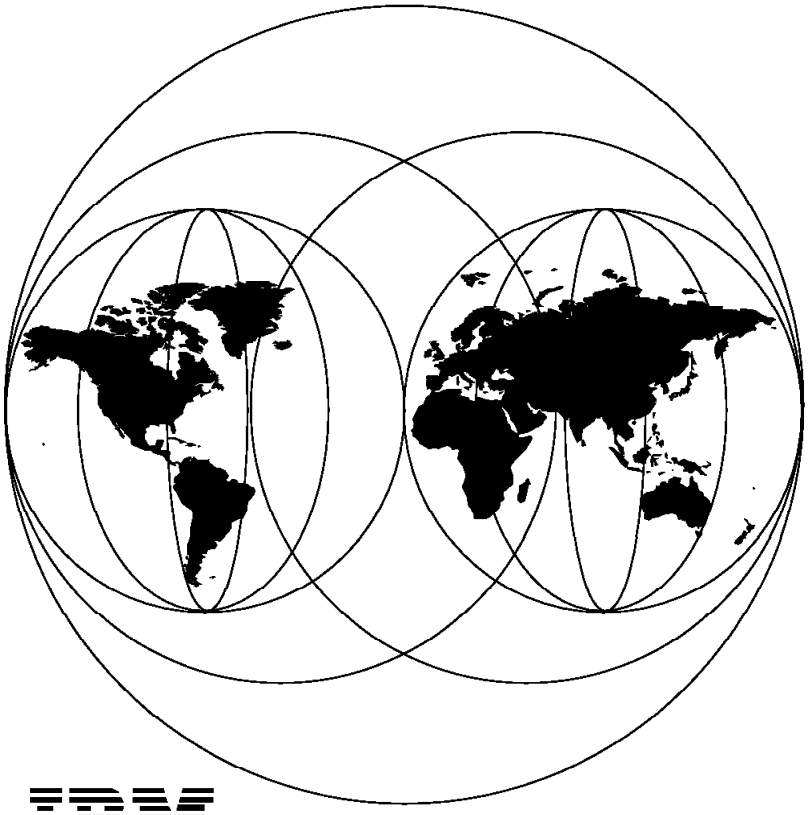
SG24-4798-00

Taking Advantage of IBM Language Environment for VSE/ESA

October 1996



IBM

International Technical Support Organization
Boeblingen Center

International Technical Support Organization

SG24-4798-00

Taking Advantage of IBM Language Environment for VSE/ESA

October 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special Notices” on page 71.

First Edition (October 1996)

This edition applies to Language Environment for VSE for use with the VSE/ESA Operating System, program number 5690-VSE.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. 3222 Building 71032-02
Postfach 1380
71032 Böblingen, 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

Tables

Preface

The IBM Language Environment for VSE/ESA (LE/VSE) now also offers for VSE customers the concept of a common run-time library for programs written in high level languages. All applications written in C, PL/I and COBOL can use a generalized set of services during program execution, thus avoiding cross-language inconsistencies.

Together with LE/VSE, IBM provides a set of new compilers, COBOL/VSE, PL/I VSE and C/VSE. Apart from of enabling the usage of the newest advanced system functions, they also offer the exploitation of the ″Year 2000″ date format which is a prerequisite for the migration of date-dependent programs; the most appropriate techniques for this simple migration are described and illustrated by examples.

This redbook has been written for all professionals concerned with the upkeep of their application program libraries, especially in anticipation of the changes required for Year 2000. Good knowledge of VSE/ESA concepts and HLL programming is assumed.

How This Redbook Is Organized

The redbook is organized as follows:

- Chapter 1, “Introduction to LE/VSE and Benefits of Using LE/VSE”

  This chapter provides a general overview of the concepts of LE/VSE, and describes the benefits for a customer using LE/VSE and the new compilers.

- Chapter 2, “Migration Scenarios”

  This chapter describes the various scenarios by which you can migrate to your target LE/VSE environment. The difference between a source migration and run-time migration is also explained.

- Chapter 3, “Migration from Old Languages to New Languages”

  This chapter describes the considerations when migrating from DOS/VS COBOL, VS COBOL II, DOS PL/I or C/370 for VSE, to an LE/VSE-conforming compiler.

- Chapter 4, “Migration from LE/VSE Release 1 to LE/VSE Release 4”

  This chapter describes considerations when migrating from LE/VSE Release 1 to LE/VSE Release 4.

- Chapter 5, “Year 2000 Considerations”

  This chapter describes how LE/VSE and the new compilers can help you to get ready for the Year 2000.

- Chapter 6, “Migration Experiences - Common Questions from Customers”

  This chapter provides migration experiences and answers questions that are often asked by customers.

- Chapter 7, “Interfaces to Other Subsystems”

  This chapter describes how LE/VSE interfaces with other subsystems.

- Chapter 8, “Using LE/VSE Between Languages”

This chapter describes some considerations that may be helpful when migrating pre-LE/VSE interlanguage applications to an LE/VSE environment.

- Appendix A, “Service Information and Performance Considerations”

  This appendix provides technical details relating to the service and performance of LE/VSE. It also provides information about particular issues relating to vendor products.

The Team that Wrote this Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Böblingen Center.

Annegret Ackel, from the International Technical Support Organization Böblingen Center, was the project leader.

Mike Moriarty from ISSC Australia.

Liz Rushton from ISSC Australia.

Thanks to the following person for the advice and guidance provided in the production of this document:

Claudia Prawirakusumah, VSE Development Laboratory, IBM Germany.

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. Introduction to LE/VSE and Benefits of Using LE/VSE

1.1 What is LE/VSE

IBM Language Environment for VSE/ESA (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 COBOL, PL/I and C high level languages (HLLs), and for many applications written in previous versions of COBOL. An LE/VSE-conforming language is any HLL that adheres to the LE/VSE common interface.

Table 1 lists the LE/VSE-conforming language compiler products you can use to generate applications that run with LE/VSE.

Table 1. LE/VSE-Conforming Languages

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

Note:

* Applications written in C/VSE can only run with LE/VSE Release 4.

Any HLL not listed in Table 1 is known as a non-LE/VSE-conforming or, alternatively, a pre-LE/VSE-conforming language. Some examples of non-LE/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 with LE/VSE:

- LE/VSE-conforming languages
- High Level Assembler (HLASM)

  LE/VSE also supports applications written in Assembler language using LE/VSE-provided macros and assembled using HLASM. For details see Chapter 8, “Using LE/VSE Between Languages” on page 63).
- DOS/VS COBOL and VS COBOL II, with some restrictions.

  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 see Chapter 2, “Migration Scenarios” on page 9.

LE/VSE Release 4 supports the Debug Tool for VSE/ESA (DT/VSE) for debugging applications written in any LE/VSE-conforming language.

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.

LE/VSE is the prerequisite run-time environment for applications generated with COBOL/VSE, PL/I VSE or C/VSE. LE/VSE does not include compilers, whereas COBOL/VSE, PL/I VSE and C/VSE are compilers only, they do not include run-time environment as the languages DOS/VS COBOL and VS COBOL II did before.

Release 1 and Release 4 of the LE/VSE product are based on Release 2 and Release 4 of the LE/370 product respectively (now called IBM Language Environment for MVS and VM).
Therefore the basic architecture is the same across the platforms, and it is possible (in many cases) for programs written in LE-conforming languages to be easily adapted to run on an alternative platform.

1.2 Why You Should Migrate - The Year 2000 Aspect

Migration to LE/VSE and the new languages is a requirement for all customers, since DOS PL/I was withdrawn from service in June 1996 and there are no future enhancements planned for DOS/VS COBOL and VS COBOL II. All future enhancements will be to the strategic products LE/VSE with the LE/VSE-conforming languages. Both DOS/VS COBOL and VS COBOL II are not intended to be Year 2000 compliant. So it is in the best interest of the customers to migrate their applications and take advantage of the increased functions and ease of maintenance that LE/VSE and the new compilers contain, as soon as possible.

1.3 What You Can Do with LE/VSE

Note

All functions related to C/VSE are only available with LE/VSE Release 4.

LE/VSE helps you create mixed-language applications and gives you a consistent method of accessing common, frequently used services. LE/VSE promotes efficient application development by providing enhanced capabilities to:

- Properly handle 2-digit years in the Year 2000 and beyond
- Mix legacy code with new code
- Debug applications interactively (with LE/VSE Release 4 only)
- Manage storage dynamically
- Perform date and time calculations
- Access an extensive set of math services
- Share common run-time services
- Handle conditions consistently
- Perform interlanguage communication more efficiently
- Customize routines for international requirements (with LE/VSE Release 4 only).

Most of these functions are available as callable services and they can easily be invoked from the applications.

1.3.1 Properly Handle 2-digit Years in the Year 2000 and Beyond

With LE/VSE′s date and time services you can use existing date formats for the Year 2000 and beyond. To process 2-digit years in the Year 2000 and beyond, LE/VSE employs a sliding scheme - called a century window. Each 2-digit year lies within the 100-year interval defined by the window. When LE/VSE performs a date function, it determines the century of the 2-digit year by using this window and, optionally, returns a 4-digit year.

This means, you can write your applications with the Year 2000 in mind, without having to change the way dates are stored in your existing databases, or you can modify the dates stored in your existing databases by using the 4-digit years provided by LE/VSE date and time services.

1.3.2 Mix Legacy Code with New Code

LE/VSE protects your investment in your applications by minimizing disruption to your system and programming resources as you migrate your current applications. With certain exceptions, LE/VSE provides object module compatibility for applications generated with the DOS/VS COBOL or VS COBOL II compilers, and executable phase compatibility for applications generated with the VS COBOL II compiler. You can also mix, in the same application, routines compiled with these predecessor COBOL compilers with routines compiled with LE/VSE-conforming language compilers.

This upward compatibility means that you can produce new applications largely from older object modules and phases. It also means that you can enhance your applications incrementally. If, for example, you want a routine to call one of the LE/VSE date and time services, you need only recompile the routine and then relink-edit; you do not have to recompile the whole application.

If your applications contain DOS PL/I programs, you must recompile them using the PL/I VSE compiler. With certain exceptions, DOS PL/I programs are source-level compatible with PL/I VSE and can be compiled with the PL/I VSE compiler without change.

If your applications contain C/370 programs, you must recompile them using the C/VSE compiler. C/370 programs are source-level compatible with C/VSE and can be compiled with the C/VSE compiler without change.

Routines that depend on dump formats, error message formats, error handling routines, Assembler routines, and the like, may have to be changed.

Further details about compatibility can be found in Chapter 2, “Migration Scenarios” on page 9.

1.3.3 Debug Applications Interactively

LE/VSE provides a common debugging interface for all LE/VSE-conforming languages. With the debugging tool 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.

1.3.4 Manage Storage Dynamically

Common storage management services are provided for all LE/VSE-conforming languages. LE/VSE controls the storage your routines use at run-time, removing the need for each language to maintain a unique storage manager and avoiding the incompatibilities between different storage mechanisms. Storage management support provides:

- a common heap for all conforming languages
- run-time options for storage tuning
- multiple heap support
- a storage reporting facility
- callable services to dynamically create, allocate, free, and discard heap storage.

LE/VSE′s storage services supplement and cooperate with the existing C *malloc()* and *free()* functions, and with the PL/I ALLOCATE and FREE statements. You can also use the storage services to allocate and free storage dynamically for your COBOL routines, eliminating the need for Assembler.

1.3.5 Perform Date and Time Calculations

With LE/VSE′s date and time services, you can:

- format date and time values according to country or customs formats
- parse date and time values
- convert values between Gregorian, Julian, Asian, and Lilian calendar formats
- calculate days between dates
- calculate elapsed time.

All LE/VSE date and time services conform to national language support guidelines, including full DBCS support.

1.3.6 Access an Extensive Set of Mathematical Services

LE/VSE′s math services are fast, accurate, and accommodate a wide range of data types. With LE/VSE, you can access a rich set of math services from COBOL, PL/I, C, and Assembler routines through a common interface, and thus avoid potentially conflicting results from language-specific math services.

1.3.7 Share Common Run-Time Services

Before LE/VSE, the language-specific run-time libraries provided many similar services, but they were managed independently and were not always consistent across languages.

LE/VSE consolidates essential run-time services for initialization, termination, message handling, condition handling, storage management, national language support, math calculations, and other common programming tasks into a single run-time environment. These services are available, through a common calling interface, to applications produced with LE/VSE-conforming language compilers.

With LE/VSE, many language-specific services use the same underlying callable service. Your application can call a common service, or it can call a language-specific service. Either way, the result is the same: The LE/VSE callable service performs the function, ensuring consistent results across languages.

1.3.8 Handle Conditions Consistently

One of LE/VSE′s most significant benefits is the way it handles conditions. LE/VSE establishes a consistent condition handling method for high-level languages and Assembler language routines that adhere to LE/VSE protocols. LE/VSE also gives you the flexibility to create your own condition handling routines, so you can deal with conditions as you wish.

LE/VSE′s condition handling honors both single- and mixed-language applications, and is integrated with run-time message handling services to provide you with specific information about each condition.

In addition, LE/VSE provides a common dump of the run-time environment for all conforming languages. In the dump you can find, in one place and in an easy-to-read format, traceback information, variable listings, control block information, error condition information, and program status data for all languages used in an application. This means that you need less language-specific knowledge to understand a run-time dump. LE/VSE also provides a common message file where all run-time diagnostics and LE/VSE-generated reports are written.

1.3.9 Perform Interlanguage Communications (ILC) More Efficiently

When you build applications, you want to be able to use program functions wherever they exist, regardless of the language of implementation.

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. This single run-time environment makes interlanguage communication in mixed-language applications easier, more efficient and more consistent. For example, calls between COBOL and PL/I have about 80% less overhead.

This ILC capability also means that you can share and reuse code easily. For instance, you can write a service routine in the language of your choice - COBOL, PL/I, C or Assembler - and allow that routine to be called from COBOL, PL/I, C or Assembler applications. Reusing already developed code instead of repeating the function in more than one language saves application development time, testing, and maintenance costs, and improves the quality of your code.

In addition, LE/VSE enhanced ILC lets you use the best language to be used for any task, and allows you to build applications with components written in a variety of languages. The result is code that runs faster, is less error prone, and is easier to maintain.

1.3.10 Customize Routines for International Requirements

LE/VSE provides a set of data files, called locales, which define coded character sets to reflect the different specific requirements of users of various countries. From your COBOL, PL/I or C routines you can access these pre-defined locales at run time through a set of locale callable services, to customize your routines for national language, culture-specific, and coded character set requirements. With locale callable services, application developers can build programs that can be marketed globally, and still meet the end user′s need to work with specific languages, cultures, and conventions. You can also create your own locales, or modify an IBM-supplied locale, using the locale definition utility supplied with LE/VSE (this utility requires an LE/VSE-conforming C compiler).

1.4 The Benefits of LE/VSE Conforming Language Products

1.4.1 COBOL/VSE

COBOL/VSE is the newest VSE COBOL compiler and is a direct descendant from VS COBOL II and DOS/VS COBOL. The major differences between the compilers are:

- DOS/VS COBOL supports ANSI 74 Standard
- VS COBOL II supports ANSI 85 Standard
- COBOL/VSE supports ANSI 85 Standard with the ANSI 85 Addendum Intrinsic Functions.

The fundamental change from DOS VS COBOL to VS COBOL II and COBOL/VSE is that source modules go from unstructured source code to structured source code. This is a significant change in that it means you will have no more ′spaghetti′ code, that only one person can maintain because they are the only one that understand it.

COBOL/VSE has new language extensions and provides support for LE/VSE features on top of the VS COBOL II language. New with COBOL/VSE is a set of intrinsic functions that enable you to perform mathematical, statistical, financial, character string or date and time calculations with simple invocations from COBOL statements.

With LE/VSE, COBOL users can:

- Get a solution to the Year 2000 problem by using:
  - COBOL/VSE intrinsic functions
  - LE/VSE callable services
- Write applications that utilize 31-bit addressing
- Get increased control over compiler output, such as Associated Data.

1.4.2 PL/I VSE

PL/I VSE enables you to integrate your PL/I applications into LE/VSE. With LE/VSE, PL/I users can:

- Dynamically fetch and call routines, regardless of the LE/VSE-conforming language in which they are written

- Direct messages that were previously directed to SYSPRINT to LE/VSE′s message file
- Write applications that utilize 31-bit addressing

  Although the PL/I VSE compiler runs below the 16MB line, PL/I applications created by the compiler can use the 31-bit addressing feature of VSE/ESA. This allows to take advantage of address space above the 16MB line.
- Make their source programs device independent

  PL/I VSE builds DTFs dynamically at run time. This means you can define input and output device types in run-time JCL, making your source programs device independent.

1.4.3 C/VSE

C/VSE adds support for the 31-bit virtual addressing feature of VSE/ESA:

- The capability to run, in 31-bit addressing mode on VSE/ESA, programs compiled by C/VSE
- The capability to load and execute most of C/VSE compiler and library above the 16MB line on VSE/ESA
- Support of VSE/VSAM buffers stored above the 16MB line.

In addition C users can:

- Use the packed decimal data type when defining data items, thus allowing C to coexist far more in a mixed language environment, that is, interlanguage communication is greatly improved.

1.4.4 COBOL/VSE, PL/I VSE and C/VSE

The common run-time environment that LE/VSE gives you for all these compilers will let you:

- Access LE/VSE′s functions that properly handle 2-digit and 4-digit years in the Year 2000 and beyond
- Use routines written in C, PL/I, or Assembler together with routines written in COBOL - all in the same run-time environment
- Dynamically manage storage effectively using LE/VSE′s callable services
- Tune your applications with information from LE/VSE′s storage report
- Handle error conditions effectively. LE/VSE′s condition handling services enable COBOL/VSE, PL/I VSE, and C/VSE applications to react to error conditions so that nothing but a severe failure need disrupt your application
- Obtain error messages from all conforming languages in one location - the LE/VSE message file
- Develop applications that can be used in multiple countries honoring the conventions of their users or the data they process.

1.4.5 Debug Tool for VSE/ESA

Debug Tool for VSE/ESA (DT/VSE) is a source-level debugger for programs that were compiled under High Level Language compilers that support LE/VSE. DT/VSE is a program-testing and analysis aid that allows users to examine, monitor, and control the execution of their programs. Programs can be dynamically patched to overcome execution failures.

Debugging sessions may be performed in either interactive or batch mode.

The debug capability can be particularly valuable in the context of the Year 2000 challenge. You are likely to find yourself making substantial changes to a significant portion of the application portfolio. These debugging facilities can help you make the testing of those changes productive and comprehensive. Thus DT/VSE can help reduce the total cost and risk of the project.

DT/VSE is supplied as an optional feature with COBOL/VSE, PL/I VSE and C/VSE. If you want to obtain DT/VSE you will only need to order the feature via one of the compilers. DT/VSE will work with all supported LE-enabled programs irrespective of the compiler with which it was ordered.

Chapter 2. Migration Scenarios

This chapter describes the various scenarios by which you can migrate to your target LE/VSE environment. The difference between a source migration and run-time migration is also explained.

The following manuals provide more detailed migration information:

- *IBM COBOL for VSE/ESA Migration Guide*
- *IBM PL/I for VSE/ESA Migration Guide*
- *IBM C for VSE/ESA Migration Guide*
- *IBM Language Environment for VSE/ESA Run-Time Migration Guide* (LE/VSE Release 4 only)

  Note: For LE/VSE Release 1, run-time migration considerations are contained in the corresponding compiler publications.

  See Appendix C, “Related Publications” on page 73 for detailed references.

2.1 Migration Types

A **source migration** consists of updating the application program source if necessary, and compiling the program with an LE/VSE-conforming compiler.

A **run-time migration** consists of link-editing the application program object module with the LE/VSE library modules to produce the program phase. The run-time environment is then changed to access the LE/VSE library modules. Alternatively, in certain circumstances a run-time migration may consist of merely substituting the new run-time library without link-editing.

2.2 COBOL

The COBOL environment is complex due to the number of supported products and the sharing of module and phase names.

The supported products are:

- DOS/VS COBOL 1.3.1
- VS COBOL II (till end of 1997)
- COBOL/VSE (compiler only)
- LE/VSE (run-time only)

The following table shows the valid combinations of COBOL products and the potential conflicts of module and phase names when multiple products are concurrently installed.

Table 2. Valid Scenarios for COBOL Compiler, Link-Edit, and Run Time

If you compile with this compiler...	And you link-edit with this run-time library...	Then you can run with this run-time library...
DOS/VS COBOL FCOB	DOS/VS COBOL ILB	DOS/VS COBOL *
		VS COBOL II *
		LE/VSE *
	VS COBOL II ILB	VS COBOL II *
		LE/VSE *
	LE/VSE ILB	LE/VSE *
VS COBOL II IGY using RES compile-time option	VS COBOL II IGZ	VS COBOL II IGZ
		LE/VSE IGZ
	LE/VSE IGZ	LE/VSE IGZ
VS COBOL II IGY using NORES compile-time option	VS COBOL II IGZ	Not required **
	LE/VSE IGZ	LE/VSE IGZ
COBOL/VSE IGY	LE/VSE IGZ	LE/VSE IGZ

Note:

* For programs compiled by DOS/VS COBOL, there are a few library routines that can be used at run time, but this is application program dependent.

** Prior to APAR PN09126, the VS COBOL II run-time library was required for DTF build routines.

FCOB Prefix for DOS/VS COBOL compiler modules.

ILB Prefix for DOS/VS COBOL run-time modules.

IGY Prefix for VS COBOL II and COBOL/VSE compiler modules.

IGZ Prefix for VS COBOL II run-time modules and COBOL language component run-time modules of LE/VSE.

Table 2 shows that for COBOL:

- There is a possibility of module name conflict when old and new products are installed concurrently,
- It is possible to use new LE/VSE run-time with older COBOL-compiled programs before implementing the COBOL/VSE compiler.

If you are a COBOL user, there are choices available so you must decide on a migration strategy. LE/VSE provides object compatibility for programs compiled with either DOS/VS COBOL or VS COBOL II. Therefore, in most cases it is possible to migrate the run-time component first and then gradually introduce the new LE-conforming compiler.

The migration scenarios for COBOL are:

- Source
  - DOS/VS COBOL 1.3.1 to COBOL/VSE
  - VS COBOL II (CMPR2) to COBOL/VSE
  - VS COBOL II (NOCMPR2) to COBOL/VSE

- Run Time
  - DOS/VS COBOL 1.3.1 to LE/VSE
  - VS COBOL II to LE/VSE
  - LE/VSE Release 1 to LE/VSE Release 4

Recompile and/or relinkedit?

- When must I recompile?
  - For COBOL, it depends on whether source code has been amended to conform with COBOL/VSE requirements or if LE/VSE callable services are being introduced into the program.
- When must I relink?
  - Always if recompile has been done.
  - If no recompile for COBOL, it depends ..., but highly recommended.
  - Few customers retain object modules, so usually recompile is done.

2.3 Migrating from DOS PL/I to PL/I VSE and LE/VSE

The migration to the LE/VSE environment consists of a source migration to PL/I VSE and run-time migration to LE/VSE. There is no object compatibility between DOS PL/I and LE/VSE so all application programs must be recompiled with the LE-conforming compiler and link-edited using the LE/VSE library.

Table 3. Valid Scenarios for PL/I Compiler, Link-Edit, and Run Time

If you compile with this compiler...	**And you link-edit with this run-time library...**	**Then you can run with this run-time library...**
DOS PL/I PLI	DOS PL/I IBM	DOS PL/I IBM
PL/I VSE IEL	LE/VSE IBM	LE/VSE IBM

Note:

Although the module prefixes are shared, all module names are unique

PLI Prefix for DOS PL/I compiler modules.

IEL Prefix for PL/I VSE compiler modules.

IBM Prefix for DOS PL/I and LE/VSE PL/I run-time modules.

Table 3 shows that for PL/I:

- There is no module name conflict when old and new products are installed concurrently,
- It is not possible to use new LE/VSE run-time with older PL/I-compiled programs before implementing the PL/I VSE compiler.

The migration scenarios for PL/I are:

- Source
  - DOS PL/I to PL/I VSE

- Run Time
  - DOS PL/I to LE/VSE
  - LE/VSE Release 1 to LE/VSE Release 4

2.4 Migrating from C/370 to C/VSE and LE/VSE

The migration to the LE/VSE environment consists of a source migration to C/VSE and run-time migration to LE/VSE. There is no object compatibility between C/370 and LE/VSE so all applications must be recompiled with the LE-conforming compiler and link-edited using the LE/VSE library.

Note: C run-time support was introduced in LE/VSE Release 4.

Table 4. Valid Scenarios for C Compiler, Link-Edit, and Run Time

If you compile with this compiler...	**And you link-edit with this run-time library...**	**Then you can run with this run-time library...**
C/370	C/370	C/370
C/VSE	LE/VSE[1]	LE/VSE[1]

Note:

1. LE/VSE Release 4 only

The module prefixes are numerous in these products, and names are not unique between the products.

Table 4 shows that for C:

- It is not possible to use new LE/VSE run-time with older C-compiled programs before implementing the C/VSE compiler.
- There are module name conflicts when old and new products are installed concurrently.

The migration scenarios for C are:

- Source
  - C/370 to C/VSE
- Run Time
  - C/370 to LE/VSE Release 4

Chapter 3. Migration from Old Languages to New Languages

This chapter describes the considerations when migrating from an old compiler to an LE/VSE-conforming compiler. An ″old″ compiler is one of:

- DOS/VS COBOL
- VS COBOL II
- DOS PL/I
- C/370 for VSE

For a complete list of migration considerations, see the appropriate migration guide:

- *IBM COBOL for VSE/ESA Migration Guide*
- *IBM PL/I for VSE/ESA Migration Guide*
- *IBM C for VSE/ESA Migration Guide*
- *IBM Language Environment for VSE/ESA Run-Time Migration Guide* (LE/VSE Release 4 only)

  Note: For LE/VSE Release 1, run-time migration considerations are contained in the corresponding compiler publications.

  See Appendix C, “Related Publications” on page 73 for detailed references.

Recommendation

Where possible do a run-time migration to LE/VSE first, then carefully plan and implement a source migration. New development can be done with the LE/VSE-conforming compiler.

3.1 Source Migration

Source migration consists of amending the source if necessary and recompiling the program. If you have programs that run using CICS, translate the programs, using the CICS/VSE-supplied translator.

A source migration is required to:

- Make changes because of language syntax differences
- Add code to take advantage of new language functionality
- Make changes to provide for Year 2000 support
- Correct poor programming techniques
- Implement LE/VSE-conforming compiler

3.1.1 COBOL Source Migration

Your target environment should be source that can be compiled with COBOL/VSE and the NOCMPR2 compile option. If you have previously migrated to VS COBOL II and NOCMPR2 then your source will not need to be changed.

NOCMPR2

NOCMPR2 compiler option provides the full ANSI 85 implementation of COBOL/VSE and VS COBOL II.

CMPR2 compiler option provides compatibility with VS COBOL II Release 2 and is provided as an aid to migration (along with the FLAGMIG option). New COBOL/VSE language elements, such as intrinsic functions, are not supported under CMPR2.

Migrating from all other environments will require varying degrees of source changes. Consult the *IBM COBOL for VSE/ESA Migration Guide* for specific details.

3.1.1.1 Hints and Tips

- Change compiler name from FCOBOL to IGYCRCTL

  The COBOL/VSE compiler is invoked by executing phase IGYCRCTL. If you are migrating from DOS/VS COBOL then ensure that your JCL is changed to execute IGYCRCTL instead of FCOBOL.
  If you are converting from VS COBOL II, then no change is required.

- Modify SELECT and ASSIGN clauses

  The format of the SELECT and ASSIGN clauses has changed to a much simpler form. COBOL/VSE will in some cases allow the DOS/VS COBOL coding format but may produce unexpected results at run time due to a different interpretation of the ASSIGN clause format.

  For example, the statement in DOS/VS COBOL
  (for a standard label tape file):
  SELECT FILEA ASSIGN TO SYS010-UT-3420-S
  must be amended to:
  SELECT FILEA ASSIGN TO *filename*
  where *filename* must be a valid
  VSE/ESA filename that matches the name
  specified on the // TLBL JCL statement.
  For example : // TLBL *filename*,'Tape file label'

- Remove unsupported language elements

  These are comprehensively documented in the *IBM COBOL for VSE/ESA Migration Guide*. Examples include the removal of Report Writer and ISAM support.

- Include Year 2000 changes

  Although this is potentially a major exercise, you have a unique opportunity with the implementation of COBOL/VSE and LE/VSE to amend program source to be Year 2000 compliant. This may include incorporating new LE/VSE Date/Time callable services or using COBOL/VSE intrinsic functions to provide 4 digit years. See Chapter 5, “Year 2000 Considerations” on page 33 for more information.

- Avoid file attribute mismatches

  DOS/VS COBOL file open processing does not comprehensively check that the file definition in your program exactly matches the file definition (as defined in the VTOC for a SAM file or a LISTCAT for a VSAM file). Because of ANSI 85 requirements, LE/VSE COBOL open processing carries out many detailed checks for consistency between program and actual file definition before opening the file. This can result in file open failures in LE/VSE COBOL even though no changes have been made. The best approach is to add a file status check following each OPEN statement. Then, if these subsequently indicate problems amend your program appropriately.

- Ensure ANSI85 option used for CICS translation

  Either ANSI85 or its equivalent COBOL2 must be specified in the XOPTS CICS/VSE translator options for CICS/VSE Command Level Programs. (The keyword COBOL2 is valid for both VS COBOL II and COBOL/VSE programs.)
  This has two main effects on the translated code. Firstly, the translated program will contain the RENT compiler option which is required by programs running with LE/VSE and CICS/VSE. Secondly, the resulting COBOL/VSE code generated by translating certain EXEC CICS statements is necessarily different for the ANSI85 option.

3.1.2 PL/I Source Migration Hints and Tips

- Change compiler name from PLIOPT to IEL1AA

  It is likely during the migration period that both old and new PL/I products will coexist on your system. (This is made easier because there are no module name conflicts between the two products.) The PL/I VSE compiler is invoked by executing phase IEL1AA. Ensure that your JCL is changed to execute the correct compiler.

- Convert programs using CHARSET(48 and BCD)

  The 48-character set is not supported by PL/I VSE. A program has been provided to assist in the migration of your source code. You must order PTF UN92140 to obtain this program.

- Remove unsupported compiler options

  Several DOS PL/I compiler options are not supported by PL/I VSE. Although the compiler will ignore them, it is preferable to remove these from JCL if specified. Refer to *IBM PL/I for VSE/ESA Migration Guide* for specific options removed.

- Include Year 2000 changes

  Although this is potentially a major exercise, you have a unique opportunity with the implementation of PL/I VSE and LE/VSE to amend program source to be Year 2000 compliant. This may include incorporating new LE/VSE Date/Time callable services or using PL/I VSE functions to provide 4-digit years. See Chapter 5, “Year 2000 Considerations” on page 33 for more information.

- Change CICS linkedit JCL to include DFHELII

  The interface to CICS/VSE is provided by LE/VSE for PL/I VSE programs. Therefore, you must change your linkedit JCL to include DFHELII instead of the DOS PL/I interface modules DFHPL1I and DFHEPI.

3.1.3 C Source Migration Hints and Tips

- The compiler name EDCCOMP is the same as C/370
- Source for C/VSE program can be included directly from a VSE Librarian member or from a sequential file by using the INFILE compiler option
- Include Year 2000 changes

3.2 Source Migration Tools

- COBOL

  COBOL and CICS Command Level Conversion Aid (CCCA).
  This is a program offering that converts ANSI 68 and 74 source to ANSI85 source. It will flag any statement that it cannot convert and report on all action taken.
  The product will convert both batch and CICS source.

- PL/I

  A 48-character to 60-character set conversion utility.
  The 48-character set (CHARSET option) is not supported under PL/I VSE. This utility provides source conversion of DOS PL/I programs using this feature. This program is provided via a PTF (UN92140) orderable through the normal service procedures.

3.3 Run-time Migration

Run-time migration is required to:

- Position to prepare for source migration in ″compatibility mode″

  If you are able to migrate the run time first (COBOL only) then you can prepare for a source migration by implementing the new LE/VSE run-time components in compatibility mode.

  Compatibility mode

  ″Compatibility mode″ means executing old programs compiled with pre-LE/VSE-conforming languages and executed using the LE/VSE run-time library. LE/VSE provides compatibility mode for both DOS/VS COBOL and VS COBOL II programs.

- Implement source migration to LE/VSE-conforming compiler.

  To run any program compiled with an LE/VSE-conforming compiler, you must use LE/VSE run-time libraries.

Recommendation

SVA usage - although there will be recommendations to place LE/VSE phases in the SVA, consider these very carefully during the migration period. This is because there may be name conflicts with existing products (in the case of COBOL and C) which could result in accessing an incorrect copy of a phase. During the migration period try to avoid placing phases in the SVA for products that have conflicting phase names.

3.3.1 COBOL Run-time Migration

The potential complexity of the COBOL run-time environment is shown in Table 2 on page 10.
By implication, you must ensure that the correct version of a module is accessed by:

- Having the correct LIBDEF order
- Avoiding use of the SVA if possible during migration
- Being aware of COBPACK contents

3.3.1.1 Hints and Tips

- Access the correct modules

  LE/VSE COBOL has modules with names common to both VS COBOL II and DOS/VS COBOL. Also, some LE/VSE COBOL batch modules have the same names as LE/VSE COBOL CICS modules.
  It is important to ensure that the correct module is accessed.
  Unless overridden, by default LE/VSE modules reside in VSE library PRD2.SCEEBASE, except LE/VSE COBOL CICS modules which reside in PRD2.SCEECICS.

  – Migrating from DOS/VS COBOL

    Batch PRD2.SCEEBASE ahead of PRD2.SCEECICS ahead of PRD2.DBASE

    CICS PRD2.SCEECICS ahead of PRD2.SCEEBASE ahead of PRD2.DBASE

  – Migrating from VS COBOL II

    Batch PRD2.SCEEBASE ahead of PRD2.SCEECICS ahead of PRD2.PROD and PRD2.CICSR

    CICS PRD2.SCEECICS ahead of PRD2.SCEEBASE ahead of PRD2.CICSR and PRD2.PROD

- Tape or Disk Manager - see your OEM Vendor for required fixes

  If your installation has a Third-Party Tape or Disk Manager such as CA-DYNAM or CA-EPIC, then contact the vendor for appropriate fixes for LE/VSE.

- ALL31(ON) default in CICS must be OFF to run AMODE 24 applications
- Assess suitability of both batch and CICS run-time options

  There are different LE/VSE supplied default options for batch and CICS. These should be examined and adjusted for suitability in your environment.

3.3.2 PL/I Run-time Migration

LE/VSE PL/I and DOS PL/I have unique modules names. Both products can coexist without interference even in a CICS environment.

3.3.2.1 Hints and Tips

- Check run-time options particularly ALL31, STACK and HEAP
- CICS - you don't need PLI=YES in startup parameters

3.3.3 C Run-time Migration

LE/VSE and C/370 for VSE (library) have common module names and therefore there is a potential conflict if these products are installed concurrently on the same system.

This conflict exists for both LE/VSE Release 1 and Release 4.

3.3.3.1 Hints and Tips

- Ensure that PRD2.SCEEBASE is ahead of PRD2.PROD in the LIBDEF chain
- If running C/370 ensure that PRD2.PROD is ahead of PRD2.SCEEBASE in the LIBDEF chain
- Avoid the SVA during the migration period
- Always try to ensure that your C/VSE program is linkedited AMODE 31 and RMODE ANY to avoid problems accessing LE/VSE phases loaded above the 16MB line and to gain optimal storage management.

Chapter 4. Migration from LE/VSE Release 1 to LE/VSE Release 4

This chapter describes considerations when migrating from LE/VSE Release 1 to LE/VSE Release 4.

4.1 Why Migrate?

You will probably want to take advantage of the many new features and facilities provided in LE/VSE Release 4.

In addition to this, LE/VSE Release 1 will be withdrawn from marketing in 12/96 and from service in 12/97.

4.2 New in LE/VSE Release 4

LE/VSE Release 4 (program number 5686-094) is a major functional enhancement to LE/VSE Release 1 (program number 5686-067); there were no intermediate releases.

LE/VSE Release 4 is based on Language Environment for MVS and VM (LE/370) Release 4 (but without multi-tasking support). New features introduced include:

- C Support
- Support for Debug Tool for VSE/ESA
- New and changed run-time options
- New callable services and other functions

4.2.1 C Support

C language run-time support has been added for C applications compiled with the new LE/VSE-conforming C language compiler.

4.2.2 Debug Tool for VSE/ESA

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 Debug Tool for VSE/ESA assumes control when the user application is being initialized. For meaningful results using Debug Tool for VSE/ESA, your program must have been compiled with the compiler TEST option set.

4.2.3 New Run-time Options

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 switches or tagged information into an application 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.

For more information about these new options, see *IBM Language Environment for VSE/ESA Programming Reference*, SC33-6685.

4.2.4 New Callable Services

The following callable services have 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.

CEECBLDY Converts a string representing a date into a COBOL integer format that is compatible with ANSI COBOL intrinsic functions. For an example of the use of CEECBLDY, see Chapter 5, “Year 2000 Considerations” on page 33.

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().

For more information about these new callable services, see *IBM Language Environment for VSE/ESA Programming Reference*, SC33-6685.

4.2.5 Other New Functions

- Predefined locales for specifying different national language and cultural conventions have been added.
- A locale definition utility for modifying and creating locales has been added (requires LE/VSE-conforming C language compiler).
- A DSECT conversion utility to generate a C structure to map an Assembler DSECT.
- Nested enclaves can now be created by the C `system()` function.

Note: In LE/VSE, an **enclave** is 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.

4.3 Changes from LE/VSE Release 1

4.3.1 TEST|NOTEST Run-Time Option

With LE/VSE Release 1, the TEST|NOTEST run-time option was included for compatibility with LE/370. If you specified the TEST|NOTEST run-time option, it was syntax-checked but had no effect on the run-time behavior of your application.

With LE/VSE Release 4, the TEST|NOTEST run-time option is fully supported. The TEST option specifies the conditions under which a debug tool, such as Debug Tool for VSE/ESA, assumes control when the user application is being initialized.

4.3.2 RPTOPTS Report Differences

With LE/VSE Release 1, the run-time options report produced by the RPTOPTS(ON) run-time option included options included in Release 1 for compatibility with LE/370, and the PL/I-compatibility run-time options ISAINC and ISASIZE.

With LE/VSE Release 4, these run-time options are not included in the report. The run-time options included in Release 1 for compatibility with LE/370 are not included in Release 4. See page 22 for a list of these options.

4.3.3 RPTSTG Report Differences

With LE/VSE Release 1, the storage report produced by the RPTSTG(ON) run-time option used the phrases “Successful GETMAINs issued” and “Successful FREEMAINs issued” to describe the number of requests LE/VSE issued to the operating system (or CICS) to get storage and to release storage.

With LE/VSE Release 4, these phrases are replaced by “Number of segments allocated” and “Number of segments freed,” respectively.

For more information about these changed options, see *IBM Language Environment for VSE/ESA Programming Reference*, SC33-6685.

4.3.4 Run-time Option Installation Defaults Changed from Release 1

Several run-time options have different IBM-supplied installation default values in LE/VSE Release 4 than in LE/VSE Release 1. The following table lists those run-time options that have different installation default values in batch.

Table 5. Comparison of Run-Time Option Installation Default Values (Batch)

LE/VSE Release 1 Default	**LE/VSE Release 4 Default**
ANYHEAP(32K,16K,ANYWHERE,FREE)	ANYHEAP(16K,8K,ANYWHERE,FREE)
BELOWHEAP(32K,16K,FREE)	BELOWHEAP(8K,4K,FREE)
HEAP(64K,64K,ANYWHERE,KEEP,16K,16K)	HEAP(32K,32K,ANYWHERE,KEEP,8K,4K)
LIBSTACK(32K,16K,FREE)	LIBSTACK(8K,4K,FREE)
NOTEST(NONE,*,NOPROMPT,*)	NOTEST(ALL,*,PROMPT,¢¢)
STACK(512K,512K,BELOW,KEEP)	STACK(128K,128K,BELOW,KEEP)

The following table lists those run-time options that have different installation default values in CICS.

Table 6. Comparison of Run-Time Option Installation Default Values (CICS)

LE/VSE Release 1 Default	**LE/VSE Release 4 Default**
NOTEST(NONE,*,NOPROMPT,*)	NOTEST(ALL,*,PROMPT,¢¢)
TERMTHDACT(TRACE)	TERMTHDACT(MSG)

For information about customizing run-time option installation default values, see *IBM Language Environment for VSE/ESA Installation and Customization Guide*, SC33-6682.

4.3.5 Run-time Options Removed since LE/VSE Release 1

With LE/VSE Release 1, you could specify the following run-time options, for compatibility with LE/370.

CBLQDA

FLOW|NOFLOW

INTERRUPT

SIMVRD|NOSIMVRD

VCTRSAVE

If you specified any of these run-time options, it was syntax-checked, but had no effect on the run-time behavior of your application.

With LE/VSE Release 4, these run-time options are no longer supported, even for compatibility purposes. If you specify any of these options, you will receive the run-time message CEE3609I, and the option is ignored.

If you specify them when assembling installation default options you will receive an Assembler warning message.

4.3.6 Abnormal Termination Considerations

4.3.6.1 LE/VSE-supplied Assembler User Exit for CICS

The Assembler user exit for CICS supplied with LE/VSE Release 1 always requested an abend when an enclave ended with the following types of unhandled LE/VSE condition of severity 2 or greater, regardless of the setting of the ABTERMENC run-time option:

- A software-raised condition, such as the condition raised by LE/VSE if you try to run an AMODE 24 program without specifying the ALL31(OFF) and STACK(,,BELOW) run-time options
- A user-raised condition (raised by a call to the CEESGL callable service)

This IBM-supplied Assembler user exit for CICS did not provide an abend code, so LE/VSE Release 1 used an abend code based upon the condition severity. A severity 2 condition produced an abend code of 2000, a severity 3 condition produced an abend code of 3000, and a severity 4 condition produced an abend code of 4000.

The Assembler user exit for CICS supplied with LE/VSE Release 4 does not specifically request an abend when an enclave terminates with an unhandled condition of severity 2 or greater. Instead, the ABTERMENC run-time option in effect at the time is honored. If ABTERMENC(ABEND) is in effect during abnormal termination, the enclave is terminated with abend code 4038.

4.3.6.2 LE/VSE-supplied Abnormal Termination Exits

LE/VSE Release 1 and Release 4 both supply a batch and a CICS abnormal termination exit CSECT. The supplied batch abnormal termination exit CSECT is CEEEXTAN, and the supplied CICS abnormal termination exit CSECT is CEECXTAN.

CEEEXTAN: The CEEEXTAN CSECT supplied with LE/VSE Release 1 did not supply any batch abnormal termination exit. The CEEEXTAN CSECT supplied with LE/VSE Release 4 supplies a null module, CEEBDATX. This exit is driven whenever a batch enclave terminates abnormally. It simply returns to its caller, and does not generate a dump of the partition.

CEECXTAN: The CEECXTAN CSECT supplied with LE/VSE Release 1 did not supply any CICS abnormal termination exit. The CEECXTAN CSECT supplied with LE/VSE Release 4 supplies a module, CEECDATX. This exit is driven whenever an enclave in CICS terminates abnormally. When this happens this abnormal termination exit requests a CICS transaction dump with a dump code of 4039. The CICS transaction dump is produced in addition to the abnormal termination information produced by LE/VSE under the control of the TERMTHDACT run-time option. The LE/VSE abnormal termination information is written to the CESE transient data queue. The CICS dump is written to the CICS dump data set.

You can customize the Assembler user exits and the abnormal termination exits to suit your requirements. For more information about customizing Assembler user exits and abnormal termination exits see *IBM Language Environment for VSE/ESA Installation and Customization Guide*, SC33-6682.

4.4 Packaging of LE/VSE

4.4.1 Changes to the Component Packaging of LE/VSE

The packaging of LE/VSE has changed between Release 1 and Release 4. The program number of LE/VSE Release 1 was 5686-067, with ten components numbered from 5686-067-00-18A to 5686-067-09-18J consecutively.

The usual procedure with a new release of a product is to maintain the same program number but allocate new component level codes. However, for LE/VSE Release 4, the whole product identification has changed. The program number for LE/VSE Release 4 is 5686-094, with nine components numbered from 5686-094-01-1DS to 5686-094-09-1F3. There were no intermediate releases between Release 1 and Release 4.

LE/VSE Release 1 was packaged with separate components for each national language feature, that is uppercase U.S English, mixed-case English and Japanese. LE/VSE Release 4 is packaged with each mixed-case English feature incorporated with the uppercase U.S English feature in the relevant base component.

The numbers of the publications have changed. LE/VSE Release 1 publication numbers were in the SC26-8... series. LE/VSE Release 4 publication numbers are in the SC33-66.. series. Several publications have been added to the series for LE/VSE Release 4, and two have been removed, *LE/VSE Diagnosis Guide* and *LE/VSE Reference Summary*. Refer to the bibliography in any LE/VSE Release 4 publication for a complete list of available publications. Refer to *LE/VSE Installation and Customization Guide* Release 4 for information about ordering LE/VSE publications.

4.4.2 Components of LE/VSE

LE/VSE Release 1 comprised three parts. They were the LE common components, the LE COBOL components and the LE PL/I components. The total number of components in LE/VSE Release 1 was ten.

LE/VSE Release 4 comprises four parts. They are the LE common components, the LE COBOL components, the LE PL/I components and the LE C components. However, the total number of components in LE/VSE Release 4 is nine.

The reason for the reduction in the total number is that with LE/VSE Release 4, the mixed-case English component in each part has been incorporated into the relevant base component. Table 7 shows the components of LE/VSE Release 1 and Release 4.

Table 7 (Page 1 of 2). Components of LE/VSE Release 1 and Release 4

	Release 1	**Release 4**
LE Common	LE common base component	LE common base component with mixed-case English[1]
	LE mixed-case English	–
	LE Japanese national language feature	LE Japanese national language feature[1]

Table 7 (Page 2 of 2). Components of LE/VSE Release 1 and Release 4

	Release 1	Release 4
LE COBOL	COBOL-specific base component	COBOL-specific base component with mixed-case English
	COBOL-specific mixed-case English	–
	COBOL-specific Japanese national language feature	COBOL-specific Japanese national language feature
	COBOL-specific CICS component	COBOL-specific CICS component
LE PLI	PL/I-specific base component	PL/I-specific base component with mixed-case English
	PL/I-specific mixed case English	–
	PL/I-specific Japanese national language feature	PL/I-specific Japanese national language feature
LE C	n/a	C-specific base component[1]
	n/a	–
	n/a	LE C Japanese national language feature[1]

Note:

1. These components comprise the run-time support for C shipped on the VSE/ESA 2.2 extended base tape. See 4.4.4, “C Language Run-time Support Shipped with VSE/ESA 2.2” on page 26.

The reason for a separate CICS-specific component in LE COBOL, in both releases, is that there are modules in LE COBOL batch and CICS with the same names.

4.4.3 Which Components to Install?

As with LE/VSE Release 1, you do not need to install all the components of LE/VSE Release 4 if you do not need them.

For example, if you do not need the Japanese national language components, omit these components when you install the product.

Similarly, if your installation uses only COBOL as the standard programming language, you do not need to install the PL/I or C components. However, you must always install the LE common or base component.

Notes:

There are two exceptions to this:

1. If you plan to use the LE/VSE locale callable services, introduced in LE/VSE Release 4, you must install the LE C base component, even if you do not intend to use the run-time support for C.
2. If you plan to use Debug Tool/VSE to debug your applications, you must install the LE C base component, regardless of the programming language of the applications you plan to debug.

4.4.4 C Language Run-time Support Shipped with VSE/ESA 2.2

Run-time support for C, (referred to as VSE C Language Run-time Support), is shipped on an extended base tape with VSE/ESA 2.2. This comprises the components of LE/VSE needed to provide the run-time environment for C programs compiled with an LE/VSE-conforming C language compiler.

These components are the two LE common components and the two LE C components, making a total of four. See the note to Table 7 on page 24. The product number and component level codes for these four components are the same, whether they are installed from the C language run-time support or the LE/VSE Release 4 product.

As with the complete LE/VSE Release 4 product, if you do not need the Japanese national language components, do not install them.

4.4.5 Do I need the LE/VSE Release 4 Product as well as VSE C Language Run-time Support

If you currently have applications that use the LE COBOL or LE PL/I components of LE/VSE Release 1, and you plan to migrate these to LE/VSE Release 4, you must order the complete LE/VSE Release 4 product, and install the LE COBOL or LE PL/I components as well as the LE common or base components. You cannot use the LE COBOL and LE PL/I components from LE/VSE Release 1 with the LE common components from the VSE C Language Run-time Support.

If you have installed VSE C Language Run-time Support and you later decide to install the complete LE/VSE Release 4 product, there are some planning issues you should consider.

Maintenance may have been applied to VSE C Language Run-time Support, or to LE/VSE Release 4, since they were released.

Therefore, before you commence the installation of LE/VSE Release 4, check with your IBM Support Center for any maintenance that you may need.

If you have applied maintenance to VSE C Language Run-time Support since you first installed it, check that this maintenance is also applied to the LE/VSE Release 4 product that you receive. If it is not, you need to make a note of the APARs concerned and apply them again when you have installed LE/VSE Release 4.

You should install the complete LE/VSE Release 4 product, and not just the components you did not install when you installed VSE C Language Run-time Support.

The product number and component level codes for VSE C Language Run-time Support components are the same as for these components distributed with LE/VSE Release 4. When you install LE/VSE Release 4, these components will be overwritten in the system history file, and in the sublibrary, if you use the same sublibraries. Therefore, before installing LE/VSE Release 4 over VSE C Language Run-time Support, you should consider deleting VSE C Language Run-time Support from the sublibrary and releasing the library space.

4.4.6 Other Packaging Changes Since LE/VSE Release 1

4.4.6.1 SVA Load Lists

Loading the SVA has been made easier. If you intend to load all the eligible routines, or just the recommended modules, into the SVA, then you can use one or more of the SVA load lists supplied. These SVA load lists make it easier for you to load the required modules into the SVA without needing to list each module by name. Table 8 gives the names of these load lists.

Table 8. LE/VSE Supplied SVA Load Lists

Phase Name	Description
$SVACEE	All LE/VSE base routines eligible for the SVA except callable service stubs and Japanese modules
$SVAIGZM	The LE/VSE COBOL component routines listed as recommended for inclusion in the SVA
$SVAIGZ	All LE/VSE COBOL component routines eligible for the SVA, assuming COBPACKs as distributed, except Japanese modules
$SVAIBMM	The LE/VSE PL/I component routines listed as recommended for inclusion on the SVA
$SVAIBM	All LE/VSE PL/I component routines eligible for the SVA except Japanese modules
$SVAEDCM	The LE/VSE C component routines listed as recommended for inclusion on the SVA
$SVAEDC	All LE/VSE C component routines eligible for the SVA except Japanese modules, locales and code page converters

To load these lists automatically, modify your VSE BG ASI procedure as follows.

After the SET SDL statement, add the statement:

LIST=$SVA*xxxx*

for each list to be loaded.

As with LE/VSE Release 1, if you intend to load modules into the SVA, use LE/VSE under CICS, or make use of COBOL COBPACKs, you should plan your installation carefully, so that you do not encounter a conflict between modules in the SVA, in COBPACKs and in sublibraries, or use the wrong modules in your CICS environment.

4.4.6.2 Supplied Sample Job Control

Where possible, the sample job control for the various functions and verification programs has been enhanced to use the IESINSRT utility supplied with VSE/ESA and the DISP=I punch facility of VSE/POWER.

This means that the only changes you need to make to this job control are specific changes required by your installation, such as POWER JECL or sublibrary changes.

As well as supplying the sample job control as Z books in the installation sublibraries, these samples are also supplied as skeletons in ICCF library 62.

4.4.6.3 Supplied DCT Samples

The sample DCT supplied with LE/VSE Release 1 has been changed to include the new transient data queue, CESO, required by the C run-time support in CICS.

A sample DCT macro was supplied with CICS/VSE 2.3 for LE/VSE Release 1 directing the CESE transient data queues to SYSLST. The sample supplied originally included only CESE as required by LE/VSE Release 1. CESO has been added and the new version of this sample DCT is available by applying the CICS/VSE APAR PN81743.

A sample DCT macro is supplied with VSE/ESA 2.2 which contains definitions for CESE and CESO. These definitions route CESE and CESO output to the IESL transient data queue. You can then use the ″Inspect Message Log″ dialog to retrieve this output.

Note: The IESL transient data queue only supports output lines up to 132 characters in length. LE/VSE messages, dumps and reports can often be longer than this, as in addition to the body of the message lines, there are 25 bytes at the front of the message line containing the terminal ID, transaction ID, and the date/time stamp. Therefore LE/VSE messages, dumps, and reports routed to IESL may be truncated.

The format of an LE/VSE transient data queue entry is shown in Figure 1.

ASA	Terminal ID	Transaction ID	sp	Timestamp YYYYMMDDHHMMSS	sp	Message
1	4	4	1	14	1	132

Figure 1. Format of a Transient Data Queue Entry

ASA	The American National Standard carriage-control character
Terminal ID	A four-character terminal identifier
Transaction ID	A four-character transaction identifier
sp	A space
Timestamp	The date and time displayed in the same format as that returned by the CEELOCT service
Message	The message identifier and message text

4.4.6.4 CSD Entries

The sample job to load the CSD has been enhanced to use SLI input. This means that if you run this supplied job, CEEWCCSD, all the required CSD entries will be made without the need to list each entry separately. It is also easier to remove the entries for components you have not installed.

When installing LE/VSE Release 4 for use under CICS, you must reload the CSD file with the new CSD entries supplied with Release 4. The CSD entries you loaded with LE/VSE Release 1 will not enable Release 4 to initialize under CICS.

4.5 Where to Install LE/VSE Release 4

The default sublibraries for LE/VSE (Release 1 and Release 4) are PRD2.SCEEBASE and PRD2.SCEECICS. Therefore, if you install LE/VSE into the default sublibraries, you must plan your migration from Release 1 to Release 4 carefully.

You cannot mix LE/VSE Release 1 and LE/VSE Release 4 modules. If you plan to have both releases current for some time (for example, while you verify the function of Release 4), then you must install Release 4 in different sublibraries than Release 1.

When you use LE/VSE Release 4 you must ensure that the Release 4 sublibraries are ahead of the Release 1 sublibraries in your LIBDEF search chain. (It is probably best to remove the Release 1 sublibraries from the search chain altogether.)

Before running with LE/VSE Release 4, ensure that there are no LE/VSE Release 1 phases present in the SVA. If necessary remove any Release 1 phases from your SDL definitions in your IPL procedures and re-IPL your VSE system.

If you do not plan to run LE/VSE Release 4 in parallel with LE/VSE Release 1, you can install LE/VSE Release 4 into the default sublibraries.

However, in order to ensure that you have enough space in your PRD2 library, it is recommended that you delete Release 1 from the sublibraries first and release the freed space by running the Librarian RELEASE command against this library.

For a description of how to delete LE/VSE Release 1, see *IBM Language Environment Installation and Customization Guide* for LE/VSE Release 1, SC26-8064.

Alternatively, a skeleton JCL to delete LE/VSE Release 1 is provided in ICCF library 59. This skeleton is called DELLEVSE.

4.6 Changes to Compiler Packaging

The LE/VSE-conforming languages are COBOL/VSE, PL/I VSE and C for VSE/ESA (C/VSE).

C/VSE is new and available at the same time as LE/VSE Release 4. It is a C application and therefore requires the C components of LE/VSE Release 4 to run.

COBOL/VSE and PL/I VSE have been available since April 1995. However, new distribution tapes of these compilers will be available at the same time as LE/VSE Release 4, comprising the original product together with maintenance applied. COBOL/VSE and PL/I VSE will run with LE/VSE Release 1 or Release 4.

In addition, on this date these compilers will be available as either full function or alternate function. Alternate function will include only the compiler, full function will include the compiler and Debug Tool for VSE/ESA.

Debug Tool for VSE/ESA is also a C application and will therefore require the C component of LE/VSE Release 4 to run, regardless of the programming language of the applications you plan to debug.

If you plan to install LE/VSE Release 4 with COBOL/VSE or PL/I VSE and you have not previously installed LE/VSE Release 1, then you should be aware of a potential problem if you do not take a refresh version of the required compiler.

COBOL/VSE and PL/I VSE, as distributed from April 1995, had prerequisite conditions for LE/VSE Release 1 components. Due to the change in the product identification number between LE/VSE Release 1 and Release 4, if you attempt to install an older version of COBOL/VSE or PL/I VSE, without LE/VSE Release 1 installed, MSHP will not install the required compiler.

You will receive the MSHP error message:

M223I INSTALL TERMINATED - REQUISITE CHECK FAILED

This situation will typically arise if you try to install a copy of the required compiler that was distributed on the optional program tapes with VSE/ESA 1.4 or VSE/ESA 2.1.

The best solution to this problem is to take the refresh version of your required compiler. However, if this is not acceptable you can circumvent the problem by running an MSHP job to archive the required components into your system history file.

The MSHP statements required to do this are shown in Figure 2 for COBOL/VSE and Figure 3 on page 31 for PL/I VSE.

```
// EXEC MSHP,SIZE=900K
  ARCHIVE 5686-067-00-18A
  ARCHIVE 06718A
  RESOLVES ¢LEVSE.BLE..1.1.0 LE/VSE BASE RELEASE 1.0¢
  COMPRISES 5686-067-00  -
     PHASES   = (CEE*) -
     MODULES  = (CEE*) -
     MACROS   = ($$$CO18A,CEE*) TYPE=Z
  RESIDENCE PRODUCT = (06718A)       -
            PRODUCTION = PRD2.SCEEBASE
  ARCHIVE 5686-067-03-18D
  ARCHIVE 06718D
  RESOLVES ¢LEVSE.BCOB.1.1.0 LE/VSE COBOL BASE RELEASE 1.0¢
  COMPRISES 5686-067-03  -
     PHASES   = (IGZ*,ILB*,CEE*,$$B*) -
     MODULES  = (IGZ*,ILB*,CEE*) -
     MACROS   = ($$$CO18D,IGZ*) TYPE=Z
  RESIDENCE PRODUCT = (06718D)       -
            PRODUCTION = PRD2.SCEEBASE
```

Figure 2. MSHP Statements to Archive the LE Components for COBOL/VSE Installation

```
// EXEC MSHP,SIZE=900K
  ARCHIVE 5686-067-00-18A
  ARCHIVE 06718A
  RESOLVES ¢LEVSE.BLE..1.1.0 LE/VSE BASE RELEASE 1.0¢
  COMPRISES 5686-067-00  -
     PHASES   = (CEE*) -
     MODULES  = (CEE*) -
     MACROS   = ($$$CO18A,CEE*) TYPE=Z
  RESIDENCE PRODUCT = (06718A)       -
            PRODUCTION = PRD2.SCEEBASE
  ARCHIVE 5686-067-07-18H
  ARCHIVE 06718H
  RESOLVES ¢LEVSE.BPLI.1.1.0 LE/VSE PLI BASE RELEASE 1.0¢
  COMPRISES 5686-067-07  -
     PHASES   = (IBM*, CEE*) -
     MODULES  = (IBM*, CEE*) -
     MACROS   = ($$$CO18H,CEE*,IBM*) TYPE=Z
  RESIDENCE PRODUCT = (06718H)       -
            PRODUCTION = PRD2.SCEEBASE
```

Figure 3. MSHP Statements to Archive the LE Components for PL/I VSE Installation

Chapter 5. Year 2000 Considerations

5.1 Overview

Put simply, the basic Year 2000 problem is 2-digit year fields in date representations.

If all your applications currently identify the year by using the full four digit number, for example, 1996, and use that value for all date arithmetic, then you probably won't encounter a Year 2000 problem.

However, if your applications use only the last two digits of the year, for example, 96, then the switch from 1999 to 2000 is going to cause problems in any date arithmetic calculations.

5.1.1 Long-term Solution

Your only safe, long-term solution is to rebuild your databases and data files, and to change your applications, to use 4-digit year dates. However, LE/VSE and the LE- conforming languages do provide other ways to solve the problem.

5.1.2 Short-term Solution

As a short-term solution to the Year 2000 problem, use the century window feature of LE/VSE. You do this by using LE/VSE callable services. This gives you the ability to interpret 2-digit year dates as 4-digit year dates. You can change the minimum number of programs without changing your databases. But this is only an interim solution.

Remember that a 2-digit year can only be unique in a given 100-year period. If your application calculates a date in the past (for example, if you need to know how old someone is), or a date in the future (for example, to check if a due date has passed), then the century window can solve your problem. But it will not solve the problem if the time period you require is greater than 100 years.

LE/VSE Date/time Callable Services

The LE/VSE callable services that you can use for the century window feature are:

CEEDATE convert Lilian date to character format

CEEDATM convert seconds to character timestamp

CEEDAYS convert date to Lilian format

CEEDYWK calculate day of week from Lilian date

CEEGMT get current Greenwich Mean Time

CEEGMTO get offset from Greenwich Mean Time to local time

CEEISEC convert integers to seconds

CEELOCT get current local time

CEEQCEN query century window

CEESCEN set century window

CEESECI convert seconds to integers

CEESECS convert timestamp to number of seconds

CEECBLDY convert date to COBOL integer format (LE/VSE Release 4 only.)

Definitions of reference dates

Lilian Date
The Lilian date is the number of days since 14 October 1582

Gregorian Date
The Gregorian date is the number of days since 31 December 1600

Showa - Japanese Era
The Showa era started on 25 December 1926, and has valid year dates of 01-64. For example, Showa 63 equals 1988.

Heisei - Japanese Era
The Heisei era started on 8 January 1989, and has valid year dates of 01-999.

MinKow - Republic of China (ROC) Era
The MinKow era started on 1 January 1912, and has valid year dates of 01-999. For example, MinKow 77 equals 1988.

You can find more information about the format of character strings used by the date and time callable services of LE/VSE, in *IBM Language Environment for VSE/ESA Programming Reference*, SC33-6685. Appendix B of that book gives information to help you use the date and time callable services and includes tables for picture term and national language era usage.

5.2 COBOL - Problems and How COBOL/VSE and LE/VSE Can Solve Them

5.2.1 Date Formats Used by COBOL Compilers

DOS/VS COBOL Uses only 2-digit year dates and this will not be changed.

VS COBOL II Uses only 2-digit year dates and this will not be changed.

COBOL/VSE Uses 4-digit year dates in the object decks produced by the compiler and in ADATA records.

COBOL/VSE compiler listings display 2-digit year dates, but you can get 4-digit year dates if you apply APAR PN84766.

5.2.2 Long-term Solution

Your only long-term solution is to rebuild your data bases and files, and change your applications to use 4-digit year dates.

COBOL/VSE provides full Year 2000 support. It provides ANSI COBOL Standard Intrinsic Functions which, together with LE/VSE callable services, give full date manipulation capability with 4-digit year support.

Therefore, rewrite your applications using COBOL/VSE and LE/VSE.

Note: If your applications perform any arithmetic on dates, do not mix date values from the COBOL/VSE intrinsic functions with those from LE/VSE callable

services. COBOL/VSE intrinsic functions use the Gregorian date as the base and LE/VSE callable services use the Lilian date. Use the CEECBLDY callable service for conversion between LE/VSE date routines and COBOL/VSE ANSI intrinsic function date routines.

COBOL/VSE Intrinsic Functions

The COBOL/VSE intrinsic functions that provide full 4-digit year support are:

CURRENT-DATE
DATE-OF-INTEGER
DAY-OF-INTEGER
INTEGER-OF-DATE
INTEGER-OF-DAY
WHEN-COMPILED

An example of obtaining the 4-digit year using COBOL intrinsic functions:

DATE-OF-INTEGER gives *YYYYMMDD*

DAY-OF-INTEGER gives *YYYYDDD*

Note: Do not confuse the COBOL intrinsic function, **WHEN-COMPILED**, with the special register, **WHEN-COMPILED**. The intrinsic function is unique to COBOL/VSE and is designed to handle 4-digit years. The special register will return only a 2-digit year, even in COBOL/VSE.

5.2.3 Short-term Solution

As a short-term solution use COBOL/VSE with the century window feature of LE/VSE to interpret your 2-digit year dates as 4-digit year dates.

5.2.4 An Example of the Century Window Using COBOL/VSE and LE/VSE

LE/VSE callable services use Lilian dates for date representation. The base used by LE/VSE callable services dates is 14 October 1582. Thus, in a COBOL program, to convert a string representing a date into a Lilian format, use the CEEDAYS callable service.

CALL ″CEEDAYS″ USING *input-date* PICSTR *lilian-days* FC.

This returns an integer representation of the date as the number of days since 14 October 1582, in *lilian-days*.

Then convert this number of days back to a character string using the CEEDATE callable service.

CALL ″CEEDATE″ USING *lilian-days* PICSTR *output-char-date* FC.

This returns a character representation of the date with a **4-digit** value for the year, in *output-char-date*.

The example in Figure 4 on page 36 shows the use of LE/VSE callable services for the century window feature in a COBOL/VSE program.

In this example the century window is set:

- to the current year to demonstrate date arithmetic that is used to calculate how many years are left before a particular ′due-date′ occurs

- to 100 years ago to demonstrate the ability to calculate the age of ′customers′, given their birth date.

Clearly, the century window can be set to any year, within a hundred-year range, that is applicable for the date arithmetic required for your application.

```
ID           DIVISION.
PROGRAM-ID. COBEXMP.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT RUNDATA ASSIGN TO INFILE
           FILE STATUS IS RUNDATA-FS.
DATA DIVISION.
FILE SECTION.
FD RUNDATA.
01 WORK-RECORD.
   03 CUST-NAME.
      05 FIRST-NAME PIC X(10).
      05 LAST-NAME  PIC X(15).
   03 BIRTH-DATE    PIC X(8).                          1
   03 ACCOUNT-NUM   PIC 9(8).
   03 CURRENT-AGE   PIC 999.
   03 DUE-DATE      PIC X(8).                          2
   03 BENNY-FACTOR  PIC 9(3)  COMP-3.
   03 ADJUSTMENT    PIC 9(3)  COMP-3.
   03 PAYOUT        PIC S9(7) COMP-3.
   03              PIC X(20).

WORKING-STORAGE SECTION.
01 RUNDATA-FS     PIC 99.
77 EOF-IND        PIC X.
  88 EOF           VALUE †Y†.
  88 NOT-EOF       VALUE †N†.

PROCEDURE DIVISION.

OPENIT. OPEN INPUT RUNDATA.
        IF RUNDATA-FS NOT EQUAL TO 0
          DISPLAY †** ERROR ** †
                  †CAN¢T OPEN RUNDATA FILE **†
        ELSE
          SET NOT-EOF TO TRUE
          PERFORM LOOP
          CLOSE RUNDATA
        END-IF

        STOP RUN.

LOOP. PERFORM UNTIL EOF
        READ RUNDATA
             AT END SET EOF TO TRUE
             NOT AT END
                 CALL †COBSUB† USING WORK-RECORD
        END-READ

      END-PERFORM.

END PROGRAM COBEXMP.
```

Figure 4 (Part 1 of 4). COBOL Example COBEXMP

```
 ID DIVISION.
  PROGRAM-ID. COBSUB.
 DATA DIVISION.
  WORKING-STORAGE SECTION.

  01 CURR-LILIAN PIC S9(9) COMP.
  01 CURR-SECONDS COMP-2.
  01 CURR-DATE.                                          3
    05 CURR-YEAR     PIC 9(4).
    05 CURR-MONTH    PIC 99.
    05 CURR-DAY      PIC 99.
    05 CURR-TIME     PIC X(9).
  77 YEARS-LEFT      PIC 9(4).

************************************************
* Working Storage & parameters for CEEDATE     *
************************************************
  01 4-DIGIT-DATE    PIC X(80).                         4
  01 4-DIGIT-REDEFINED REDEFINES 4-DIGIT-DATE.
    05 YYYY PIC 9(4).
    05 MM   PIC 9(2).
    05 DD   PIC 9(2).
    05 FILLER PIC X(72).

  01 PIC4STR.
    05 PIC4STR-LENGTH PIC 9(2) COMP VALUE 8.
    05 PIC4STR-STRING PIC X(8) VALUE †YYYYMMDD†.

************************************************
*        Working storage & Parameters for      *
*               CEEDAYS and CEESCEN            *
************************************************

  01 WORK-DATE.                                          5
    05 WORK-DATE-LEN PIC 9(2) VALUE 8.
    05 WORK-DATE-MM  PIC X(2) .
    05 WORK-DATE-F1  PIC X VALUE †/†.
    05 WORK-DATE-DD  PIC X(2).
    05 WORK-DATE-F2  PIC X VALUE †/†.
    05 WORK-DATE-YY  PIC X(2).

  01 PICSTR.
    05 PICSTR-LENGTH PIC 9(2) COMP VALUE 8.
    05 PICSTR-STRING PIC X(50) VALUE †MM/DD/YY†.
  77 LILIAN          PIC 9(9) COMP.
  77 START-CW        PIC 9(9) COMP.
  77 FC              PIC X(12).
```

Figure 4 (Part 2 of 4). COBOL Example COBEXMP

```
  LINKAGE SECTION.
***********************************************
* Linkage Section description of the record    *
* that is read by COBEXMP, and shared by COBSUB.*
* Note that BIRTH-DATE and DUE-DATE            *
* are expanded to show formats.                *
***********************************************
  01 WORK-RECORD.
     03 CUST-NAME.
       05 FIRST-NAME   PIC X(10).
       05 LAST-NAME    PIC X(15).
     03 BIRTH-DATE.
       05 BMONTH       PIC 99.
       05             PIC X.
       05 BDAY         PIC 99.
       05             PIC X.
       05 BYEAR        PIC 99.
     03 ACCOUNT-NUM    PIC 9(8).
     03 CURRENT-AGE    PIC 999.
     03 DUE-DATE.
       05 DMONTH       PIC 99.
       05             PIC X.
       05 DDAY         PIC 99.
       05             PIC X.
       05 DYEAR        PIC 99.
     03 BENNY-FACTOR   PIC 9(3)  COMP-3.
     03 ADJUSTMENT     PIC 9(3)  COMP-3.
     03 PAYOUT         PIC S9(7) COMP-3.
     03                PIC X(20).

 PROCEDURE DIVISION USING WORK-RECORD.

***********************************************
* get todays date with 4-digit year.           *
***********************************************
     CALL †CEELOCT† USING CURR-LILIAN , CURR-SECONDS ,   6
          CURR-DATE , FC .

***********************************************
* set century window to start with the current   *
* year (current system date).                  *
***********************************************
     MOVE 0 TO START-CW.
     CALL †CEESCEN† USING START-CW FC.                   7

***********************************************
* convert 2-digit due date year from input file  *
* into integer value using LE/VSE service to get *
* COBOL Lilian date                            *
***********************************************
     MOVE DMONTH TO WORK-DATE-MM.                        8
     MOVE DDAY TO WORK-DATE-DD.                          8
     MOVE DYEAR TO WORK-DATE-YY.                         8
     CALL †CEEDAYS† USING WORK-DATE                      9
                          PICSTR LILIAN FC.
```

Figure 4 (Part 3 of 4). COBOL Example COBEXMP

```
      ************************************************
      * convert COBOL Lilian date into YYYYMMDD format *
      ************************************************
           CALL †CEEDATE† USING LILIAN , PIC4STR ,            10
                4-DIGIT-DATE , FC.

      ************************************************
      * find out how many years until due date       *
      ************************************************
           COMPUTE YEARS-LEFT = YYYY - CURR-YEAR.             11

      ************************************************
      * set century window to start 100 years ago    *
      ************************************************
           MOVE 100 TO START-CW.
           CALL †CEESCEN† USING START-CW FC.                  12

      ************************************************
      * convert 2-digit birthdate year from input file *
      * into integer value to get COBOL Lilian date  *
      ************************************************
           MOVE BMONTH TO WORK-DATE-MM.                       13
           MOVE BDAY TO WORK-DATE-DD.                         13
           MOVE BYEAR TO WORK-DATE-YY.                        13
           CALL †CEEDAYS† USING WORK-DATE                     14
                              PICSTR LILIAN FC.

      ************************************************
      * convert COBOL Lilian date into YYYYMMDD format *
      ************************************************
           CALL †CEEDATE† USING LILIAN , PIC4STR ,            15
                4-DIGIT-DATE , FC.

      ************************************************
      * find the age † even after 1999!              *
      ************************************************
           COMPUTE CURRENT-AGE = CURR-YEAR - YYYY.            15

           GOBACK.

       END PROGRAM COBSUB.
```

Figure 4 (Part 4 of 4). COBOL Example COBEXMP

1 2 The code in the COBOL program, COBEXMP, shows that the work records contain both a BIRTH-DATE field for the customer and a DUE-DATE field for some provided service.

The main routine of the program simply opens the input file and then reads each record in turn, calling the subroutine, COBSUB, to process the records.

3 Working storage area to hold the returned values from the callable service CEELOCT 6 to obtain the current local date and time.

4 Working storage area to hold the date, including a 4-digit value for the year, returned by CEEDATE 10 15

5 Working storage area to provide a length-prefix field to the date character string, for the calls to CEEDAYS 9 14

6 CEELOCT is called to obtain the current local date. The values returned are:

CURR-LILIAN A 32-bit integer representing the current local date in Lilian format.

CURR-SECONDS A 64-bit double-floating point number representing the current local date and time as the number of seconds since 00:00:00 on 14 October 1582.

CURR-DATE A 17-byte fixed-length character string in the form YYYYMMDDHHMISS999 representing local year, month, day, hour, minute, second, and millisecond.

(The examples quoted here were run with a current date of 17 April 1996, that is CURR-DATE = 19960417142127770.)

7 CEESCEN is called to set the century window to start with the current year.

8 The DUE-DATE values are moved to the WORK-DATE field to provide the required parameter format.

9 CEEDAYS is called to convert the DUE-DATE value from the input record to an integer value Lilian date. The input date field to CEEDAYS requires a halfword length-prefixed character string indicating its format.

10 CEEDATE is called to convert the DUE-DATE value into a character string representation including a 4-digit year value.

For example, a record containing the DUE-DATE value of

- DMONTH = 01
- DDAY = 30
- DYEAR = 03

produces a date value of **20030130**

11 After this statement, the YEARS-LEFT field contains the value **7**.

12 CEESCEN is called again to set the century window to start 100 years before the current system date.

13 The BIRTH-DATE values are moved to the WORK-DATE field.

14 CEEDAYS is called again to get the Lilian date integer value.

15 CEEDATE is called again to convert the date to a 4-digit year date.

For example, a record containing the BIRTH-DATE value of

- BMONTH = 04
- BDAY = 05
- BYEAR = 09

produces a date value of **19090405**.

16 After this statement, the CURRENT-AGE field contains the value **87**.

This example is a simple, but short-term, approach to a solution to the Year 2000 problem using the century window feature of LE/VSE with COBOL/VSE.

5.2.5 CEECBLDY - LE/VSE Callable Service to Convert a Date to COBOL Integer Format

The CEECBLDY callable service is provided with LE/VSE Release 4. It is primarily designed for COBOL applications, to convert a date into a format compatible with ANSI 85 Intrinsic Functions. However, you can also use it in a PL/I or C program to convert a date to a format suitable for use in a COBOL program called subsequently from a PL/I or C program.

COBOL/VSE intrinsic functions use Gregorian dates for date representation. The base used is 31 December 1600.

Use CEECBLDY to access the century window and to perform date calculations with ANSI intrinsic functions. For example, use CEECBLDY to convert a string representing a date into a COBOL integer format which is compatible with ANSI intrinsic functions. This integer is the number of days since 31 December 1600.

Call CEECBLDY only from programs that will use the returned value as the input to a COBOL intrinsic function. Do not use this returned value with other LE/VSE callable services.

5.2.6 An Example of CEECBLDY Callable Service

Figure 5 is a sample COBOL program showing the use of CEECBLDY to convert a date to COBOL integer format.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDAYS.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CHRDATE.
    05  CHRDATE-LENGTH      PIC  9(2) BINARY.
    05  CHRDATE-STRING      PIC X(50).
01  PICSTR.
    05  PICSTR-LENGTH       PIC S9(4) BINARY.
    05  PICSTR-STRING       PIC X(50).
01  INTEGER                 PIC S9(9) BINARY.
01  NEWDATE                 PIC 9(8).
01  FC                      PIC X(12).

PROCEDURE DIVISION.
*************************************************
** Specify input date and length               **
*************************************************
    MOVE †1 January 00† to CHRDATE-STRING.            1
    MOVE 25 TO CHRDATE-LENGTH.

*************************************************
** Specify a picture string that describes     **
** input date, and the picture string†s length.**
*************************************************
    MOVE †ZD Mmmmmmmmmmmmmz YY†
                        TO PICSTR-STRING.             2
    MOVE 23 TO PICSTR-LENGTH.
```

Figure 5 (Part 1 of 2). An Example of the Use of CEECBLDY

```
*************************************************
** Call CEECBLDY to convert input date to a    **
** COBOL Integer date                          **
*************************************************
     CALL †CEECBLDY† USING CHRDATE, PICSTR,              3
                           INTEGER, FC.


*************************************************
** If CEECBLDY runs successfully, then compute **
** the date of the 90th day after the          **
** input date using Intrinsic Functions        **
*************************************************
     IF (FC = LOW-VALUE) THEN
         COMPUTE INTEGER = INTEGER + 90
         COMPUTE NEWDATE = FUNCTION                      4
                       DATE-OF-INTEGER (INTEGER)
         DISPLAY NEWDATE † is Integer day: † INTEGER     5
     ELSE
         CONTINUE
     END-IF.

     GOBACK.
```

Figure 5 (Part 2 of 2). An Example of the Use of CEECBLDY

1 2 The MOVE statement 1 sets up the character string representing the date in the format specified in the PICSTR field 2 .

3 CEECBLDY is called to obtain the COBOL integer format date in the field INTEGER. This date represents the number of days since 31 December 1600.

4 Convert the COBOL integer date returned by CEECBLDY to the standard date form YYYYMMDD (Year, Month, Day) in the field NEWDATE.

5 The result of the DISPLAY statement is

20000331 is Integer day: 000145822

5.3 PL/I - Problems and How PL/I VSE and LE/VSE Can Solve Them

5.3.1 Date Formats Used by PL/I Compilers

DOS PL/I Uses only 2-digit year dates and this will not be changed.

PL/I VSE Uses 2-digit year dates in the object decks produced by the compiler.

PL/I VSE compiler listings display 2-digit year dates, but you can get 4-digit year dates if you apply APAR PN87150.

5.3.2 Long-term Solution

Your only long-term solution is to rebuild your data bases and files, and change your applications to use 4-digit year dates.

PL/I VSE, together with the callable services of LE/VSE, provides full Year 2000 support.

Therefore, rewrite your applications using PL/I VSE and LE/VSE.

5.3.3 Short-term Solution

As a short-term solution use PL/I VSE with the century window feature of LE/VSE to interpret your 2-digit year dates as 4-digit year dates.

5.3.4 An Example of the Century Window Using PL/I VSE and LE/VSE

LE/VSE callable services use Lilian dates for date representation. The base used by LE/VSE callable services dates is 14 October 1582. Thus, in a PL/I program, to convert a string representing a date into a Lilian format, use the CEEDAYS callable service.

```
CALL CEEDAYS (input-date, PICSTR, lilian-days, FC);
```

This returns an integer representation of the date as the number of days since 14 October 1582, in *lilian-days*.

Then convert this number of days back to a character string using the CEEDATE callable service.

```
CALL CEEDATE (lilian-days, PICSTR, output-char-date, FC);
```

This returns a character representation of the date with a **4-digit** value for the year, in *output-char-date*.

The example in Figure 6 on page 44 shows the use of LE/VSE callable services for the century window feature in a PL/I VSE program.

In this example the century window is set:

- to the current year to demonstrate date arithmetic that is used to calculate how many years are left before a particular 'due-date' occurs
- to 100 years ago to demonstrate the ability to calculate the age of 'customers', given their birth date.

Clearly, the century window can be set to any year, within a hundred-year range, that is applicable for the date arithmetic required for your application.

```
*PROCESS MACRO;
 PLIEXMP: PROC OPTIONS(MAIN);
        %INCLUDE  CEEIBMAW;
        %INCLUDE  CEEIBMCT;
        DCL EOF_IND  BIT(1) INIT(¢0¢B); /* End of File indicator */
        DCL RUNDATA FILE RECORD INPUT SEQUENTIAL  /* File        */
            ENV(RECSIZE(80));                     /* declaration */
        DCL 1 IN,                      /* Input record          */
              2 IN_CUST_NAME,                     /*  structure  */
                3 IN_FIRST_NAME CHAR(10),
                3 IN_LAST_NAME  CHAR(15),
              2 IN_BIRTH_DATE,                                  1
                3 IN_BMONTH     CHAR(2),
                3 IN_BF1        CHAR(1),
                3 IN_BDAY       CHAR(2),
                3 IN_BF2        CHAR(1),
                3 IN_BYEAR      CHAR(2),
              2 IN_ACCOUNT_NUM  CHAR(8),
              2 IN_CURRENT_AGE  CHAR(3),
              2 IN_DUE_DATE,                                    2
                3 IN_DMONTH     CHAR(2),
                3 IN_DF1        CHAR(1),
                3 IN_DDAY       CHAR(2),
                3 IN_DF2        CHAR(1),
                3 IN_DYEAR      CHAR(2),
              2 IN_BENNY_FACTOR CHAR(3),
              2 IN_ADJUSTMENT   CHAR(3),
              2 IN_PAYOUT       CHAR(7),
              2 IN_WRF          CHAR(15);
       ON ENDFILE(RUNDATA) EOF_IND = ¢1¢B;

       OPEN FILE(RUNDATA);

       READ FILE(RUNDATA) INTO(IN);

       DO WHILE (EOF_IND = 0);
         CALL PLISUB(IN);
         READ FILE(RUNDATA) INTO(IN);
         END;
     PLISUB: PROC(WORK_RECORD);
       DCL LILIAN          INT4;                             3
       DCL SECONDS         FLOAT8;                           3
       DCL 1 GREG,                                           3
             2 CURR_YEAR   CHAR(4),
             2 CURR_MONTH  CHAR(2),
             2 CURR_DAY    CHAR(2),
             2 CURR_TIME   CHAR(9),
           GREGORN         CHAR(17) DEF GREG;
       DCL CHRDATE         VSTRING;
       DCL PICSTR          VSTRING;
       DCL PIC4STR         VSTRING;
       DCL LIL2            INT4;
       DCL STARTCW         INT4;
       DCL YYYY            CHAR(4);
       DCL FOUR_DIGIT_DATE CHAR(80);                         4
       DCL YEARS_LEFT      FIXED DECIMAL(15,0);
       DCL AGE             FIXED DECIMAL(15,0);
       DCL 01 FC           FEEDBACK;
```

Figure 6 (Part 1 of 3). PL/I Example PLIEXMP

```
DCL 1 WORK_RECORD,                                      5
      2 CUST_NAME,
        3 FIRST_NAME CHAR(10),
        3 LAST_NAME  CHAR(15),
      2 BIRTH_DATE,
        3 BMONTH     CHAR(2),
        3 BF1        CHAR(1),
        3 BDAY       CHAR(2),
        3 BF2        CHAR(1),
        3 BYEAR      CHAR(2),
      2 ACCOUNT_NUM  CHAR(8),
      2 CURRENT_AGE  CHAR(3),
      2 DUE_DATE,
        3 DMONTH     CHAR(2),
        3 DF1        CHAR(1),
        3 DDAY       CHAR(2),
        3 DF2        CHAR(1),
        3 DYEAR      CHAR(2),
      2 BENNY_FACTOR CHAR(3),
      2 ADJUSTMENT   CHAR(3),
      2 PAYOUT       CHAR(7),
      2 WRF          CHAR(15);
/*************************************************/
/* Call CEELOCT to return local time in 3 formats */
/*************************************************/

 CALL CEELOCT ( LILIAN, SECONDS, GREGORN, FC );         6

/*************************************************/
/* Set century window to start with the current   */
/* year current system date)                      */
/*************************************************/

 STARTCW = 0;
 CALL CEESCEN ( STARTCW, FC );                          7
/*************************************************/
/* Convert 2-digit due date year from input file  */
/* into integer value using LE/VSE service to get */
/* PL/I Lilian date                               */
/*************************************************/

 CHRDATE = DMONTH || ¢/¢ || DDAY || ¢/¢ || DYEAR;       8
 PICSTR = ¢ZM/ZD/YY¢;
 CALL CEEDAYS ( CHRDATE, PICSTR, LILIAN, FC);           9

/*************************************************/
/* Convert PL/I Lilian date into YYYYMMDD format  */
/*************************************************/

 PIC4STR = ¢YYYYMMDD¢;
 CALL CEEDATE ( LILIAN, PIC4STR, FOUR_DIGIT_DATE, FC);  10

/*************************************************/
/* Calculate number of years until due date       */
/*************************************************/

 YYYY = SUBSTR(FOUR_DIGIT_DATE, 1, 4);
 YEARS_LEFT = YYYY - CURR_YEAR;                         11
```

Figure 6 (Part 2 of 3). PL/I Example PLIEXMP

```
/************************************************/
/* Set century window to start 100 years before   */
/* the current system date                        */
/************************************************/
 STARTCW = 100;
 CALL CEESCEN (STARTCW, FC);                              12

/************************************************/
/* Convert 2-digit birthdate year from input file */
/* into integer value using LE/VSE service to get */
/* PL/I Lilian date                               */
/************************************************/
 CHRDATE = BMONTH || ¢/¢ || BDAY || ¢/¢ || BYEAR;         13
 PICSTR = ¢ZM/ZD/YY¢;
 CALL CEEDAYS ( CHRDATE, PICSTR, LILIAN, FC);             14

/************************************************/
/* Convert PL/I Lilian date into YYYYMMDD format  */
/************************************************/
 CALL CEEDATE ( LILIAN, PIC4STR, FOUR_DIGIT_DATE, FC);    15

/************************************************/
/* Find the age - even after 1999!                */
/************************************************/
 YYYY = SUBSTR(FOUR_DIGIT_DATE, 1, 4);
 AGE = CURR_YEAR - YYYY;                                  16

 END PLISUB;
 END PLIEXP;
```

Figure 6 (Part 3 of 3). PL/I Example PLIEXMP

1 2 The code in the PL/I program, PLIEXMP, shows that the work records contain both an IN_BIRTH_DATE field for the customer and a IN_DUE_DATE field for some provided service.

The main routine of the program simply opens the input file and then reads each record in turn, calling PLISUB to process the records.

3 Work areas to hold the returned values from the callable service CEELOCT 6 to obtain the current local date and time.

4 Work area to hold the date, including a 4-digit value for the year, returned by CEEDATE 10 15

5 Definition of the structure of the input record passed to the sub-routine, PLISUB.

6 CEELOCT is called to obtain the current local date. The values returned are:

LILIAN A 32-bit integer representing the current date in Lilian format.

SECONDS A 64-bit double-floating point number representing the current local date and time as the number of seconds since 00:00:00 on 14 October 1582.

GREGORN A 17-byte fixed-length character string in the form YYYYMMDDHHMISS99 (the Gregorian format) representing local year, month, day, hour, minute,second, and millisecond.

(The examples quoted here were run with a current date of 17 April 1996, that is GREGORN = 19960417142117770.)

7 CEESCEN is called to set the century window to start with the current year.

8 The DUE_DATE values are assigned to the CHRDATE field to provide the required parameter format as defined in the PICSTR field.

9 CEEDAYS is called to convert the DUE_DATE value from the input record to an integer value Lilian date. The input date field to CEEDAYS requires a halfword length-prefixed character string indicating its format.

10 CEEDATE is to convert the DUE_DATE value into a character string representation including a 4-digit year value.

For example, a record containing the DUE_DATE value of

- DMONTH = 01
- DDAY = 30
- DYEAR = 03

produces a date value of **20030130**.

11 After this statement the YEARS_LEFT field contains the value **7**.

12 CEESCEN is called again to set the century window to start 100 years before the current system date.

13 The BIRTH_DATE values are assigned to the CHRDATE field.

14 CEEDAYS is called again to get the Lilian date integer value.

15 CEEDATE is called again to convert the date to a 4-digit year date.

For example, a record containing the BIRTH_DATE value of

- BMONTH = 04
- BDAY = 05
- BYEAR = 09

produces a date value of **19090405**.

16 After this statement the AGE field contains the value **87**.

This example is a simple, but short-term, approach to the Year 2000 problem using the century window feature of LE/VSE with PL/I VSE.

5.3.5 The DATE and DATETIME Built-In Functions

PL/I provides two built-in functions to return the current system date and time:

DATE returns a character string of length 6, in the format **yymmdd**.

DATETIME returns a character string of length 17, in the format **yyyymmddhhmmssttt**.

DATE returns the date with only 2-digit years. Use the **DATETIME** built-in function to obtain a date with a 4-digit year.

5.4 C - Problems and How C/VSE and LE/VSE Can Solve Them

5.4.1 Date Formats Used by C Compilers

C/370 Version 2 Release 1

Uses only 2-digit year dates and this will not be changed.

C/VSE

Uses 2-digit year dates in the object decks produced by the compiler.

C/VSE compiler listings display 4-digit year dates.

5.4.2 Long-term Solution

Your only long-term solution is to rebuild your data bases and files, and change your applications to use 4-digit year dates.

C/VSE, together with the callable services of LE/VSE, provides full Year 2000 support.

Therefore, rewrite your applications using C/VSE.

5.4.3 Short-term Solution

As a short-term solution use C/VSE with the century window feature of LE/VSE to interpret your 2-digit year dates as 4-digit year dates.

5.4.4 An Example of the Century Window Using C/VSE and LE/VSE

LE/VSE callable services use Lilian dates for date representation. The base used by LE/VSE callable services dates is 14 October 1582. Thus, in a C/VSE program, to convert a string representing a date into a Lilian format, use the CEEDAYS callable service.

```
CEEDAYS(&date, &date_pic, &lil_date, &fc);
```

This returns an integer representation of the date as the number of days since 14 October 1582, in *lil_date*.

Then convert this number of days back to a character string using the CEEDATE callable service.

```
CEEDATE(&lil_date, &date_pic, date_out, &fc);
```

This returns a character representation of the date with a **4-digit** value for the year, in *date_out*.

The example in Figure 7 on page 49 shows code that uses the LE/VSE callable services for the century window feature in a C/VSE program.

In this example the century window is set:

- to the current year to demonstrate date arithmetic that is used to calculate how many years are left before a particular 'due-date' occurs
- to 100 years ago to demonstrate the ability to calculate the age of 'customers', given their birth date.

Clearly, the century window can be set to any year, within a hundred-year range, that is applicable for the date arithmetic required for your application.

```
  /* ********************************************** */
  /* EDCEXMP                                        */
  /* C/VSE Routine to demonstrate the LE/VSE Century    */
  /*                  Window Feature                */
  /* ********************************************** */

   #include <stdlib.h>
   #include <string.h>
   #include <leawi.h>
   #include <ceeedcct.h>

int main(void) {

  _FEEDBACK fc;
  _INT4    century_start;

  _INT4    lil_today;                                    1
  _FLOAT8  lil_secs;                                     1
  _CHAR17  gregorian_date;                               1

  _VSTRING due_date,due_date_pic;
  _INT4    lil_due;
  _VSTRING due4_date,due4_date_pic;
  _CHAR80  date_out1;
  _CHAR80  digit4_due_date;                              2

  _INT4    curr4_year_int;
  _INT4    due4_year_int;
  char     curr4_year[5];
  char     due4_year[5];
  _INT4    years_left;

  _VSTRING birth_date,birth_date_pic;
  _INT4    lil_birth;
  _VSTRING birth4_date,birth4_date_pic;
  _CHAR80  date_out2;
  _CHAR80  digit4_birth_date;                            2

  _INT4    birth4_year_int;
  char     birth4_year[5];
  _INT4    curr_age;
  /* ********************************************** */
  /* Call CEELOCT to get todays date with a 4-digit year */
  /* ********************************************** */

  CEELOCT(&lil_today,&lil_secs,gregorian_date,&fc);      3
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf(†CEELOCT failed with message number %d\n†,
            fc.tok_msgno);
     exit(2999);
  }
```

Figure 7 (Part 1 of 3). C/VSE Example EDCEXMP

```
/* ************************************************ */
/* Call CEESCEN to set the century window to start     */
/* with the current year (current system date)         */
/* ************************************************ */

century_start = 0;
CEESCEN(&century_start,&fc);                                    4
if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
   printf(†CEESCEN failed with message number %d\n†,
          fc.tok_msgno);
   exit(2999);
}
/* ************************************************ */
/* Call CEEDAYS to convert a 2-digit year Due Date     */
/* to Lilian format                                    */
/* ************************************************ */

 strcpy(due_date.string,†01/30/03†);                            5
 due_date.length = strlen(due_date.string);
 strcpy(due_date_pic.string,†MM/DD/YY†);
 due_date_pic.length = strlen(due_date_pic.string);

 CEEDAYS(&due_date,&due_date_pic,&lil_due,&fc);                 6
 if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf(†CEEDAYS failed with message number %d\n†,
           fc.tok_msgno);
    exit(2999);
 }
/* ************************************************ */
/* Call CEEDATE to convert the Lilian date to          */
/* MM/DD/YYYY format                                   */
/* ************************************************ */

 strcpy(due4_date_pic.string,†MM/DD/YYYY†);
 due4_date_pic.length = strlen(due4_date_pic.string);

 CEEDATE(&lil_due,&due4_date_pic,digit4_due_date,&fc);          7

 if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
    printf(†CEEDATE failed with message number %d\n†,
            fc.tok_msgno);
    exit(2999);
 }
   printf(†The due date is %.80s\n†,digit4_due_date);
/* ***************************************** */
/* Calculate years left to due date          */
/* ***************************************** */

memset (curr4_year,¢\0¢,5);                                     8
memset (due4_year,¢\0¢,5);
strncpy (curr4_year,gregorian_date,4);
strncpy (due4_year,digit4_due_date+6,4);

curr4_year_int = atoi(curr4_year);
due4_year_int = atoi(due4_year);
years_left = due4_year_int - curr4_year_int;                    9
printf(†Number of years to due date is is %d\n†,years_left);
```

Figure 7 (Part 2 of 3). C/VSE Example EDCEXMP

```
  /* ************************************************ */
  /* Call CEESCEN to set the century window to start     */
  /* 100 years ago                                       */
  /* ************************************************ */

  century_start = 100;

  CEESCEN(&century_start,&fc);                                      10
  if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
     printf(†CEESCEN failed with message number %d\n†,
            fc.tok_msgno);
     exit(2999);
  }

  /* ************************************************ */
  /* Call CEEDAYS to convert a 2-digit year Birth Date   */
  /* to Lilian format                                    */
  /* ************************************************ */

   strcpy(birth_date.string,†04/05/09†);                            11
   birth_date.length = strlen(birth_date.string);
   strcpy(birth_date_pic.string,†MM/DD/YY†);
   birth_date_pic.length = strlen(birth_date_pic.string);

   CEEDAYS(&birth_date,&birth_date_pic,&lil_birth,&fc);             12
   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf(†CEEDAYS failed with message number %d\n†,
             fc.tok_msgno);
      exit(2999);
   }
  /* ************************************************ */
  /* Call CEEDATE to convert the Lilian date to          */
  /* MM/DD/YYYY format                                   */
  /* ************************************************ */

   strcpy(birth4_date_pic.string,†MM/DD/YYYY†);
   birth4_date_pic.length = strlen(birth4_date_pic.string);

   CEEDATE(&lil_birth,&birth4_date_pic,digit4_birth_date,&fc);      13

   if ( _FBCHECK ( fc , CEE000 ) != 0 ) {
      printf(†CEEDATE failed with message number %d\n†,
             fc.tok_msgno);
      exit(2999);
   }
     printf(†The birth date is %.80s\n†,digit4_birth_date);
  /* ****************************************** */
  /* Calculate age in current year                */
  /* ****************************************** */

  memset (curr4_year,¢\0¢,5);                                       14
  memset (birth4_year,¢\0¢,5);
  strncpy (curr4_year,gregorian_date,4);
  strncpy (birth4_year,digit4_birth_date+6,4);

  curr4_year_int = atoi(curr4_year);
  birth4_year_int = atoi(birth4_year);
  curr_age = curr4_year_int - birth4_year_int;                      15
  printf(†Age is %d\n†,curr_age);

}
```

Figure 7 (Part 3 of 3). C/VSE Example EDCEXMP

The code in the C program, EDCEXMP, shows that the work records contain both a birth date field for the customer and a due date field for some provided service.

1 Areas defined to hold the returned values from the callable service CEELOCT 3 to obtain the current local date and time.

2 Area defined to hold the date, including a 4-digit value for the year, returned by CEEDATE 7 13

3 CEELOCT is called to obtain the current local date. The values returned are:

lil_today A 32-bit integer representing the current date in Lilian format.

lil_secs A 64-bit double-floating point number representing the current local date and time as the number of seconds since 00:00:00 on 14 October 1582.

gregorian_date A 17-byte fixed-length character string in the form YYYYMMDDHHMISS99 (the Gregorian format) representing local year, month, day, hour, minute,second, and millisecond.

(The example quoted here was run with a current date of 20 August 1996, that is **gregorian_date** = 19960820142117770.)

4 CEESCEN is called to set the century window to start with the current year.

5 The due date values are set up in the **due_date** fields.

6 CEEDAYS is called to convert the **due_date** value from the input record to an integer value Lilian date. The input date field to CEEDAYS requires a halfword length-prefixed character string, **due_date_pic**. to indicate its format.

7 CEEDATE is called to convert the due date value into a character string representation including a 4-digit year value.

For example, a record containing the **due_date** value ″01/30/03″, that is, 30th January 2003, produces a date value of **01/30/2003**.

8 After these statements the **years_left** field 9 contains the value **7**.

10 CEESCEN is called again to set the century window to start 100 years before the current system date.

11 The birth date values are set up in the **birth_date** fields.

12 CEEDAYS is called again to convert the **birth_date** value from the input record to an integer value Lilian date.

13 CEEDATE is called again to convert the date to a 4-digit year date.

For example, a record containing the birth_date value of ″04/05/09″, that is 5th April 1909, produces a date value of **04/05/1909**.

14 After these statements the **curr_age** field 15 contains the value **87**.

This example is a simple, but short-term, approach to the Year 2000 problem using the century window feature of LE/VSE with C/VSE.

5.5 The LE/VSE COUNTRY Code Option and the CEE5CTY Callable Service

LE/VSE provides a run-time option, COUNTRY, and a callable service, CEE5CTY, which set the default country code.

The country code affects the formats of date and time, the currency symbol, decimal separator, and the thousands separator.

You may need to change the format of these items to suit the standards of the country for which you are programming. You do this by using the COUNTRY option or the CEE5CTY callable service.

Therefore, when performing arithmetic on dates, particularly when manipulating dates to solve your Year 2000 problem, ensure you code the date fields in the correct format for the default country code in use.

The IBM-supplied country code defaults are listed in Appendix A of *IBM Language Environment for VSE/ESA Release 4*, SC33-6685. The installation-wide default for your installation will have been set when LE/VSE was installed. The COUNTRY run-time option and the CEE5CTY callable service are also described in this book.

The setting of the COUNTRY code run-time option also affects the format of the date and time in the options and storage reports generated by the RPTOPTS and RPTSTG run-time options. If the default country code has a date format with a 4-digit year, the date in these reports will appear with a 4-digit year. If the default country code has a date format with a 2-digit year, the date will appear with a 2-digit year.

For example, the US country code has the date format, **MM/DD/YY**, therefore the date on the options report produced on 15 January 2000 will be **01/15/00**.

However, if you live in Germany and have set your country code to DE, which has a date format of **DD.MM.YYYY**, the date on the report will be **15.01.2000**.

Note: Changing the country code setting does not change the default settings for the picture string set by CEESETL or *setlocale()*. The country code affects only the LE/VSE national language support services, not the LE/VSE locale callable services.

Chapter 6. Migration Experiences - Common Questions from Customers

This chapter relates a number of commonly asked questions posed through various IBM support organizations and customer accessible facilities, for example COBOL CFORUM and VSEESA CFORUM.

6.1 General

- **Since I am not receiving documentation updates anymore, where can I find information relating to changes to manuals for LE/VSE and related products?**

  Each publication should have an Information APAR describing changes - for example,the COBOL/VSE Migration Guide changes are contained in II09069. If there is not an Information APAR available then there are no changes required.

  See A.3, “Documentation” on page 68 for APAR numbers.

- **I have seen the term ″running in compatibility mode″ many times in relation to LE/VSE, what does it mean?**

  ″Compatibility mode″ means executing old programs compiled with pre-LE/VSE-conforming languages and executed using the LE/VSE run-time library. LE/VSE provides compatibility mode for both DOS/VS COBOL and VS COBOL II programs.
  There is no compatibility mode for either PL/I or C products.

6.2 COBOL

- **Do I need to recompile all my programs, if I wish to use LE/VSE?**

  This depends on whether you have done a source migration to COBOL/VSE, in which case a compile is always required. However, most DOS/VS COBOL and VS COBOL II programs can run unchanged using LE/VSE run time and a compile of these programs is not necessary.

- **Do I need to linkedit all my programs, if I wish to use LE/VSE?**

  This question needs to be more specific, because it will depend on whether the program has been compiled with DOS/COBOL or VS COBOL II. It is also probable that you have not retained object modules from previous compiles, so that a link-edit will also imply a recompile.

- **If I wish to migrate from DOS/COBOL to COBOL/VSE and LE/VSE, do I have to install VS COBOL II first?**

  No, COBOL/VSE is the strategic COBOL compiler and there are no technical reasons to install VS COBOL II first.

- **Will my DOS/VS COBOL main programs require recompile or relink to run in compatibility mode with LE/VSE?**

  No, relinking a DOS/VS COBOL main program with LE/VSE will have no affect. Relinking is only necessary if the program has been recompiled with COBOL/VSE.

- **Will DOS/VS COBOL subroutines called by COBOL/VSE programs require recompile and link-edit?**

If the subroutines were compiled originally with releases earlier than DOS/VS COBOL 1.3.1, then you should recompile the subroutines. If you are able to migrate these to COBOL/VSE, now is a good time!

- **Why are my output tapes being written as unlabeled tapes? I am providing the // TLBL and under DOS/VS COBOL I get a standard label tape.**

  This is because of the way that COBOL/VSE interprets the ASSIGN clause on the SELECT clause. The file name on the ASSIGN clause is used by LE/VSE at run time to look for a // TLBL JCL statement. If present then a standard label tape is written, if not, an unlabeled tape is written. You will need to change the ASSIGN clause to the COBOL/VSE format.

- **I am receiving File Status codes 35, 39 or 90 during file open processing in programs that have worked for years on DOS/VS COBOL. The message states that I have a file attribute mismatch, why?**

  LE/VSE COBOL conforms to ANSI 85 standards and does far more pre-open checking than DOS/VS COBOL. You may have to make adjustments to your programs to satisfy these requirements. See Appendix J Preventing File Status 39 for SAM Files in COBOL/VSE *Migration Guide*, GC26-8070 for some coding assistance.

- **I use CLOSE with REMOVAL to perform an unload of tapes with DOS/VS COBOL. If I migrate to COBOL/VSE and LE/VSE will this perform the same function?**

  CLOSE for REMOVAL causes confusion, because it doesn′t close the file, the file remains open and available for more processing. It causes a Force-End-Of-Volume to be issued, so that the tape rewinds and unloads. A subsequent CLOSE (without REMOVAL) is required to close the file. This behavior is the same across all VSE COBOL products and has not changed with COBOL/VSE and LE/VSE.

- **With DOS/VS COBOL CLOSE with LOCK will unload tapes as well as closing the file. Can I still do this with COBOL/VSE and LE/VSE?**

  Yes, this support was introduced in LE/VSE 1.1 APAR PN85885 and provides a means of closing a file and unloading the tape. Programs compiled with either COBOL/VSE or VS COBOL II and run with LE/VSE can use this function. This support is not provided in VS COBOL II run time.

- **Does LE/VSE provide Advanced Function Printing (AFP-5A) support?**

  This is provided in COBOL/VSE for WRITE AFTER ADVANCING via SPECIAL-NAMES paragraph.

  Note: A documentation change is provided in APAR PN77700.

6.3 COBOL and CICS

- **Will COBOL/VSE and LE/VSE support Macro Level Programs running under CICS/VSE?**

  No, all programs running with LE/VSE and CICS/VSE must use the command level interface.

- **Is COBOL2=YES required to initialize LE/VSE in CICS/VSE?**

  No. It is recommended that COBOL2=NO is specified when running LE/VSE. CICS/VSE will search for phase CEECCICS anywhere in the standard VSE

search sequence (SVA, LIBDEF). If it is found, then CICS will initialize LE/VSE.

- **There is a CICS Translator Option ′COBOL2′, but I could not find an option for COBOL/VSE. Is there an option for COBOL/VSE?**

  The option ′COBOL2′ or its equivalent ′ANSI85′ must be used when translating COBOL/VSE CICS command level programs. This will cause different COBOL code to be generated, and will also force the ′RENT′ COBOL compile option, which is required for LE/VSE CICS programs.

6.4 PL/I

- **Can the DOS PL/I transient library in the LIBDEF chain simply be replaced by the LE/VSE library?**

  No, programs compiled and linkedited with DOS PL/I must use the DOS PL/I resident library at link-edit time and the DOS PL/I transient library at run time.

  Note: This will include any products supplied by IBM or other vendors. The DOS PL/I transient library must be retained until the product can run using LE/VSE, that is, when the source has been recompiled with PL/I VSE.

- **Can DOS PL/I and PL/I VSE with LE/VSE coexist?**

  Yes, there are no conflicts in module names, so both can exist on the system without the risk of accessing the incorrect product. Remember that in a CICS environment PLI=YES initialization parameter is required if DOS PL/I programs are to be executed.

- **Does PL/I VSE support the use of the X′5A′ character for Advanced Function Printing?**

  Yes, but to write records larger than 512 bytes there are some additional considerations. LE/VSE PL/I run time will require that the record is defined as Variable Unblocked and has the CTLASA attribute. Refer to VSE/ESA System Macros Reference under DTFPR parameters for further discussion.

6.5 C

- **Can my C/370 for VSE compiled programs be run using LE/VSE Release 4 runtime libraries?**

  No! There is no object compatibility between these products, so although some of the module names may be the same, the LE/VSE library cannot be used in place of the C/370 run-time library.

- **How does CICS decide if C/370 or LE/VSE needs to be initialized?**

  It is possible for both to be initialized.
  To initialize C/370, CICS searches for phases EDCCICS and EDCXV.
  To initialize LE/VSE, CICS searches for phase CEECCICS.
  It is not recommended that this is done, keep the two products separate. The module names between the two products are not entirely unique, therefore there is the possibility of inadvertently accessing the wrong module.

6.6 LE/VSE

- **Is LE/VSE used during the compilation of my programs?**

  PL/I VSE and C/VSE use LE/VSE library routines during compilation. COBOL/VSE does not currently use LE/VSE during program compilation.

- **ALL31 run time option? What is it, when do I need it?**

  The ALL31 run-time option determines whether AMODE24 programs can be executed. If ALL31(ON) is used, AMODE24 programs cannot be executed. With ALL31(OFF) both AMODE24 and AMODE31 programs can be executed. However, the additional code in LE/VSE to test and switch modes where required, can result in performance degradation.

- **I am trying to pass LE/VSE run-time options to my program, why are they being ignored?**

  The format of the PARM= parameter on the EXEC JCL statement is implemented differently for COBOL and PL/I (and C) with LE/VSE. LE/VSE expects the run-time options to precede program arguments. However, the CBLOPTS run-time option by default reverses this for COBOL programs so that the program arguments are expected first. (This was done to provide compatibility with VS COBOL II.) To make the parameter format the same for all LE/VSE languages simply change the value of the CBLOPTS option to OFF.

- **Will LE/VSE run in VSE/ESA 370 Mode on VSE/ESA 1.4?**

  Yes, it is supported in this environment, although with additional mode switching code, and hence performance is degraded.

- **I am running VSE/ESA 1.4 which includes the old DOS/VS Assembler. Why do I need the High Level Assembler (HLASM) as a prerequisite for LE/VSE?**

  High Level Assembler is required if you plan to assemble options modules or user exits supplied with LE/VSE. This is because they contain Assembler instructions (such as AMODE) and utilize other language enhancements provided only in the High Level Assembler.

- **I assembled the CICS default options module, CEECOPT, to change the STACK option and omitted the *init_size* suboption. I received High Level Assembler warning messages and the value defaulted to 512K when I expected 4K.**

  The default values for omitted suboptions **are not the same** as the values assigned to those suboptions in the IBM-supplied default options modules, CEECOPT and CEEDOPT.
  The *LE/VSE Installation and Customization Guide*, under the description of each option, gives the default value for each suboption. The Installation Guide also states that when assembling CEECOPT or CEEDOPT **no option or suboption** may be omitted. They may be omitted when assembling the user options module CEEUOPT.

6.7 LE/VSE and CICS

- **LE/VSE supplies only CICS CSD program entries but my installation still uses CICS PPT table. Are sample PPT entries available?**

  Yes, the PPT equivalent entries are contained in Information APARs:

  - II08994 - LE/VSE Base
  - II08995 - LE/VSE COBOL
  - II08996 - LE/VSE PL/I

  Note: For LE/VSE Release 4 the PPT equivalent entries are distributed with the product as samples.

- **If I am running multiple CICS systems without MRO, do I need to run a separate LE/VSE in each region?**

  Yes, to run programs in each region that require LE/VSE, CICS/VSE must initialize LE/VSE in each region.

6.8 Year 2000

- **Do the LE/VSE-conforming compilers fully support the Year 2000 requirements?**

  Yes, but some trivial changes to COBOL/VSE and PL/I VSE have been made via maintenance to provide 4-digit years on compiler listing headings.

  APARs required

  PN84766 for COBOL/VSE
  PN87150 for PL/I VSE

- **Will DOS/VS COBOL be enhanced to support a 4-digit year?**

  No, DOS/VS only supports a 2-digit year. COBOL/VSE is the product required for full Year 2000 support.

- **Can LE/VSE callable services (such as those for date and time) be issued from a DOS/VS COBOL running with the LE/VSE run-time library?**

  No, DOS/VS COBOL programs cannot contain calls to LE/VSE services.

- **Will VS COBOL II be enhanced to support a 4-digit year?**

  No, VS COBOL II supports a 2-digit year. COBOL/VSE is the product required for full Year 2000 support.

  Note: Some LE/VSE callable services are supported by VS COBOL II as a migration aid only. Refer to APAR PN80806 for full details.

- **Will DOS/VS COBOL return the correct 2-digit year after the Year 2000?**

  Yes, DOS/VS COBOL will produce ′00′ for 2000, ′01′ for 2001 for example when date services are requested via the CURRENT-DATE or WHEN-COMPILED special registers.
  Compiler listings will also show 2-digit years.

- **Will DOS PL/I return the correct 2-digit year after the Year 2000?**

  Yes, DOS PL/I will produce ′00′ for 2000, ′01′ for 2001 for example when date services are requested via the DATE built-in function.
  Compiler listings will also show 2-digit years.

- **Can I use COBOL/VSE Intrinsic date functions and LE/VSE Callable Services date services together? What happens if I mix the dates returned from COBOL Intrinsic function with those from LE/VSE callable services?**

  No, you should not mix the two; COBOL/VSE intrinsic functions date functions use the Gregorian date (31 December 1600) as the base and the LE/VSE callable services date service use the Lilian date (14 October 1582).
  Use the CEECBLDY callable service (LE/VSE Release 4 only) for conversion between LE/VSE date routines and COBOL/VSE ANSI intrinsic function date routines.

6.9 Debug Tool for VSE (DT/VSE)

- **How do I order DT/VSE?**

  DT/VSE is only available as an optional feature of the LE/VSE-conforming compilers.

  Note: LE/VSE Release 4 (including C run-time functions) is required to support DT/VSE.

- **If I order multiple compilers, will I need DT/VSE for each compiler?**

  No, only one copy of DT/VSE is required and this will operate with all LE/VSE-conforming languages. If multiple copies are received, you only need to install DT/VSE once.

- **Can DT/VSE be used to debug CICS programs?**

  Yes, DT/VSE will debug both CICS and batch programs.

- **Must I change my source code and recompile in order to use DT/VSE?**

  No, source code does not need to be changed. However, a recompile is necessary with the ′TEST′ compile option to insert the requisite hooks into the object code.

Chapter 7. Interfaces to Other Subsystems

7.1 CICS/VSE

For LE/VSE programs running in a CICS/VSE environment there are a number of important considerations. The minimum supported release that will run LE/VSE programs is CICS/VSE Version 2 Release 3.

LE/VSE is initialized by CICS/VSE if the LE/VSE phase CEECCICS is found in the normal VSE/ESA library search sequence (which includes the SVA).
LE/VSE must be initialized by CICS/VSE to run programs compiled using an LE/VSE-conforming compiler.

Note: If LE/VSE phase CEECCICS has been loaded into the SVA then CICS/VSE will always attempt to initialize LE/VSE. Macro level CICS programs are not supported by LE/VSE.

Documentation

The following LE/VSE manuals contain information on how to run LE/VSE-conforming programs under CICS:

- *Installation and Customization Guide*, SC26-8064 (Release 1)
- *Installation and Customization Guide*, SC33-6682 (Release 4)

This should be read in conjunction with information provided in:

CICS/VSE System Definition and Operations Guide Version 2 Release 3, SC33-0706

7.1.1 COBOL

Application programs compiled with:

- DOS/VS COBOL
- VS COBOL II
- COBOL/VSE

can be run using LE/VSE run-time under CICS/VSE.

- Translator requirements

  ANSI85 or COBOL2 XOPTS parameter is required for programs to be compiled with COBOL/VSE (or VS COBOLII)

- Compiler requirements

  COBOL/VSE programs must be compiled with the RENT option.

- Link-edit requirements

  For COBOL/VSE programs, include module DFHELII rather than DFHECI in your link-edit statements.

- Run-time requirements

  To run CICS with LE/VSE initialized, set COBOL2=NO in your CICS/VSE initialization parameters. This applies even if running programs compiled with VS COBOL II.

7.1.2 PL/I

PL/I VSE and DOS PL/I programs can run concurrently.

- Translator requirements

  There are no special translator requirements for PL/I VSE programs.

- Compiler requirements

  Use the SYSTEM(CICS) compile-time option

- Link-edit requirements

  Change your link-edit JCL to include DFHELII instead of DFHPL1I and DFHEPI.

- Run-time requirements

  PLI=YES (System Initialization) is only required if DOS PL/I programs are coexisting with PL/I VSE in the CICS region.
  Don't specify PLI=YES if only PL/I VSE programs are to be run.

7.1.3 C

C/VSE programs cannot coexist with C/370 for VSE programs in a CICS region.

- Translator requirements

  There are no special translator requirements for C/VSE programs.

- Compiler requirements

  The RENT compile option must be used for C/VSE programs.

- Link-edit requirements

  The C prelinker must be run prior to the link-edit.

- Run-time requirements

  When running in system 370-mode, LE/VSE C run-time modules cannot be loaded by CICS/VSE because of phase size restrictions. These modules (CEEEV003 and EDCZ24) must be loaded into the SVA.

Chapter 8. Using LE/VSE Between Languages

This chapter describes some considerations that may be helpful when migrating pre-LE/VSE interlanguage (ILC) applications to an LE/VSE environment. Interlanguage Communication (ILC) is described in detail in the following LE/VSE publications:

- *Programming Guide*, SC26-8065 Release 1 (Part 5 and Chapter 25)
- *Writing Interlanguage Communication Applications*, SC33-6686 Release 4

8.1 LE/VSE ILC - What are the Advantages?

When coding applications that involve more than one programming language (including High Level Assembler) communicating with one another, there are two main considerations:

- Will it work?

  LE/VSE run-time environment provides the certainty that calls between languages will work correctly, if all those participating languages are LE/VSE-conforming. This includes the High Level Assembler, which is a special case because it is LE/VSE-conforming only if coded correctly.

- Will it work efficiently?

  One of the major benefits of ILC under LE/VSE is the performance improvement gained by having a single run-time environment. If all programs (calling and called) are LE/VSE-conforming then the benefits can be realized, and you will have the certainty that it will work.

How do I make my programs LE/VSE-conforming?
With HLLs, recompile your program with an LE/VSE-conforming compiler:

- COBOL/VSE
- PL/I VSE
- C/VSE

A program or subroutine written using the High Level Assembler must use special coding (LE/VSE macros) to make it LE/VSE-conforming so that it can use the single run-time environment.

8.2 Pre-LE/VSE Conforming Assembler calling HLLs

Many applications consist of an Assembler program loading and calling subroutines written in HLLs. These HLLs can also be mixed-language applications such as COBOL and PL/I link-edited statically together.
If you are migrating your COBOL subroutine to COBOL/VSE, and PL/I subroutine to PL/I VSE, change your Assembler routine to make it LE/VSE-conforming.

8.3 How to Code LE/VSE-conforming Assembler Main Program

The examples in this section illustrate how to code a simple Assembler program containing LE/VSE macros (such as CEEENTRY and CEETERM) to make it LE/VSE-conforming.
It calls a COBOL/VSE program that is statically linked with the Assembler program.

CEEILCOB

```
*/* Module/File Name: CEEILCOB */
* ═══════════════════════════════════════════════════════════
*   Bring up the LE/VSE environment
* ═══════════════════════════════════════════════════════════
CEEILCOB CEEENTRY PPA=MAINPPA
         USING WORKAREA,13
*
*   Call the COBOL program
*
         CALL  ASMCOB,(X,Y)       Invoke COBOL subroutine
*
*   Call the CEEMOUT service
*
         CALL  CEEMOUT,(MESSAGE,DESTCODE,FC)  Dispatch message
         CLC   FC(8),CEE000       Was MOUT successful?
         BE    GOOD               Yes.. skip error logic
         LH    2,MSGNO            No.. get message number
         DUMP  RC=(2)                  LIGHTS OUT!
*
*   Terminate the LE/VSE environment
*
GOOD     CEETERM  RC=0            Terminate with return code zero
*
* ---------------------------------------------------------------
*
*   Data Constants and Static Variables
*
Y        DC    PL3¢+200¢          2nd parm to COBOL program (input)
MESSAGE  DS    0H
MSGLEN   DC    Y(MSGEND-MSGTEXT)
MSGTEXT  DC    C¢AFTER CALL TO COBOL: X=¢
X        DS    ZL6                1st parm for COBOL program (output)
MSGEND   EQU   *
```

Figure 8 (Part 1 of 2). LE/VSE-Conforming Assembler Routine Calling COBOL/VSE Routine

```
DESTCODE DC    F¢2¢                Directs message to MSGFILE
CEE000   DC    3F¢0¢               Success condition token
FC       DS    0F                  12-byte feedback/condition code
SEV      DS    H                   severity
MSGNO    DS    H                   message number
CSC      DS    X                   flags - case/sev/control
CASE     EQU   X¢C0¢  11.....      case (1 or 2)
SEVER    EQU   X¢38¢  ..111..      severity (0 thru 4)
CNTRL    EQU   X¢03¢  .....11      control (1=IBM FACID, 0=USER)
FACID    DS    CL3                 facility ID
ISI      DS    F                   index into ISI block
*
MAINPPA  CEEPPA                    Constants describing the code block
* ═══════════════════════════════════════════════════════════════
*   Workarea
* ═══════════════════════════════════════════════════════════════
WORKAREA DSECT
         CEEDSA  ,                 Mapping of the Dynamic Save Area
         CEECAA  ,                 Mapping of the Common Anchor Area
         CEEEDB  ,                 Mapping of the Enclave Data Block
*
         END   CEEILCOB
```

Figure 8 (Part 2 of 2). LE/VSE-Conforming Assembler Routine Calling COBOL/VSE Routine

IGZTASM

```
      *Module/File Name: IGZTASM
       IDENTIFICATION DIVISION.
         PROGRAM-ID. ASMCOB.
      *
       ENVIRONMENT DIVISION.
       DATA DIVISION.
         WORKING-STORAGE SECTION.
      *
       LINKAGE SECTION.
         01 X PIC +9(5).
         01 Y PIC S9(5) COMP-3.
      *
       PROCEDURE DIVISION USING X Y.
           COMPUTE X = Y + 1.
      *
           GOBACK.
```

Figure 9. COBOL/VSE Routine Called from LE/VSE-Conforming Assembler

8.4 ILC in the CICS Environment

In general, LE/VSE provides the same ILC support for applications running under CICS as for those running in a non-CICS environment.
There is one major exception - there is no support for LE/VSE-conforming Assembler main routines under CICS.
Some other points you should note:

- ILC applications must be linked with the CICS stub, DFHELII, in order to get ILC support under CICS
- You can use EXEC CICS LINK from an LE/VSE-conforming routine to call non-LE/VSE conforming routines and vice versa. This may provide a technique to migrate routines within an application in a more controlled manner.
  However you should evaluate the CICS performance issues involved, as noted in A.4, “Performance Recommendations” on page 68.

Appendix A. Service Information and Performance Considerations

This appendix provides technical detail relating to the service and performance of LE/VSE. It also provides information about particular issues relating to vendor products.

A.1 Recommended Service (Status October 1996)

Review the following list of recommended maintenance and ensure you apply PTFs if important to your installation.

- Migration from DOS/VS COBOL and DOS PL/I

  These PTFs address problems reported as behavior differences between DOS/VS COBOL, DOS PL/I and LE/VSE.

  - LE/VSE COBOL PN79374/UN87859 - Blank card problem
  - LE/VSE COBOL PN84947/UN92489 - Performance with VSAM
  - LE/VSE BASE PN78876/UN85561 - Max Punch Card length set to 512
  - LE/VSE COBOL PN78877/UN85715 - Max Punch Card length set to 512
  - LE/VSE PL/I PN83941/UN91933 - Max Punch Card length set to 512
  - LE/VSE COBOL PN85885/UN92666 - CLOSE WITH LOCK rewind/unload

- Vendor products

  - LE/VSE COBOL PN79258/UN86586 - Vendors with Tape managers
  - COBOL/VSE PN84081/UN93151 - Vendors with Disk/Tape Managers
  - LE/VSE PL/I PN82323/UN91202 - Vendors using PRODEXIT and PL/I
  - LE/VSE PL/I PN85269/UN????? - Vendors using PRODEXIT and PL/I

    At the time of writing this document, the number for this PTF was not known. Please check with your local service representative.

A.2 Ensuring Service is Applied Correctly

- How do I know if a PTF is applied?

  - There is no reliable way to determine if a PTF is applied, without retaining all MSHP PTF install listings.
    An MSHP RETRACE will show you if MSHP indicates that the PTF has been applied.

- How do I ensure that the application really uses the PTF?

  - Check the MSHP listing of the PTF install and determine which phases have been link-edited and the libraries updated.
  - Check if each phase modified by the PTF resides in the SVA.
  - If so, then an IPL is required to refresh the phases in the SVA.
  - Without an IPL, the PTF may be tested by including the SDL parameter in the LIBDEF statement to search the product library before searching the SVA.

A.3 Documentation

Documentation updates to existing manuals are contained in Information APARs.

- II09066 - COBOL/VSE Installation and Customization (Release 1)
- II09067 - COBOL/VSE Language Reference (Release 1)
- II09068 - COBOL/VSE Diagnosis Guide (Release 1)
- II09069 - COBOL/VSE Migration Guide (Release 1)
- II09070 - COBOL/VSE Programming Guide (Release 1)
- II09071 - COBOL/VSE General Information (Release 1)
- II09072 - LE/VSE Installation and Customization (Release 1)
- II09073 - LE/VSE Debugging Guide and Run-Time Messages (Release 1)
- II09074 - LE/VSE Programming Guide (Release 1)
- II09075 - LE/VSE Diagnosis Guide (Release 1)
- II09532 - PL/I VSE Programming Guide (Release 1)

Future APARs will include updates to Compiler Manuals for LE/VSE Release 4 functions, C/VSE and Debug Tool for VSE.

A.4 Performance Recommendations

Although the LE/VSE and language publications provide information about performance, here are some very specific recommendations to help you overcome potential problems.

- DFSORT/VSE STXIT/NOSTXIT option

  Were you aware that DFSORT/VSE is provided as the only IBM sort product with VSE/ESA 2.1 and above?
  Maybe not! This is because SORT control statements are mostly compatible amongst the different products, and you may be using DFSORT/VSE without being aware of it. Further, you may not be aware that DFSORT/VSE has introduced new error handling features which will have a direct effect on SORT performance. The new STXIT option introduced by DFSORT/VSE (and supplied as the default), will invoke additional SVC overhead for every DFSORT/VSE user exit.

  Why may this be a problem?
  All language products use SORT user exits to handle input and output processing (E15 and E35 exits). These exits are usually called for each record passed to SORT(E15) and passed back from SORT(E35). With the STXIT option, DFSORT/VSE issues the STXIT SVC on return from *every* SORT user exit. So for languages, there is an additional overhead for each record passed to sort and each record returned. In addition, LE/VSE also provides error condition handling via the TRAP(ON) run-time option, so it also must issue the STXIT SVC to reestablish its error handling on return from DFSORT/VSE. But this will only occur if the DFSORT/VSE STXIT option is set. So use the DFSORT/VSE NOSTXIT option. Either change the DFSORT/VSE supplied defaults (STXIT=YES|NO) or add NOSTXIT to a DFSORT/VSE OPTION statement and pass it to the application program. Information APAR II 09745 furnishes you with more details.

Notes:

1. For COBOL/VSE, the FASTSRT compiler option is available to bypass DFSORT/VSE exit processing in some circumstances. Read the appropriate section in the *IBM COBOL for VSE/ESA Programming Guide* for details.
2. If you use a non-IBM SORT product, the same considerations may apply. Change the equivalent SORT STXIT option if necessary.
3. The same considerations will apply to older language products using DFSORT/VSE, such as DOS/VS COBOL and DOS PL/I.

- APAR PN84947/UN92489 VSAM SHOWCB - (LE/VSE COBOL only)

  Improve your VSAM performance, particularly when reading a large number of records, by ensuring that this fix is applied.
  Although there will be an improvement on VSE/ESA 1.4 there will be even better gains on a VSE/ESA 2.1 system with Turbo Dispatcher.

- ALL31(ON) - LE/VSE Run-time Option

  Running all applications with run-time option ALL31(ON) provides a performance improvement over ALL31(OFF). This is because there is code within LE/VSE to test and switch addressing modes if required. This has further implications.
  Running with ALL31(OFF) means that HEAP and STACK storage must be allocated below-the-line.
  Under CICS with ALL31(OFF) the Thread control blocks are not pre-allocated, so the Run-Unit Initialization routine must issue a CICS GETMAIN for below-the-line storage for these control blocks. This therefore requires the Run-Unit Termination routine to also issue a CICS FREEMAIN to release the storage for these control blocks.

  The supplied batch default is ALL31(OFF), the supplied CICS default is ALL31(ON).

- Dynamic CALL instead of EXEC CICS LINK

  Within a CICS environment use a dynamic CALL rather than an EXEC CICS LINK. DOS/VS COBOL had a restriction that any called subroutine could not issue EXEC CICS commands. This resulted in widespread use of EXEC CICS LINK to subroutines. However, this restriction does not exist with COBOL/VSE and LE/VSE and dynamic calls are recommended.

  Why? EXEC CICS LINK causes LE/VSE to create a nested enclave which will have its own resources such as storage. The termination of this enclave with an EXEC CICS RETURN will free these resources, but along the way there will be more LE/VSE code executed and a corresponding degradation in performance.
  CICS/VSE 2.3 uses VSE/ESA GETVIS/FREEVIS services for CICS storage obtained or released above 16MB, so this will incur further overhead.

- COBOL/VSE compiler options that affect performance

  The compiler options that you select either explicitly or by default, can affect performance at run time. Ask your IBM representative for a copy of the COBPERF PACKAGE available on MKTTOOLS disk. This paper discusses IBM COBOL for MVS and VM as well as LE/370, but many of the recommendations are also valid for COBOL/VSE and LE/VSE.

- To gain maximum benefit from storage above the 16MB line, always attempt to linkedit your program AMODE 31 and RMODE ANY, either by default, or if appropriate by using AMODE and RMODE linkedit parameters.
  This is especially true for programs linkedited AMODE ANY because VSE/ESA will always invoke these programs in AMODE 24. In this case you must explicitly override the linkedit AMODE by means of linkedit parameters.

  Note: For COBOL/VSE programs the RMODE and RENT compiler options can affect the resulting AMODE and RMODE of the linkedited program.

A.5 Service Issues with Vendor Products

Most vendor tape and disk managers such as CA-DYNAM, CA-DYNAM/T and CA-EPIC, interface with LE/VSE OPEN processing. Typically, these products intercept the OPEN to provide the required functionality.
Because of the increased number of pre-OPEN checks introduced by VS COBOL II (for ANSI 85 conformance), changes to VS COBOL II OPEN processing were required to enable the vendor code to function correctly. For VS COBOL II the vendor interface was provided by a number of VS COBOL II APARs/PTFs.
The solution provided was considered to be a temporary measure because of the technique used (communicating via the Label Parameter List (LPL) during a call to the LABEL macro processor ($IJBSLA)).
With LE/VSE, the VSE/ESA PRODEXIT facility was introduced, providing a standard interface without the need to change VSE/ESA control blocks. Ask your vendor, if the product they supply, requires the PRODEXIT interface.

What kinds of problems can be expected, if the vendor interface has not been enabled?
Typically problems will be seen during OPEN processing of SAM or VSAM managed SAM files.

How do I obtain diagnostic information for file OPEN problems?
An SDAID Branch trace is usually the best way to obtain information to help debug a failing file OPEN.
You will be directed by your IBM support representative on the modules that must be traced. This will vary according to the file type (for example, VSAM or SAM) of the file being opened.
The following are examples of SDAID control statements:

- COBOL SAM file

  ```
  TRACE BR AR=ALL PHASE=IGZEQOC OUTP=GREG
  ```

- COBOL VSAM file

  ```
  TRACE BR AR=ALL PHASE=IGZEVOC OUTP=GREG
  TRACE BR AR=ALL PHASE=IGZEVOP OUTP=GREG
  ```

- PL/I SAM file

  ```
  TRACE BR AR=ALL PHASE=IBMROPDA OUTP=GREG
  ```

- PL/I VSAM file

  ```
  TRACE BR AR=ALL PHASE=IBMROPEA OUTP=GREG
  ```

Appendix B. Special Notices

This publication is intended to help System customer and IBM technical personnel to:

- Better understand the concepts of Language Environment for VSE
- Better understand the benefits provided by LE/VSE
- Provide guidelines for the migration to LE/SE.

The information in this publication is not intended as a specification of any programming interfaces that are provided by LE/VSE. See the PUBLICATIONS section of the IBM Programming Announcement for VSE/ESA 2.1, program number 5690-VSE, for more information about what publications are considered to be product documentation.

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

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other

operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

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

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

Advanced Function Printing	AFP
C/370	CICS
CICS/VSE	DFSORT
IBM	ILE
ISSC	Language Environment
OS/2	VSE/ESA

The following terms are trademarks of other companies:

C-bus	Corollary, Inc.
CA	Computer Associates International, Inc.
CA-DYNAM	CA International
CA-DYNAMT	CA International
CA-EPIC	CA International
HP	Hewlett-Packard Company
PC Direct	Ziff Communications Company (used by IBM Corporation under license)
UNIX	X/Open Company Ltd. (registered trademark in the United States and other countries)
Windows, Windows 95 logo	Microsoft Corporation

Appendix C. Related Publications

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

C.1 International Technical Support Organization Publications

For information on ordering ITSO publications see “How To Get ITSO Redbooks” on page 77.

C.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RISC System/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RISC System/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection (available soon)	SBOF-7250	SK2T-8042

C.3 Other Publications

These publications are also relevant as further information sources.

C.3.1 Language Environment Publications

Language Environment for VSE/ESA Release 1

- *Diagnosis Guide*, SC26-8060
- *Fact Sheet*, GC26-8062
- *Concepts Guide*, GC26-8063
- *Installation and Customization Guide*, SC26-8064
- *Programming Guide*, SC26-8065

Attention

The following manuals will be available together with the products.

Language Environment for VSE/ESA Release 4

- *Fact Sheet*, GC33-6679
- *Concepts Guide*, GC33-6680
- *Debugging Guide and Run-Time Messages*, SC33-6681
- *Installation and Customization Guide*, SC33-6682

- *Programming Guide*, SC33-6684
- *Programming Reference*, SC33-6685
- *Writing Interlanguage Communication Applications*, SC33-6686
- *Run-Time Migration Guide*, SC33-6687
- *C Run-Time Programming Guide*, SC33-6688
- *C Run-Time Library Reference*, SC33-6689

C.3.2 LE/VSE-Conforming Languages

COBOL for VSE/ESA

- *General Information*, GC26-8068
- *Migration Guide*, GC26-8070
- *Installation and Customization Guide*, SC26-8071
- *Programming Guide*, SC26-8072
- *Language Reference*, SC26-8073
- *Diagnosis Guide*, SC26-8528

PL/I for VSE/ESA

- *Fact Sheet*, GC26-8052
- *Programming Guide*, SC26-8053
- *Language Reference*, SC26-8054
- *Migration Guide*, SC26-8056
- *Installation and Customization Guide*, SC26-8057
- *Diagnosis Guide*, SC26-8058
- *Compile-Time Messages and Codes*, SC26-8059

Softcopy Publications

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

- *VSE Collection*, SK2T-0060

Attention

The following manuals will be available together with the products.

C for VSE/ESA

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

Debug Tool for VSE/ESA

- *User's Guide and Reference*, SC26-8797
- *Installation and Customization Guide*, SC26-8798

C.3.3 Additional Information

The following packages contain useful information relating to this document. The packages are available upon request from your IBM representative:

- *VE21PERF PACKAGE* - VSE/ESA 2.1 Performance Considerations

  A separate document about the performance aspects of the Turbo Dispatcher is included.

- *VE13PERF PACKAGE* - VSE/ESA 1.3 Performance Considerations
- *VSENEW11 PACKAGE* - VSE/ESA Newsletter, Third/Fourth Quarter 1995

The packages are stored on the IBM internal tools disk IBMVSE. IBM employees may obtain a copy of the packages by entering the following command from their local VM user IDs:

```
TOOLS SENDTO BOEVM3 VMTOOLS IBMVSE GET xxxxxxxx PACKAGE
```

where xxxxxxxx is the name of the package.

The packages are also available on the Internet in the:

- FTP Server at ***lscftp.kgn.ibm.com/pub/vse/docs***
- VSE/ESA Home Page at ***www.ibm.de/go/d00000166***

These packages are in zipped format that can be down loaded, unzipped, and printed on a Postscript-capable printer:

- *VE21PERF.ZIP* - VSE/ESA 2.1 Performance Considerations
- *VE21TDP.ZIP* - Performance Considerations for the VSE/ESA 2.1 Turbo Dispatcher
- *VE13PERF.ZIP* - VSE/ESA 1.3 Performance Considerations
- *VSENEW11.ZIP* - VSE/ESA Newsletter, Third/Fourth Quarter 1995

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **ITSO4USA category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

- **Please put me on the mailing list for updated versions of the IBM Redbook Catalog.**

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number

- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

List of Abbreviations

APAR	Authorized Program Analysis Report
ANSI	American National Standards Institute
BCD	Binary Coded Decimal
CCCA	COBOL and CICS Command Level Conversion Aid
CIB	Condition Information Block
CICS	Customer Information Control System
CSD	CICS System Definition Data Set
DBCS	Double-byte Character Set
DCT	Destination Control Table
DTF	DEFINE THE FILE macro
DT/VSE	Debug Tool for VSE/ESA
HLASM	High Level Assembler
HLL	High Level Language
IBM	International Business Machines Corporation
ICCF	Interactive Computing and Control Facility
ILC	Interlanguage Communication
IPL	Initial Program Load
ISAM	Indexed Sequential Access Method
ITSO	International Technical Support Organization
JCL	Job Control Language
JECL	Job Entry Control Language
LE	Language Environment
LPL	Label Parameter List
MRO	Multiregion Operation
MSHP	Maintain System History Program
PTF	Program Temporary Fix
PPT	Processing Program Table
SAM	Sequential Access Method
SDAID	System Debugging Aid
SDL	System Directory List
SLI	Source Library Inclusion statement
SVA	Shared Virtual Area
SVC	Supervisor Call
VSAM	Virtual Storage Access Method
VTOC	Volume Table Of Contents

Index

Numerics

A

B

C

D

E

R

S

T

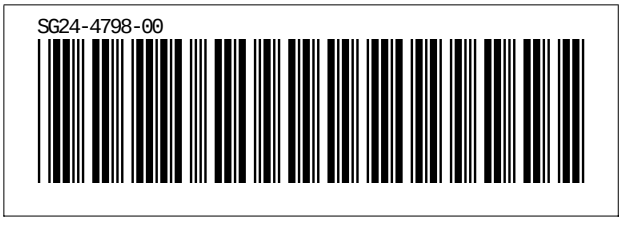
U

V

X

Y

Printed in U.S.A.



Artwork Definitions

Table Definitions

Figures

48

ASMMN	4798CH08	64	8
COBAM	4798CH08	65	9

Headings

id	File	Page	References
CH1	4798CH01	1	Chapter 1, Introduction to LE/VSE and Benefits of Using LE/VSE xi
CH2	4798CH02	9	Chapter 2, Migration Scenarios xi, 1, 3
CH3	4798CH03	13	Chapter 3, Migration from Old Languages to New Languages xi
CH4	4798CH04	19	Chapter 4, Migration from LE/VSE Release 1 to LE/VSE Release 4 xi
HOPTREM	4798CH04	22	4.3.5, Run-time Options Removed since LE/VSE Release 1 21
CRUN	4798CH04	26	4.4.4, C Language Run-time Support Shipped with VSE/ESA 2.2 25
CH5	4798CH05	33	Chapter 5, Year 2000 Considerations xi, 14, 15, 20
CH6	4798CH06	55	Chapter 6, Migration Experiences - Common Questions from Customers xi
CH7	4798CH07	61	Chapter 7, Interfaces to Other Subsystems xi
CH8	4798CH08	63	Chapter 8, Using LE/VSE Between Languages xi, 1
APPXA	4798AX01	67	Appendix A, Service Information and Performance Considerations xii
APPXAD	4798AX01	68	A.3, Documentation 55
APPXAP	4798AX01	68	A.4, Performance Recommendations 66
NOTICES	SG244798 SCRIPT	71	Appendix B, Special Notices ii
BIBL	4798BIBL	73	Appendix C, Related Publications 9, 13
ORDER	REDB$ORD	77	How To Get ITSO Redbooks 73

Index Entries

id	File	Page	References
LEIND	4798CH01	1	(1) LE/VSE 2, 5, 5, 6, 19, 19, 22, 22, 22, 23, 23, 24, 24, 24, 24, 28, 28, 29, 53, 58, 61, 61, 62, 62, 68
COBIND	4798CH01	6	(1) COBOL/VSE 6, 6, 6, 9, 10, 10, 14, 14, 14, 14, 15, 17, 17, 34, 34, 35, 55, 61, 69, 69
PLIND	4798CH01	6	(1) PL/I 6, 7, 7, 11, 11, 15, 15, 15, 15, 15, 16, 17, 42, 47, 57, 62
CIND	4798CH01	7	(1) C/VSE

Tables

id	File	Page	References
TAB1A	4798CH01	1	1 1, 1
TAB1	4798CH02	10	2 10, 17
TAB2	4798CH02	11	3 11
TAB3	4798CH02	12	4 12
PACK	4798CH04	24	7 24, 26
SVAL	4798CH04	27	8 27

Processing Options

Runtime values:

Document fileid	SG244798 SCRIPT
Document type	USERDOC
Document style	REDBOOK
Profile	EDFPRF30
Service Level	0029
SCRIPT/VS Release	4.0.0
Date	96.10.17
Time	05:02:28
Device	3820A
Number of Passes	4
Index	YES
SYSVAR D	YES
SYSVAR G	INLINE
SYSVAR S	OFFSET
SYSVAR X	YES

Formatting values used:

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

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

Imbed Trace

Page 0	4798SU
Page 0	4798VARS
Page 0	REDB$BOE
Page i	REDB$ED1
Page i	4798EDNO
Page i	REDB$ED2
Page xi	4798ABST
Page xi	4798ORG
Page xii	4798ACKS
Page xii	REDB$COM
Page xii	4798MAIN
Page xii	4798CH01
Page 8	4798CH02
Page 12	4798CH03
Page 18	4798CH04
Page 31	4798CH05
Page 53	4798CH06
Page 60	4798CH07
Page 62	4798CH08
Page 66	4798AX01
Page 71	4798SPEC
Page 71	REDB$SPE
Page 72	4798TMKS
Page 72	4798BIBL
Page 73	REDB$BIB
Page 76	REDB$ORD
Page 79	4798ABRV