Callable Services
Message Handling Services

Table 3. LE/VSE Callable Services	
CEECMI—Store and Load Message Insert Data	Stores the message insert data and loads the address of that data into the instance specific information (ISI) field associated with the condition being processed, after optionally creating an ISI.
CEEMGET—Get a Message	Retrieves, formats, and stores a message in a buffer for manipulation or output by the caller.
CEEMOUT—Dispatch a Message	Dispatches a message to a destination which you specify.
CEEMSG—Get, Format, and Dispatch a Message	Obtains/formats/dispatches a message corresponding to an input condition token received from a callable service. You can use this service to print a message after a call to anyLE/VSE service that returns a condition token.

Table 3. LE/VSE Callable Services
National Language Support Services

Table 3. LE/VSE Callable Services	
CEE5CTY—Set Default Country	Allows the calling routine to change or query the current national country setting. The country setting affects the date format, the time format, the currency symbol, the decimal separator character, and the thousands separator.
CEE5LNG—Set National Language	Allows the calling routine to change or query the current national language. The national languages may be recorded on a LIFO national language stack. Changing the national language changes the languages of error messages, the names of the days of the week, and the names of the months.
CEE5MCS—Obtain Default Currency Symbol	Returns the default currency symbol for the specified country.
CEE5MDS—Obtain Default Decimal Separator	Returns the default decimal separator for the specified country.
CEE5MTS—Obtain Default Thousands Separator	Returns the default thousands separator for the specified country.
CEEFMDA—Obtain Default Date Format	Returns the default date picture string for the specified country.
CEEFMDT—Obtain Default Date and Time Format	Returns the default date and time picture strings for the specified country.
CEEFMTM—Obtain Default Time Format	Returns the default time picture string for the specified country.

Chapter 4. LE/VSE Run-Time Options

This chapter lists the run-time options available with LE/VSE. Although most LE/VSE options apply to all LE/VSE-conforming languages, some are specific only to a single language.LE/VSE helps migration by mapping run-time options to C, COBOL, and PL/I options. Specific mapping information is found in LE/VSE Programming Reference.

Table 4. LE/VSE Run-Time Options

Run-Time Option	Description
ABPERC	Exempts a specified VSE cancel code, program-interruption code, or user abend code from LE/VSELE/VSE condition handling.
ABTERMENC	Determines how an enclave ending with an unhandled condition of severity 2 or greater terminates: with a return code and reason code, or with an abend.
AIXBLD	Dynamic invocation of access method services for COBOL programs.
ALL31	Indicates the entire application is running AMODE 31.
ANYHEAP	Controls allocation of LE/VSE and HLL heap storage not restricted to below the 16MB line.
ARGPARSE	Specifies whether arguments on the command line are to be parsed in the usual C format.
BELOWHEAP	Controls allocation of LE/VSE and HLL heap storage below the 16MB line.
CBLOPTS	Indicates the order of run-time options. This option is honored only when the main routine is written in COBOL.
CBLPSHPOP	Controls whether CICS PUSH HANDLE and CICS POP HANDLE are issued when a COBOL subroutine is called.
CHECK	Checking of index, subscript, reference modification, and variable length group ranges in COBOL programs.
COUNTRY	Specifies the default formats for date, time, currency symbol, decimal separator, and the thousands separator based upon a country.
DEBUG	Activates the COBOL batch debugging features.
DEPTHCONDLMT	Limits the extent to which conditions can be nested.
ENV	Specifies the operating system environment for a C application.
ENVAR	Sets the initial values for the environment variables specified. With ENVAR, you can pass into an application switches or tagged information that can then be accessed during application execution using the C functions `getenv()`, `setenv()`, and `clearenv()`.
ERRCOUNT	Specifies how many non-fatal errors are allowed before the program is abnormally terminated.

Table 4. LE/VSE Run-Time Options (continued)

Run-Time Option	Description
EXECOPS	Specifies whether run-time options can be specified on the command line.
HEAP	Controls the allocation of the user heap.
LIBSTACK	Controls the allocation of the enclave's library stack storage.
MSGFILE	Specifies the *filename* of the run-time diagnostics file.
MSGQ	Specifies the maximum number of message inserts allocated on a per thread basis during execution.
NATLANG	Specifies the national language to be used for the run-time environment.
PLIST	Specifies the format of the invocation parameters received by your C application when it is invoked.
REDIR	Specifies whether redirections for `stdin`, `stderr`, and `stdout` are allowed from the command line.
RPTOPTS	Specifies that a report of the run-time options in use by the program will be generated.
RPTSTG	Specifies that a report of the storage used by the program be generated at the end of execution.
RTEREUS	Specifies a reusable run-time environment for COBOL programs.
STACK	Controls the allocation of the enclave's stack storage.
STORAGE	Used for debugging. Specifies initial values to which all heap and stack storage is set when first allocated or freed.
TERMTHDACT	Sets the level of information produced due to an error of severity 2 or greater.
TEST	Specifies the conditions under which a debug tool (such asDebug Tool for VSE/ESA) assumes control of the application.
TRACE	Determines whether LE/VSE run-time library tracing is active.
TRAP	Specifies how LE/VSE routines should handle error conditions and program interrupts.
UPSI	Sets UPSI switches (affects only COBOL programs).
XUFLOW	Specifies whether an exponent underflow causes a program interrupt.

Chapter 5. Sample Routines

This chapter includes sample routines that demonstrate several aspects of LE/VSE.

- Assembler routine, Figure 15 on page 29
- C routine, Figure 16 on page 30
- COBOL routine, Figure 17 on page 32
- PL/I routine, Figure 18 on page 34

Sample Assembler Routine

```
*COMPILATION UNIT: LEASMMN
* ===================================================================
*
*        A simple main assembler routine that brings up the
*        LE/VSE environment, prints a message in the main routine,
*        and returns with a return code of 0, modifier of 0.
*
* ===================================================================
MAIN     CEEENTRY PPA=MAINPPA
*
*        Invoke CEEMOUT to issue a message for us
*
         CALL  CEEMOUT,(STRING,DEST,0)      Omitted feedback code
*
*   Terminate the LE/VSE environment and return to the caller
*
         CEETERM  RC=0,MODIFIER=0
* ===================================================================
*              CONSTANTS AND WORKAREAS
* ===================================================================
*
DEST     DC    F'2'
STRING   DC    Y(STRLEN)
STRBEGIN DC    C'In the main routine'
STRLEN   EQU   *-STRBEGIN
MAINPPA  CEEPPA  ,               Constants describing the code block
         CEEDSA  ,               Mapping of the dynamic save area
         CEECAA  ,               Mapping of the common anchor area
         END   MAIN              Nominate MAIN as the entry point
```

Figure 15. A Simple Main Assembler Routine

Sample C Routine

This routine demonstrates the followingLE/VSE callable services:

- CEEMOUT—Dispatch a message
- CEELOCT—Get current time
- CEEDATE—Convert Lilian date to character format

```
#include <leawi.h>
#include <string.h>
main ()
{
  _FEEDBACK   fbcode;            /* fbcode for all callable services */


/*********************************************************************/
/* Parameters passed to CEEMOUT.  Typedefs found in leawi.h.         */
/*********************************************************************/
  _VSTRING    msg;

  _INT4       destination;
/*********************************************************************/
/* Parameters passed to CEELOCT.  Typedefs found in leawi.h.         */
/*********************************************************************/
  _INT4       lildate;

  _FLOAT8     lilsecs;
  _CHAR17     greg;
/*********************************************************************/
/* Parameters passed to CEEDATE.  Typedefs found in leawi.h.         */
/*********************************************************************/

  _CHAR80  str_date;
  _VSTRING pattern;
/*********************************************************************/
/* Starting and ending messages                                      */
/*********************************************************************/

  _CHAR80  startmsg  = "Callable service example starting (C).";
  _CHAR80  endingmsg = "Callable service example ending (C).";

/*********************************************************************/
/* Start execution. Print the first message.                         */

/*********************************************************************/
  destination = 2;
  strcpy( msg.string, startmsg );
  msg.length  = strlen( msg.string );
  CEEMOUT ( &msg, &destination, &fbcode );


/*********************************************************************/
/* Get the local date and time, format it, and print it out.         */
/*********************************************************************/
  CEELOCT ( &lildate, &lilsecs, greg, &fbcode );

  strcpy  ( pattern.string,\
          "Today is Wwwwwwwwwwwwz, Mmmmmmmmmz ZD, YYYY." );
  pattern.length = strlen( pattern.string );
  memset ( msg.string , ' ' , 80 );
  CEEDATE ( &lildate, &pattern, msg.string, &fbcode );

  msg.length = 80;
  CEEMOUT ( &msg, &destination, &fbcode );
/*********************************************************************/
/* Say goodbye.                                                      */
/*********************************************************************/

  strcpy  ( msg.string, endingmsg );
  msg.length = strlen( msg.string );
  CEEMOUT ( &msg, &destination, &fbcode );
}
```

Figure 16. Sample C Routine

Sample COBOL Routine

This routine demonstrates the followingLE/VSELE/VSE callable services:

- CEEMOUT—Dispatch a message
- CEELOCT—Get current time
- CEEDATE—Convert Lilian date to character format

```
CBL C,RENT,APOST,OPTIMIZE,LIST,DATA(31),NODYNAM,LIB
      ****************************************************************
      *  This routine demonstrates the following LE/VSE callable     *
      *  services : CEEMOUT, CEELOCT, CEEDATE                        *
      ****************************************************************

      ****************************************************************
      **          I D          D I V I S I O N                    ***
      ****************************************************************
       Identification Division.
       Program-Id.    AWIXMP.

      ****************************************************************
      **          D A T A      D I V I S I O N                    ***
      ****************************************************************
       Data Division.
       Working-Storage Section.

      ****************************************************************
      **  Declarations for the local date/time service.
      ****************************************************************
       01   Feedback.
       COPY CEEIGZCT.
        02   Fb-severity      PIC 9(4) Binary.
        02   Fb-detail        PIC X(10).
       77   Dest-output       PIC S9(9) Binary.
       77   Lildate           PIC S9(9) Binary.
       77   Lilsecs           COMP-2.
       77   Greg              PIC X(17).

      ****************************************************************
      **  Declarations for messages and pattern for date formatting.
      ****************************************************************
       01   Pattern.
        02                    PIC 9(4) Binary Value 45.
        02                    PIC X(45) Value
            'Today is Wwwwwwwwwwwwz, Mmmmmmmmmz ZD, YYYY.'.

       77   Start-Msg         PIC X(80) Value
            'Callable Service example starting.'.

       77   Ending-Msg        PIC X(80) Value
            'Callable Service example ending.'.

       01 Msg.
         02 Stringlen         PIC S9(4) Binary.
         02 Str               .
          03                  PIC X Occurs 1 to 80 times
                                 Depending on Stringlen.

      ****************************************************************
      **          P R O C      D I V I S I O N                    ***
      ****************************************************************
       Procedure Division.
       000-Main-Logic.
           Perform 100-Say-Hello.
           Perform 200-Get-Date.
           Perform 300-Say-Goodbye.
           Stop Run.

      **
      ** Setup initial values and say we are starting.
      **
       100-Say-Hello.
           Move 80 to Stringlen.
           Move 02 to Dest-output.
           Move Start-Msg to Str.
           CALL 'CEEMOUT' Using Msg     Dest-output Feedback.
           Move Spaces to Str.
           CALL 'CEEMOUT' Using Msg     Dest-output Feedback.

      **
      ** Get the local date and time and display it.
      **
       200-Get-Date.
           CALL 'CEELOCT' Using Lildate Lilsecs     Greg      Feedback.
           CALL 'CEEDATE' Using Lildate Pattern     Str       Feedback.
           CALL 'CEEMOUT' Using Msg     Dest-output Feedback.
           Move Spaces to Str.
           CALL 'CEEMOUT' Using Msg     Dest-output Feedback.

      **
      ** Say Goodbye.
      **
       300-Say-Goodbye.
           Move Ending-Msg to Str.
           CALL 'CEEMOUT' Using Msg     Dest-output Feedback.
       End Program AWIXMP.
```

Figure 17. Sample COBOL Routine

Sample PL/IPL/I Routine

This sample demonstrates the followingLE/VSE callable services:

- CEEMOUT—Dispatch a message
- CEELOCT—Get current time
- CEEDATE—Convert Lilian date to character format

```
*PROCESS MACRO;
 /*compilation unit: cecgxmp                                    */
 /*************************************************************/
 /* This routine demonstrates the following LE/VSE callable **/
 /* services:  CEEMOUT, CEELOCT, and CEEDATE.                 **/
 /*                                                           **/
 /*************************************************************/
 cecgxmp: proc options(main);

 /* Declarations for callable services */
    %INCLUDE  CEEIBMAW;
    %INCLUDE  CEEIBMCT;

 /* feedback code for all callable services*/
    dcl 01 fc FEEDBACK;
```

```
 /*************************************************************/
 /** Parameters passed to CEEMOUT.                           **/
 /**                                                         **/
 /*************************************************************/

    dcl startmsg VSTRING
          init('Callable service example starting (PL/I)');
    dcl endmsg VSTRING
          init('Callable service example ending (PL/I)');
    dcl strmsg VSTRING;
    dcl destination real fixed binary ( 31,0 );
```

```
 /*************************************************************/
 /** Parameters passed to CEELOCT.                           **/
 /**                                                         **/
 /*************************************************************/
    dcl lildate real fixed binary ( 31,0 );
    dcl lilsecs real float decimal ( 16 );
    dcl greg character ( 17 );
```

```
 /*************************************************************/
 /** Parameters for CEEDATE.                                 **/
 /**                                                         **/
 /*************************************************************/
    dcl pattern VSTRING;
    dcl chrdate CHAR80 init ((80)' ');
```

```
 /*************************************************************/
 /** Start execution.  Print the first message.              **/
 /**                                                         **/
 /*************************************************************/
    destination = 2;
    call CEEMOUT ( startmsg , destination , fc );
    IF  ¬ FBCHECK( fc, CEE000)  THEN  DO;
       DISPLAY( 'CEEMOUT failed with msg ' || fc.MsgNo );
       STOP;
       END;
```

```
 /*************************************************************/
 /** Get the local date and time. Format it, and print it    **/
 /** out.                                                    **/
 /*************************************************************/
    call CEELOCT ( lildate , lilsecs , greg , fc );
    IF  ¬ FBCHECK( fc, CEE000)  THEN  DO;
       DISPLAY( 'CEELOCT failed with msg ' || fc.MsgNo );
       STOP;
       END;
```

```
    pattern = 'Today is Wwwwwwwwwwwz, Mmmmmmmmmz, ZD, YYYY.';
    call CEEDATE ( lildate , pattern , chrdate , fc );
    IF  ¬ FBCHECK( fc, CEE000)  THEN  DO;
       DISPLAY( 'CEEDATE failed with msg ' || fc.MsgNo );
       STOP;
       END;
```

```
    strmsg = chrdate;
    call CEEMOUT ( strmsg , destination , fc );
    IF  ¬ FBCHECK( fc, CEE000)  THEN  DO;
       DISPLAY( 'CEEMOUT failed with msg ' || fc.MsgNo );
       STOP;
       END;
```

```
 /*************************************************************/
 /** Say good bye.                                           **/
 /**                                                         **/
 /*************************************************************/
    call CEEMOUT ( endmsg , destination , fc );
    IF  ¬ FBCHECK( fc, CEE000)  THEN  DO;
       DISPLAY( 'CEEMOUT failed with msg ' || fc.MsgNo );
       STOP;
       END;

 end;
```

Figure 18. Sample PL/I Routine

Chapter 6. Product Requirements

Machine Requirements

LE/VSE-conforming compiler-generated object code runs on any hardware configuration supported by the licensed programs specified below.LE/VSE supports the DBCS character sets on the IBM Personal System/55 (as 3270) and IBM 5550 Family (as 3270).

Programming Requirements

LE/VSE runs under the control of, or in conjunction with, the following IBM licensed programs and their subsequent releases unless otherwise announced by IBM.

Required Licensed Programs

The licensed programs listed in Table 5 on page 35 are required to install and customize LE/VSE, or to run LE/VSE applications.

Table 5. Required Licensed Programs for LE/VSE

Required Licensed Program	Minimum Release	Program Number
One of:		
VSE/ESA	Version 1 Release 4	5750-ACD
VSE/ESA	Version 2 Release 1	5690-VSE
High Level Assembler/MVS & VM & VSE	Release 1	5696-234

Optional Licensed Programs

The licensed compiler programs listed in Table 6 on page 35, with or without the Debug Tool feature, can optionally be used to generate LE/VSE applications.

Table 6. Optional Licensed Compiler Programs for LE/VSE

Optional Licensed Program	Minimum Release	Program Number
C/VSE	Release 1	5686-A01
COBOL/VSE	Release 1	5686-068
PL/I VSE	Release 1	5686-069

The licensed programs listed in Table 7 on page 35 can optionally be used with LE/VSE.

Table 7. Optional Licensed Programs for LE/VSE

Optional Licensed Program	Minimum Release	Program Number
BookManager Read	Release 2 is required to view softcopy documentation	73F6-023 (Read/2)

Table 7. Optional Licensed Programs for LE/VSE (continued)

Optional Licensed Program	Minimum Release	Program Number
CICS/VSE[“1” on page 36]	Version 2 Release 3 with PTF UN89454	5686-026
CSP/AD	Version 3 Release 3	5668-813
CSP/AE	Version 3 Release 3	5668-814
DFSORT/VSE	Version 3 Release 1	5746-SM3
DL/I DOS/VS	Release 10 with PTF UN73450	5746-XX1
DOS/VS Sort/Merge	Version 2 Release 5	5746-SM2
QMF/VSE	Version 3 Release 2	5648-061
REXX/VSE	Release 1	5686-058
SQL/DS	Version 3 Release 4 with PTF UN76254	5688-103

Note:

1. LE/VSE is not supported in VSE/ICCF interactive partitions.

Compatibility Considerations

LE/VSE's open architecture ensures that there will be minimal disruption of your resources as you migrate to LE/VSE; it provides COBOL and PL/I source code compatibility, and, with certain exceptions, COBOL object code and executable phase compatibility with your existing applications. Many of your existing VS COBOL IIVS COBOL II applications can run withLE/VSELE/VSE without relink-editing or recompiling. Routines compiled withLE/VSE -conforming compilers can be mixed with old VS COBOL II routines in an application, so applications can be enhanced and maintained selectively. However, routines that depend on dump formats, condition handling routines, or assembler routines may have to be changed.

Compatibility and migration are fully documented in LE/VSE Run-Time Migration Guide, *IBM COBOL for VSE/ESA Migration Guide* , and *IBM PL/I for VSE/ESA Migration Guide*

Bibliography

Language Environment Publications

IBM Language Environment for VSE/ESA

Fact Sheet GC33-6679
Concepts Guide GC33-6680
Debugging Guide and Run-Time Messages SC33-6681
Installation and Customization Guide SC33-6682
*Licensed Program Specifications,*GC33-6683
Programming Guide SC33-6684
Programming Reference SC33-6685
Run-Time Migration Guide SC33-6687
Writing Interlanguage Communication Applications SC33-6686
C Run-Time Programming Guide SC33-6688
C Run-Time Library Reference SC33-6689

LE/VSE-Conforming Language Product Publications

IBM C for VSE/ESA

*Licensed Program Specifications,*GC09-2421GC09-2421
*Installation and Customization Guide,*GC09-2422
*Migration Guide,*SC09-2423
*User's Guide,*SC09-2424
*Language Reference,*SC09-2425
*Diagnosis Guide,*GC09-2426

IBM COBOL for VSE/ESA

*General Information,*GC26-8068
*Licensed Program Specifications,*GC26-8069
*Migration Guide,*GC26-8070
*Installation and Customization Guide,*SC26-8071
*Programming Guide,*SC26-8072
*Language Reference,*SC26-8073
*Diagnosis Guide,*SC26-8528
*Reference Summary,*SX26-3834

IBM PL/I for VSE/ESAIBM PL/I for VSE/ESA

*Fact Sheet,*GC26-8052
*Programming GuideProgramming Guide,*SC26-8053
*Language Reference,*SC26-8054
*Licensed Program Specifications,*GC26-8055GC26-8055
*Migration Guide,*SC26-8056
*Installation and Customization Guide,*SC26-8057SC26-8057
*Diagnosis Guide,*SC26-8058
*Compile-Time Messages and Codes,*SC26-8059
*Reference Summary,*SX26-3836

Debug Tool for VSE/ESA

User's Guide and Reference, SC26-8797
Installation and Customization Guide, SC26-8798
Fact Sheet, GC26-8925

Softcopy Publications

The following collection kit contains the LE/VSE and LE/VSE-conforming language product publications:

VSE Collection, SK2T-0060

You can order these publications from Mechanicsburg through your IBM representative.

Language EnvironmentLanguage Environment Glossary

abend
Abnormal end of application.

active routine
The currently executing routine.

additional heap
An LE/VSELE/VSE 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/VSELE/VSE heap controlled by the ANYHEAP run-time option. It contains library data, such as LE/VSELE/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*, and *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/VSELE/VSE heap controlled by the BELOWHEAP run-time option, which contains library data, such as LE/VSELE/VSE control block and data structures not normally accessible from user code. Below heap always resides below 16MB. See also *anywhere heap*, *initial heap*, and *additional heap*.

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 an LE/VSELE/VSE-conforming high level language using the conventional LE/VSELE/VSE-defined call interface, and usable by all programs sharing the LE/VSELE/VSE 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 an LE/VSELE/VSE *enclave*.

command line
The command used to invoke an application program, and the associated program arguments and LE/VSELE/VSE run-time options. This can be the job control EXEC statement and the associated PARM parameter, or the parameter string passed to the CC `system()` function.

compilation unit
An independently compilable sequence of HLL statements. Each HLL product has different rules for what makes up a compilation unit. Synonym for *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 LE/VSELE/VSE 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 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/IPL/I ON-unit) invoked by theLE/VSELE/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*.

condition token
In LE/VSELE/VSE, 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*.

dynamic storage area (DSA)
An area of storage obtained during the running of an application that consists of a register save area and an area for automatic data, such as program variables. DSAs are generally allocated within LE/VSELE/VSE-managed stack segments. DSAs are added to the stack when a routine is entered and removed upon exit in a last in, first out (LIFO) manner. In LE/VSELE/VSE, a DSA is known as a *stack frame*.

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

ESDS
Entry sequenced data sets. See *VSAM*.

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*.

extended binary-coded decimal interchange code (EBCDIC)
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.

handle cursor
Points to the first condition handler within the stack frame that is to be invoked when a condition occurs. As condition handling progresses, the handle cursor moves to earlier handlers within the stack frame, or to the first handler in the calling stack frame.

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 also *additional heap*, *anywhere heap*, *below heap*, *heap element*, and *initial 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 LE/VSELE/VSE storage management scheme subdivides heap segments into individual heap elements. If the initial heap segment becomes full, LE/VSELE/VSE 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 LE/VSELE/VSE 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/VSELE/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 EnvironmentLanguage Environment-conforming compilers.

Language Environment for VSE/ESA
An IBM software product that is the implementation of Language Environment on the VSE platform.

LE/VSELE/VSE
Short form of Language Environment for VSE/ESA.

LE/VSELE/VSE-conforming
Adhering to LE/VSELE/VSE's common interface.

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 local data in C and WORKING-STORAGE in COBOL.

locale
The definition of the subset of a user's environment that depends on language and cultural conventions.

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.

non-LE/VSELE/VSE conforming
Any HLL program that does not adhere to LE/VSELE/VSE's common interface. For example, VS COBOL II, DOS/VS COBOL, and DOS/VS PL/I are all non-LE/VSELE/VSE conforming HLLs. Synonym for *pre-LE/VSELE/VSE conforming*.

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
Synonym for *object module*.

object module
A portion of an object program suitable as input to a linkage editor. Synonym for *object deck*.

online
Pertaining to a user's ability to interact with a computer.

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.

pre-LE/VSELE/VSE conforming
Any HLL program that does not adhere to LE/VSE's common interface. For example, VS COBOL II, DOS/VS COBOL, and DOS/VS PL/I are all pre-LE/VSELE/VSE conforming HLLs. Synonym for *non-LE/VSELE/VSE conforming*.

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

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.

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 LE/VSELE/VSE, 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 LE/VSELE/VSE, refers to a procedure, function, or subroutine.

RRDS
Relative record data sets. See *VSAM*.

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

run time
Any instant at which a program is being executed. Synonym for *execution time*.

run-time environment
A set of resources that are used to support the execution of a program. Synonym for *execution environment*.

run unit
One or more object programs that are executed together. In LE/VSELE/VSE, 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.

scope
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/VSELE/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 LE/VSELE/VSE 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 LE/VSELE/VSE 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 thecommon run-time environmentcommon 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*.

Index

A

assembler language
  application example 18

B

bibliography 37

C

C
  application example 29
  sample callable service syntax 17
callable services
  invoking 17
  table listing 19
COBOL
  application example 31
  sample callable service syntax 18
common run-time environment, introduction 1
compatibility 36
condition 12
condition handler 12
condition handling
  callable services for 19
  model
    description 12
    introduction 11
    responses 15
    terminology 12
  simplified error recovery 3
condition token
  definition 12
  how created and used 14
  how represented 14
conforming languages, LE/VSELE/VSE xv
cursor
  handle 12
  resume 12, 15

D

Debug Tool for VSE/ESA 4
debugging, simplified with common dump 4
dump, common 4, 16

E

enclave 8
environment, common run-time 1
exception handling 11

F

feedback code
feedback code *(continued)*
  definition 12
  in callable services 14, 17, 18
file sharing 7

H

handle cursor 12
hardware requirements 35
heap
  element 10
  increment 10
  segment 10
  storage 10
HLL condition handler 13

I

increment
  heap 10
interlanguage communication (ILC) 3
interrupts 13

J

Japanese language support 16

L

language support
  callable services for 24
  description of 16
LE/VSE
  common run-time environment 1
  condition handling model 16
  conforming languages xv
  introduction xv
  message handling model 16
  overview 1
  program management model 5
  storage handling model 10
LE/VSELE/VSE
  architectural models 5
  condition handling model 11
locale callable services 4, 22

M

math services 23
message handling
  callable services for 24
  model 16
models, architectural
  condition handling 11, 16
  message handling 16
  program management 5

models, architectural *(continued)*
storage management 10, 11

N

national language support (NLS)
callable services for 24
description of 16

O

options, run-time 27

P

parallel processing 8
percolate action 16
PL/I
sample callable service syntax 18
PL/IPL/I
application example 33
process 7
program and tasking model 5
program management model
enclave 8
entities 6
process 7
terminology 5
thread 8
promote action 16

R

report
storage 11
resume
action 15
cursor 12, 15
run-time environment, introduction 1
run-time options 27

S

scope
of language semantics 8
software requirements 35
stack
frame 12
storage 10
stack, storage 10
static storage, in enclave 8
storage
callable services for 21
in thread 8
management model 10
report 11
static, in enclave 8
storage handling model, overview 10
storage handling model, terminology 10
suballocations, of storage 12
syntax
calling 17

T

terminology
condition handling model 12
glossary 39
program management model 5
storage management model 10
thread 8
token, condition 14

U

user-written condition handler 13

V

VS COBOL IIVS COBOL II, using with LE/VSELE/VSE 3

C33-6680-00