IBM Library Server Print Preview

DOCNUM = SC09-2423-00
DATETIME = 10/09/96 08:33:53
BLDVERS = 1.2
TITLE = C/VSE V1R1 Migration Guide
AUTHOR =

PATH = /home/webapps/epubs/htdocs/book

COVER Book Cover

IBM C for VSE/ESA

Migration Guide

Release 1

Document Number SC09-2423-00

Program Number
5686-A01

NOTICES Notices

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" in topic FRONT_1.

EDITION Edition Notice

First Edition (December 1996)

This edition applies to Version 1, Release 1, Modification Level 0, of IBM C for VSE/ESA (Program 5686-A01); Version 1, Release 4, Modification Level 0, of IBM Language Environment for VSE/ESA (Program 5686-094); the VSE C Language Run-Time Support feature of VSE/ESA Version 2 Release 2 (Program 5690-VSE); and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

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

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

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 Eglinton Avenue East
North York, Ontario, Canada M3C 1H7

You can also send your comments by facsimile (attention: RCF Coordinator), or you can send your comments electronically to IBM. See "Communicating Your Comments to IBM" for a description of the methods. This page immediately precedes the Readers' Comment Form at the back of this publication.

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

CONTENTS Table of Contents

FIGURES Figures

FRONT_1 Notices

Canada Ltd., Department 071, 1150 Eglinton Avenue East, North York, Ontario M3C 1H7, Canada. Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

Subtopics:

FRONT_1.1 Programming Interface Information

This book is intended to help you migrate VSE C application programs to the IBM C for VSE/ESA compiler, and to IBM Language Environment for VSE/ESA. It documents General-Use Programming Interface and associated guidance information provided by these products.

General-Use Programming Interfaces allow the customer to write programs that obtain the services of the IBM C for VSE/ESA compiler and IBM Language Environment for VSE/ESA.

FRONT_1.2 Standards

Extracts are reprinted from *IEEE Std 1003.1--1990, IEEE Standard Information Technology--Portable Operating System Interface (POSIX)--Part 1: System Application Program Interface (API) [C language]*, copyright 1990 by the Institute of Electrical and Electronic Engineers, Inc.

Extracts are reprinted from IEEE P1003.1a Draft 6 July 1991, Draft Revision to Information Technology--Portable Operating System Interface (POSIX), Part 1: System Application Program Interface (API) [C Language], copyright 1992 by the Institute of Electrical and Electronic Engineers, Inc.

Extracts are reprinted from *IEEE Std 1003.2--1992, IEEE Standard Information Technology--Portable Operating System Interface (POSIX)--Part 2: Shells and Utilities*, copyright 1990 by the Institute of Electrical and Electronic Engineers, Inc.

Extracts are reprinted from *IEEE Std P1003.4a/D6--1992, IEEE Draft Standard Information Technology--Portable Operating System Interface (POSIX)--Part 1: System Application Program Interface (API)--Amendment 2: Threads Extension [C language]*, copyright 1990 by the Institute of Electrical and Electronic Engineers, Inc.

FRONT_1.3 Trademarks

FRONT_2 About This Book

This book describes how to migrate VSE C applications from the IBM C/370 (C/370) Version 2 Release 1 Compiler to the IBM C for VSE/ESA (C/VSE) Release 1 compiler. It also describes how to migrate C applications to IBM Language Environment for VSE/ESA (LE/VSE) Release 4 from the IBM C/370 Version 2 Release 1 Library.

Note: Throughout this book, references to LE/VSE also apply to the VSE C Language Run-Time Support feature of VSE/ESA Version 2 Release 2.

This book does not discuss all of the enhancements that have been made to the C/VSE compiler and LE/VSE Release 4, relative to the C/370 Compiler and Library. Its purpose is to help programmers determine what must be done to continue using existing source code, and to point out differences in behavior between products that programmers should be aware of. Note

that in most cases, existing well-written applications will, without modification, compile successfully with C/VSE and run successfully with LE/VSE.

To use this book, or any other books in the C/VSE library, you must have a working knowledge of the C programming language, the operating system, and where appropriate, the related products.

Subtopics:

FRONT_2.1 The C Language

The C language is a general purpose, function-oriented programming language that allows a programmer to create applications quickly and easily. C provides high-level control statements and data types as do other structured programming languages, and it also provides many of the benefits of a low-level language. Using the C/VSE language, you can write portable code conforming to the ANSI standard.

IBM offers the C language on other platforms, such as the OS/2, AIX/6000, OS/400, OS/390, and VM operating systems.

The elements of the C/VSE implementation include:

- All elements of the joint ISO and IEC standard: ISO/IEC 9899:1990 (E)
- ANSI/ISO 9899:1990[1992] (formerly ANSI X3.159-1989 C)
- Locale based internationalization support as defined in: ISO/IEC DIS 9945-2:1992/IEEE POSIX 1003.2-1992 Draft 12 (There are some limitations to fully-compliant behavior as noted in the *LE/VSE C Run-Time Programming Guide*.)
- Extended multibyte and wide character utilities as defined by a subset of the Programming Language C Amendment 1, which will be ISO/IEC 9899:1990/Amendment 1:1994(E)

FRONT_2.2 IBM Language Environment for VSE/ESA

C/VSE exploits the C run-time environment and library of run-time callable services provided by IBM Language Environment for VSE/ESA (LE/VSE).

LE/VSE establishes a common run-time environment and common run-time callable services for language products, user programs, and other products.

The common execution environment is made up of data items and services performed by library routines available to a particular application running in the environment. The services that LE/VSE provides to your application may include:

- Services that satisfy basic requirements common to most applications. These include support for the initialization and termination of applications, allocation of storage, support for interlanguage communication (ILC), and condition handling.

- Extended services often needed by applications. These functions are contained within a library of callable routines, and include interfaces to operating system functions and a variety of other commonly used functions.

- Run-time options that help the execution, performance tuning, performance, and diagnosis of your application.

- Access to language-specific library routines, such as the C functions.

FRONT_2.3 Using Your Documentation

The publications in the C/VSE and LE/VSE libraries are designed to help you develop C/VSE applications that run with LE/VSE. Each publication helps you perform a different task. For a complete list of publications you might need, see "Bibliography" in topic BIBLIOGRAPHY. Figure 1 lists the publications in the C/VSE library.

Figure 1. How to Use C/VSE Publications

To...	Use...	
Plan for, install, customize, and maintain C/VSE	Installation and Customization Guide	GC09-2422
Migrate VSE applications from C/370 to C/VSE	*Migration Guide*	SC09-2423
Get details on C/VSE syntax and specifications of language elements	Language Reference	SC09-2425
Find syntax for compile-time options; compile your C/VSE applications; get details on compile-time messages	User's Guide	SC09-2424

Diagnose compiler problems and report them to IBM	Diagnosis Guide	GC09-2426
Understand warranty information	*Licensed Program Specifications*	GC09-2421

Figure 2 lists the publications in the LE/VSE library. These include publications designed to help you develop and debug your C/VSE applications, diagnose run-time problems that occur in your C/VSE applications, and use C/VSE-related utilities.

Figure 2. How to Use LE/VSE Publications

To...	**Use...**	
Evaluate LE/VSE	*Fact Sheet* *Concepts Guide*	GC33-6679 GC33-6680
Plan for, install, customize, and maintain LE/VSE	*Installation and Customization Guide*	SC33-6682
Understand the LE/VSE program models and concepts	*Concepts Guide* *Programming Guide*	GC33-6680 SC33-6684
Find syntax for LE/VSE run-time options and callable services	*Programming Reference*	SC33-6685
Develop your C/VSE applications	*Programming Guide* *C Run-Time Programming Guide* *C Run-Time Library Reference*	SC33-6684 SC33-6688 SC33-6689
Develop interlanguage communication (ILC) applications	*Writing Interlanguage Communication Applications*	SC33-6686
Debug your C/VSE applications and get details on run-time messages	*Debugging Guide and Run-Time Messages*	SC33-6681
Migrate applications to LE/VSE	*Run-Time Migration Guide*	SC33-6687
Diagnose run-time problems that occur in your C/VSE applications	*Debugging Guide and Run-Time Messages*	SC33-6681
Use C/VSE-related utilities	*C Run-Time Programming Guide*	SC33-6688
Understand warranty information	*Licensed Program Specifications*	GC33-6683

Subtopics:

- FRONT_2.3.1 Softcopy Examples

FRONT_2.3.1 Softcopy Examples

Most examples in the following books are available in machine-readable form:

- *C/VSE Installation and Customization Guide*, GC09-2422
- *C/VSE User's Guide*, SC09-2424
- *C/VSE Language Reference*, SC09-2425

Softcopy examples are indicated in the book by a label in the form, EDCX*bnnn*. The *b* refers to the book:

- I is the *C/VSE Installation and Customization Guide*
- U is the *C/VSE User's Guide*
- R is the *C/VSE Language Reference*

Softcopy examples are installed on your system along with C/VSE, in the sublibrary PRD2.DBASE.

Example member names are the same as the labels indicated in the book.

Contact your system programmer if the default names are not used at your installation.

FRONT_2.4 How to Read the Syntax Diagrams

In this book, syntax for commands, directives, and statements is described using the following structure:

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

  A double right-arrowhead indicates the beginning of a command, directive, or statement; the single right-arrowhead indicates that it is continued on the next line. (In the following diagrams, statement is used to represent a command, directive, or statement.)

  ```
  >>__statement__>
  ```

  The following indicates a continuation; the opposing arrowheads indicate the end of a command, directive, or statement.

  ```
  >__statement__><
  ```

  Diagrams of syntactical units other than complete commands, directives, or statements look like this:

```
>__statement__>
```

- Required items are on the horizontal line (the main path).

```
>>__statement__required_item__><
```

- IBM-supplied default items are above the main path.

```
                 _default_item_
>>__statement__|_____________|__><
```

- Optional items are below the main path.

```
>>__statement__ _____________ __><
               |_optional_item_|
```

- If you can choose from two or more items, they are vertical in a stack.

  If you *must* choose one of the items, one item of the stack is on the main path.

```
>>__statement__ _required_choice1_ __><
               |_required_choice2_|
```

  If choosing one of the items is optional, the entire stack is below the main path.

```
>>__statement__ ________________ __><
               |_optional_choice1_|
               |_optional_choice2_|
```

- An arrow returning to the left above a line indicates an item that you can repeat.

```
               <________________
>>__statement____repeatable_item_|__><
```

  or

```
>>__statement__ ____________________ __><
               | <________________  |
               |___repeatable_item_|_|
```

  A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

- Keywords are in non-italic letters and should be entered exactly as

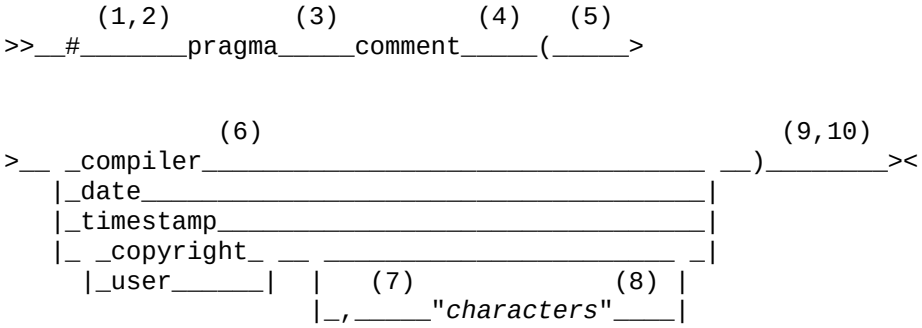
shown (for example, pragma). They must be spelled exactly as shown. Variables are in italics and lowercase letters (for example, *identifier*). They represent user-supplied names or values.

° Keywords that appear in mixed-case letters (for example, AGGregate) indicate that the keyword can be abbreviated (AGG) or entered in full (AGGREGATE).

° If punctuation marks, parentheses, arithmetic operators, or other non-alphanumeric characters are shown, you must enter them as part of the syntax.

Note: The white space is not always required between tokens but you should include at least one blank space between tokens unless otherwise specified.

The following syntax diagram example shows the syntax for the #pragma comment directive.



Notes:

(1) This is the start of the syntax diagram.

(2) The symbol # must appear first.

(3) The keyword pragma must follow the # symbol.

(4) The keyword comment must follow the keyword pragma.

(5) An opening parenthesis must follow the keyword comment.

(6) The comment type must be entered only as one of the following: compiler, date, timestamp, copyright, or user.

(7) If the comment type is copyright or user, and an optional character string is following, a comma must be present after the comment type.

(8) A character string must follow the comma. The character string must be enclosed in double quotation marks.

(9) A closing parenthesis is required.

(10) This is the end of the syntax diagram.

The following examples of the #pragma comment directive are syntactically correct according to the diagram shown above:

```
#pragma comment(date)
#pragma comment(user)
#pragma comment(copyright,"This text will appear in the module")
```

1.0 Chapter 1. Common Questions about Migration

This section describes the kind of migration impacts that you might encounter, and the possible solutions.

Subtopics:

1.1 Will Existing C/370 Applications Work with LE/VSE?

In order to run an existing C/370 application with LE/VSE, you must recompile the application with the C/VSE compiler and link-edit it with LE/VSE. A well-coded C/370 application should, in most cases, recompile successfully with C/VSE and run successfully with LE/VSE, without any modifications.

There are, however, a number of migration items (described in the following chapters) that might affect your application, and you should review these before attempting migration.

1.2 I Attempt to Recompile My Application and It Fails -- Why?

If your application previously compiled successfully with the C/370 compiler, but will not compile successfully with the C/VSE compiler, do the following:

1. Review the migration items listed in Chapter 2, "Source Program Compatibility" in topic 2.0 and make any suggested changes to your source code.

2. Review your source code to ensure it follows the ANSI standard of the C language, as described in the *C/VSE Language Reference*.

1.3 My Application Compiles but Does Not Run -- Now What?

If your application does not run, it might be due to either a migration problem, or to an error in your program that surfaces as a result of a new design feature in the run-time library. You should then take the following steps:

1. Verify the search order of your sublibraries.

   Check the search order of your sublibraries in your JCL LIBDEF statements to ensure that you did not accidentally compile with the wrong compiler, prelink with the wrong prelinker, or link-edit with the wrong run-time library:

   - The default installation sublibrary for the C/VSE compiler is PRD2.DBASE. In your LIBDEF PHASE statement, this sublibrary must be specified ahead of the sublibrary where the C/370 compiler is installed (usually PRD2.PROD) when you compile your application.

     Similarly, if any C/370 compiler phases have been placed in the shared virtual area (SVA), you must either remove these phases from the SVA or, by specifying the SDL keyword at the appropriate place in your LIBDEF PHASE statement, ensure that the SVA is searched after the sublibrary containing the C/VSE compiler.

   - The default installation sublibrary for LE/VSE is PRD2.SCEEBASE. In your LIBDEF PHASE and LIBDEF OBJ statements, this sublibrary must be specified ahead of the sublibrary where the C/370 library is installed (usually PRD2.PROD) when you prelink or link-edit your application.

     Similarly, if the C/370 prelinker phase has been placed in the SVA, you must either remove it from the SVA or, by specifying the SDL keyword at the appropriate place in your LIBDEF PHASE statement, ensure that the SVA is searched after the sublibrary containing LE/VSE.

2. Use environment variables to obtain the "old behavior."

   Under LE/VSE, you can use the ENVAR run-time option to specify the values of environment variables at execution time. Some environment

variables allow you to specify the "Old Behavior" for particular items. The following setting is recommended with the ENVAR option:

```
ENVAR("_EDC_COMPAT=32767")
```

The value assigned to _.EDC_COMPAT is used as a bit mask. A value of 32767 will tell the library to use "old behavior" for all of the general compatibility items currently defined by _EDC_COMPAT. For more information about _EDC_COMPAT and its possible values, refer to the *LE/VSE C Run-Time Programming Guide*.

If _EDC_COMPAT solves your migration problem, then you can use it with the ENVAR run-time option, as shown above, or in a call to setenv() either in the CEEBINT High-Level Language user exit or in your main() program. For more information about setenv(), see the *LE/VSE C Run-Time Library Reference* and the *LE/VSE C Run-Time Programming Guide*. For more information about the CEEBINT High-Level Language user exit, see the *LE/VSE Programming Guide*.

3. Review the migration items documented in this book.

   If you find a migration item in this manual that you think might affect your application, use the recommended workaround. If setting of an environment variable is not recommended, then a source change is required.

4. Look for undocumented interfaces.

   It is possible that your application has dependencies on undocumented interfaces. For example, you might have dependencies on library control blocks, specific errno values, or specific return values. Alter your code to use only documented interfaces.

5. Look for uninitialized storage.

   In some cases, applications will run with uninitialized storage, because the run-time library might inadvertently clear storage, or because the storage location referenced is set to zero.

   It is unlikely that an application will run with uninitialized storage. However, it is possible; for example, a NULL pointer can be dereferenced without error, if used read-only. You can verify that your application is coded correctly, by specifying the STORAGE run-time option and setting the storage to X'FE'. Any uninitialized pointers will fail at first reference instead of accidentally referencing storage locations at random.

6. Contact your service representative.

   If you have followed the advice listed in the steps above, but you still cannot run your application with LE/VSE, then contact your System Programmer to verify whether or not all service has been

applied to your system. Often, the problem you have encountered has already been reported to IBM, and a fix is available. If this is not the case, then ask your Service Representative to open a Problem Management Record (PMR) against the applicable IBM product. See the *C/VSE Diagnosis Guide* for information about how to do this.

2.0 Chapter 2. Source Program Compatibility

This chapter describes the changes you might have to make to your source code when moving applications from the C/370 compiler and library to the C/VSE compiler and LE/VSE. The information in this chapter will help you understand what changes, if any, you will have to make to your present source code to ensure it is compatible with LE/VSE.

Expanded capabilities of LE/VSE are not discussed here. For instance, LE/VSE honors all existing run-time options and also offers new run-time options. You can read about these in the *LE/VSE Programming Reference*.

Chapter 4, "Other Migration Considerations" in topic 4.0 has information on run-time options, which might also affect source code compatibility.

Subtopics:

2.1 Input and Output Operations

Programs running with the C/370 library might have to be changed if they have dependencies on any of the input and output behaviors listed in Chapter 3, "Input and Output Operations Compatibility" in topic 3.0.

2.2 The #pragma runopts Directive

With C/370, if you wanted to run a C application in the batch environment with DL/I, you needed to specify the following values for the ENV and PLIST run-time options using the #pragma runopts compiler directive:

```
ENV(IMS)
PLIST(IMS)
```

These values were used for source compatibility with C/370 in the MVS environment. With C/VSE and LE/VSE, these values are no longer supported. Instead, you must specify the following values:

```
ENV(DLI)
PLIST(OS)
```

For more information about the ENV and PLIST run-time options, see the *LE/VSE Programming Reference*.

2.3 Specifying Run-Time Options with the system() Function

In C/370, when passing only run-time options to a C/370 program called with the system() function, you did not have to end the options with a slash (/). With LE/VSE, you must end the options with a slash.

With LE/VSE, if you have no run-time options but the input arguments passed to main() contain a slash, you must prefix the arguments with a slash.

2.4 SIGFPE Exceptions

Decimal overflow conditions were masked in the C/370 library. With the introduction of the new packed decimal data type, these conditions are now enabled with LE/VSE. If any of your applications contain programs that accidentally generate decimal overflow conditions, the applications might behave differently with LE/VSE, by raising unexpected SIGFPE exceptions. Without source alteration, such applications cannot be migrated to LE/VSE, and are unsupported.

It is unlikely that such applications will be present in a C-only environment. These unexpected exceptions might occur in mixed-language applications, particularly those involving C and assembler code when the assembler code explicitly manipulates the program mask.

2.5 Program Mask Manipulations

Programs that depend on or explicitly manipulate the program mask might require source alteration to execute correctly with LE/VSE. The following are the only conditions that require this change. They also apply to Customer Information Control System (CICS) programs in the handling and management of the PSW mask.

° A C program containing assembler interlanguage calls, where the invoked code uses the S/370 decimal instructions that might generate an unmasked decimal overflow condition, requires modification for migration. There are two methods for migrating the code. The first one is preferred:

- Modify the assembler code to save the existing mask, set the new value, and when finished, restore the saved mask.

- Change the C code so that the produced SIGFPE signal is ignored in the called code. In the following example the signal() calls surround the overflow-producing code. The SIGFPE exception signal is ignored, and then reenabled:

```
signal(SIGFPE, SIG_IGN);   /* ignore exceptions */
...
callit();                  /* in called routine */
...
signal(SIGFPE, SIG_DFL);   /* restore default handling */
```

° A C program containing assembler interlanguage calls, where the invoked code explicitly alters the program mask, and does not explicitly save and restore it, also requires modification for migration.

If user code explicitly alters the state of the program mask, the value prior to modification must be saved, and the value restored to its former value after the modification. You must ensure that the decimal overflow program mask bit is enabled during the execution of C code. Failure to preserve the mask might result in unpredictable behavior.

2.6 The release() Function

In C/370, the release() function deleted phases retrieved by fetch() or control blocks created by fetchep(). With C/VSE and LE/VSE, you cannot issue a release() call for a fetched COBOL or PL/I phase. If you do, release() returns a nonzero return code. You can, however, use release() with C/VSE phases and non-LE/VSE enabled assembler phases (see below).

If your application fetches and releases COBOL phases (fetching of PL/I phases was not supported by C/370), you will need to change your source code.

Although release() could be issued for any assembler phases with C/370, it cannot be issued for LE/VSE-enabled assembler routines. These routines, known as ASM15 routines, are assembled with LE/VSE assembler prologs. (ASM15 routines are coded with CEEENTRY macro.) If any assembler routines are rewritten as ASM15 routines, make sure the calling code does not issue a release() call for them.

2.7 The realloc() Function

When the realloc() function is used with LE/VSE, a new area is always obtained and the data is copied. This is different from C/370, where, if the new size was equal to or less than the original size, the same area was used. This mechanism was not defined or documented.

You might want to make sure your source code has no dependencies on the mechanism of the old version of the realloc() function so that the code will be compatible with LE/VSE.

2.8 Preinitialization Interface

C/370 provided a preinitialization interface that you could use to preinitialize the C run-time environment for multiple invocations of a C program. The C run-time environment could be preinitialized by using an assembler program to call the C program with the C entry point, CEESTART, and pass the program a special Extended Parameter List. LE/VSE does not support this preinitialization interface. Instead, under LE/VSE, you must use the LE/VSE preinitialization interface, CEEPIPI. For more information about CEEPIPI, see the *LE/VSE Programming Guide*.

2.9 Default of the Invalid Escape Character

For C/370, the default character used in place of an invalid character in an escape sequence was the backslash (\). For C/VSE, it is the character following the backslash. For example, the default of an invalid escape sequence \w for C/370 was the backslash, while the default for C/VSE is w.

2.10 The #line Directive

The #line directive changes the line number and file name of output after it is processed. Debug Tool for VSE/ESA depends on the actual line numbers. To avoid problems with Debug Tool for VSE/ESA that would result

from the changed line numbers, the C/VSE compiler ignores #line directives when the TEST compile-time option is in effect.

Note: You should not use any #line directives before a #pragma options directive that contains the TEST option. If you do so, the compiler will be unable to catch it. The #line directive will not be ignored, and Debug Tool for VSE/ESA will not work with the output.

2.11 The __librel() Function

The __librel() library function is a System/370 extension to SAA C. It returns the release level of the library that your program is using. With C/VSE, the release level returned by __librel() contains a field for a number to represent the library product. LE/VSE is Product 1. The product number is specified in the top 4 bits that were used for part of the Version number in C/370. Programs that use the old format of the returned information could interpret release level for LE/VSE as Version 17 (1 * 16 + 1), Release 4, Modification Level 0 instead of Product 1, Version 1, Release 4, Modification Level 0.

You will have to modify programs that use the information returned from __librel().

The following program uses __librel() to print a string that indicates the library, version, release, and modification level.

```
/*--------------------------------------------------------------------*/
/*    librel()  will return an unsigned int                           */
/*    format    fVrrmmmm                                              */
/*    where     f    = 1 LE/VSE                                       */
/*                   = 0 C/370 Library                                */
/*              V    = version number                                 */
/*              rr   = release number                                 */
/*              mmmm = modification level                             */
/*--------------------------------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>


int main(int argc, char ** argv)
{
  unsigned int   ret_value;
  unsigned int   version  ;
  unsigned int   release  ;
  unsigned int   mod      ;
  char    *      product_name;


  ret_value = __librel();
  version = ret_value >> 24 & 0x0f ; /* 2nd nibble of the 1st byte */
  release = ret_value >> 16 & 0xff ; /* the second byte            */
  mod     = ret_value       & 0xffff; /* the last two bytes        */


  if ( (ret_value >> 28) > 0 )
```

```
      product_name = "LE/VSE";
   else
      product_name = "C/370 Library";


   printf("Library is: %s Version %d Release %d Modification "
          "Level %d\n", product_name , version , release, mod);
}
```

2.12 Messages

There are differences in diagnostic messages and return codes between the C/370 compiler and the C/VSE compiler. Message contents have changed, and return codes for some messages have changed (errors have become warnings, and the other way around). Any application that is affected by compiler message content or compiler return codes must be updated accordingly. Do not build dependencies on message content, message numbers, or return codes.

There are also differences in messages between the C/370 library and LE/VSE. Some run-time messages have been added and some have been deleted; the contents of others have been changed. Any application that is affected by the format or contents of these messages will have to be updated accordingly. Do not build dependencies on message content or message numbers.

Subtopics:

2.12.1 Prefix of perror() and strerror() Messages

All perror() and strerror() messages under LE/VSE contain a prefix (in C/370, there were no prefixes on these messages). The prefix is EDCxxxxa, where xxxx is a number (always 5xxx) and the a is either I, E, or S. See *LE/VSE Debugging Guide and Run-Time Messages* for a list of these messages.

2.13 Alternative Code Points

If your source code contains either of the following alternative code points, you must specify the NOLOCALE compile-time option when compiling with the C/VSE compiler:

- X'8B' as alternative code point for X'C0' (the left brace)

- X'9B' as alternative code point for X'D0' (the right brace)

3.0 Chapter 3. Input and Output Operations Compatibility

If your C applications use input/output (I/O) operations, you should read the changes listed in this section. When moving from the C/370 compiler and library to C/VSE and LE/VSE, you will generally be able to recompile and run "well-behaved" programs without source changes. Well-behaved programs do not rely on undocumented behavior, restrictions, or invalid behaviors of C/370. For example, if C/370 documentation specified that a return code was a negative value, and your code relies on that value being -3, your code is not well-behaved and is relying on undocumented behavior.

Subtopics:

3.1 Opening Files

The following changes have been made to file open processing:

- When you call the fopen() or freopen() library function, you can specify each mode string parameter only once. If you specify any parameter more than once, the function call fails. In C/370, you could specify more than one instance of a parameter.

- You cannot open a file more than once for a write operation. Although this was never officially supported, C/370 allowed you, in some cases, to open a file for write by its file ID and then again by its filename.

- In C/370, fopen() allowed spaces and commas as delimiters for mode string parameters. Only commas are allowed now.

3.2 Writing to Files

The following changes have been made to file write processing:

- Write operations to files opened in binary mode are no longer deferred. Previously, C/370 did not do the physical write of a block that held *nn* bytes until *nn*+1 bytes had been written to the block. Now, LE/VSE follows the rules for full buffering, described in the *LE/VSE C Run-Time Programming Guide*, and writes out data as soon as the block is full. The *nn* bytes are still written to the file; the only difference is in the timing of when it is done.

- The backspace character ('\b') is now placed into files as is. In C/370, the backspace character backed up the file position in some cases.

- For all text I/O, truncation for fwrite() is now handled the same way that it is handled for puts() and fputs(). If you write more data than a record can hold, and your output data contains any of the terminating control characters, '\n' or '\r' (or '\f', if you are using ASA), LE/VSE still truncates extra data. However, recognizing that the text line is complete, LE/VSE writes subsequent data to the next record boundary. Previously, fwrite() stopped immediately after C/370 began truncating data, so that you had to add a control character before writing any more data.

- The rules for changing fixed-format text records are now consistent. When you are updating any fixed-format, non-ASA text file, you can lengthen or shorten a logical record. The maximum length of an updated logical record is the length specified in the LRECL file attribute, plus the new-line character; the minimum length is a single new-line character.

  Writing new data into an existing record replaces the old data and, if the new data is longer or shorter than the old data, changes the size of the logical record by changing the number of blank characters in the physical record. Consider that you have the following record:

	logical record	physical record
Char	a b c \n	a b c
Hex	8 8 8 1 1 2 3 5	8 8 8 4 4 4 4 4 4 4 1 2 3 0 0 0 0 0 0 0

  If you update the record to contain a single new-line, this is what happens:

	logical record	physical record

```
    Char     \n

    Hex      1                         4 4 4 4 4 4 4 4 4 4
             5                         0 0 0 0 0 0 0 0 0 0
```

If you update the record to contain the maximum number of characters it can hold, this is what happens:

```
             logical record            physical record
             --------------            ---------------

    Char      1 2 3 4 5 6 7 8 9 0 \n    1 2 3 4 5 6 7 8 9 0

    Hex       F F F F F F F F F  1      F F F F F F F F F F
              1 2 3 4 5 6 7 8 9 0  5    1 2 3 4 5 6 7 8 9 0
```

When you extend a record, thereby writing over the old new-line, there is a new-line character implied after the new characters. Here is an example:

```
             logical record            physical record
             --------------            ---------------

    Char      a b c \n                  a b c

    Hex       8 8 8  1                  8 8 8 4 4 4 4 4 4 4
              1 2 3  5                  1 2 3 0 0 0 0 0 0 0
```

You can overwrite this record with the following:

```
             logical record            physical record
             --------------            ---------------

    Char      1 2 3 4 5                 physical record not changed yet

    Hex       F F F F F
              1 2 3 4 5
```

At this point, the position of the new-line character defaults to follow the 5. If you then call fflush(), this is what happens:

```
             logical record            physical record
             --------------            ---------------
```

```
Char     1 2 3 4 5 \n            1 2 3 4 5

Hex      F F F F F  1            F F F F F 4 4 4 4 4
         1 2 3 4 5  5            1 2 3 4 5 0 0 0 0 0
```

Note that calling fflush() has changed the contents of the physical record to be five characters followed by five blanks, and has placed a new-line character in the logical stream. If you now write to the file, LE/VSE continues to extend the record. If you read from the file, you will read the new-line and set the file position to the end of the current record. All subsequent read and write operations then go on to the next record. Also note that, in the past, the implicit new-line was not acknowledged.

- You can now update a record partially in a file opened with type=record. C/370 returned an error if you tried to make a partial update to a record. Now, a record is updated up to the number of characters you specify, and the remaining characters are untouched. The next update is to the next record.

- For files opened with type=record and recfm=F, incomplete new records are padded with null characters. C/370 returned an error if you tried to add a record with fewer characters than specified in the LRECL file attribute.

- In some cases, LE/VSE blocks files more efficiently than C/370. Applications that depend on the creation of short blocks might fail.

- The behavior of ASA files when you close them has changed. In C/370, this is what happened:

Written to File	**Read from File after** fclose(), fopen()
abc\n\n\n	abc\n\n\n\n
abc\n\n	abc\n\n\n
abc\n	abc\n

In LE/VSE, you read from the file what you wrote to it. For example:

Written to File	**Read from File after** fclose(), fopen()
abc\n\n\n	abc\n\n\n
abc\n\n	abc\n\n
abc\n	abc\n

In C/370, writing a single new-line to a new file created an empty file. Now, LE/VSE treats a single new-line written to a new file as a special case, because it is the last new-line of the file. LE/VSE writes a single blank to the file. When you read this file, you will see two new-line characters instead of one. You also see two new-line characters on a read if you have written two new-line characters to the file.

The behavior of appending to ASA files has also changed. The following table shows what you get from an ASA file when you:

1. Open an ASA file for write
2. Write abc
3. Close the file
4. Append xyz to the ASA file
5. Open the same ASA file for read

Write abc **to File,** fclose(), **then Append** xyz	**What You Read from File after** fclose(), fopen()	
	C/370	**LE/VSE**
abc ÿ xyz	\nabc\nxyz\n	same as C/370
abc ÿ \nxyz	\nabc\nxyz\n	\nabc\n\nxyz\n
abc ÿ \rxyz	\nabc\rxyz\n	\nabc\n\rxyz\n
abc\n ÿ xyz	\nabc\nxyz\n	same as C/370
abc\n ÿ \nxyz	\nabc\nxyz\n	\nabc\n\nxyz\n
abc\n ÿ \rxyz	\nabc\rxyz\n	\nabc\n\rxyz\n
abc\n\n ÿ xyz	\nabc\n\n\nxyz\n	\nabc\n\nxyz\n
abc\n\n ÿ \nxyz	\nabc\n\n\nxyz\n	same as C/370
abc\n\n ÿ \rxyz	\nabc\n\n\rxyz\n	same as C/370

° The behavior of DBCS strings has changed:

- I/O now checks the value of MB_CUR_MAX to determine whether to interpret DBCS characters within a file.

- When MB_CUR_MAX is 4, you can no longer place control characters in the middle of output DBCS strings for interpretation. Control characters within DBCS strings are treated as DBCS data. C/370 split the DBCS string at the '\n' (new-line) control character position by adding an SI (Shift In) control character at the new-line position, and then adding an SO (Shift Out) control character before the data following the new-line character. If MB_CUR_MAX is 1, LE/VSE interprets control characters within any string, but does not interpret DBCS strings. SO and SI characters

are treated as ordinary characters.

- When you are writing DBCS data to text files, if there are multiple SO (Shift Out) control-character write operations with no intervening SI (Shift In) control character, LE/VSE discards the SO characters and marks that a truncation error has occurred. C/370 allowed multiple SO control-character write operations with no intervening SI control character without issuing an error condition.

- When you are writing DBCS data to text files and specify an odd number of DBCS bytes before an SI control character, the last DBCS character is padded with a X'FE' byte. If a SIGIOERR handler exists, it is triggered. C/370 allowed incorrectly placed SI control-character write operations to complete without any indication of an error.

- Now, when an SO has been issued to indicate the beginning of a DBCS string within a text file, the DBCS must terminate within the record. The record will have both an SO and an SI.

3.3 Repositioning within Files

The following changes have been made to file reposition processing:

° The behavior of fgetpos(), fseek(), and fflush() following a call to ungetc() has changed. In C/370, fgetpos(), fseek(), and fflush() all ignored characters pushed back by ungetc(), and considered the file to be at the position where the first ungetc() character was pushed back. Also, ftell() acknowledged characters pushed back by ungetc() by backing up one position if there was a character pushed back. Now:

- The fgetpos() library function behaves just as ftell() does

- When a seek from the current position (SEEK_CUR) is performed, fseek() accounts for any ungetc() character before moving, using the user-supplied offset

- The fflush() library function moves the position back one character for every character that was pushed back

If you have applications that depend on the previous behavior of fgetpos(), fseek(), or fflush() you can use the new _EDC_COMPAT environment variable so that source code need not be changed to compensate for the new behavior. The _EDC_COMPAT environment variable is described in the *LE/VSE C Run-Time Programming Guide*.

° The ungetc() library function now supports up to four push-back characters instead of one. Code that relies on failing after one

ungetc() character is pushed back will not work.

- For I/O to and from files opened in text mode, the ftell() encoding system now supports higher blocking factors for smaller block sizes. In general, you should not rely on ftell() values generated by code you developed using C/370. You can try ftell() values taken in C/370 for files opened in text or binary format if you set the environment variable _EDC_COMPAT before you call fopen() or freopen(). Do not rely on ftell() values saved across program boundaries. The _EDC_COMPAT environment variable is described in the *LE/VSE C Run-Time Programming Guide*.

- For record I/O, ftell() now returns the relative record number instead of an encoded offset from the beginning of the file. You can supply the relative record number without acquiring it from ftell(). You cannot use old ftell() values for record I/O, regardless of the setting of _EDC_COMPAT. The _EDC_COMPAT environment variable is described in the *LE/VSE C Run-Time Programming Guide*.

- For files that are not fixed-format or are opened in text mode, fseek() now supports nonzero relative byte offsets from SEEK_CUR and SEEK_END. In C/370, you could only use SEEK_CUR or SEEK_END with an offset of 0, unless you had opened a fixed-format file in binary mode. For record I/O, this new fseek() support means that you are using relative record offsets.

- The fseek() and fsetpos() library functions are now supported for ASA and spanned files. In C/370, calls to fseek() and fsetpos() failed for these files.

- If you have used ungetc() to move the file pointer to a position before the beginning of the file, calls to ftell() and fgetpos() now fail. Previously, ftell() returned the value 0 for such calls, yet set errno to a non-zero value. fgetpos() did not previously account for ungetc() calls. See the beginning of this section on how to change fgetpos() behavior using _EDC_COMPAT.

  For example, suppose that you are at relative position 1 in the file and ungetc() is performed twice. ftell() and fgetpos() will now report the relative position -1, which is before the start of the file, causing both ftell() and fgetpos() to fail.

- Once you have called ftell(), calls to setbuf() or setvbuf() might fail. Applications should never call I/O functions between calls to fopen() or freopen() and calls to the functions that control buffering.

3.4 Closing Files

The behavior of ASA files when you close them is now consistent. When closing VS/VBS binary files, a zero-length last segment is written to the end of the file if both of the following conditions are true:

- A first segment has been written to the file
- There is no more data to write to the file

Written to File	Physical Record after Close					
	Record	C/370		LE/VSE		
		Char	Hex	Char	Hex	
abc	1	abc	4888 0123	same as C/370		
abc\n	1	abc	4888 0123	same as C/370		
abc\n\n	1	abc	4888 0123	abc	4888 0123	
	2	0	F 0		4 0	
abc\n\n\n	1	abc	4888 0123	abc	4888 0123	
	2	-	6 0		4 0	
abc\r	1	abc	4888 0123	same as C/370		
	2	+	4 E	same as C/370		
abc\r	1	abc	4888 0123	same as C/370		
	2	1	F 1	same as C/370		

3.5 fldata() Return Values

There are minor changes to the values that the fldata() library function returns. It now returns the full file name in the new field, __dsname. It might also return more specific information in other fields.

For more information on fldata(), see the *LE/VSE C Run-Time Programming Guide*.

3.6 Error Handling

The general return code for errors is now EOF. In C/370, some I/O functions returned 1 as an error code to indicate failure. This caused some confusion, as 1 is a possible errno value as well as a return code. EOF is not a valid errno value.

Programs that rely on specific values of errno might not run as expected, because certain errno values have changed. Refer to the *LE/VSE C Run-Time Programming Guide* for a list of the current errno values.

3.7 Miscellaneous

The following miscellaneous changes have been made:

- The inheritance model for standard streams now supports repositioning. In C/370, if you opened stdout or stderr in update mode, and then called another C program by using the system() function, the program that you called inherited the standard streams, but moved the file position for stdout or stderr to the end of the file. Now, LE/VSE does not move the file position to the end of the file. For text files, the position is moved only to the nearest record boundary at or following the current position. This is consistent with the way stdin behaves for text files.

- The values for L_tmpnam and FILENAME_MAX have been changed.

Constant	**Old Value**	**New Value**
L_tmpnam	47	1024
FILENAME_MAX	57	1024

- The names produced by the tmpnam() library function are different in LE/VSE. Any code that depends on the internal structure of these names might fail.

3.8 VSAM I/O Changes

The following changes have been made to VSAM processing:

- LE/VSE does not append an index key when you read from an RRDS file

opened in text or binary mode.

- RRDS files opened in text or binary mode no longer support setting the access direction to BWD.

4.0 Chapter 4. Other Migration Considerations

This chapter provides additional considerations on migrating applications from the C/370 Compiler and Library to the C/VSE compiler and LE/VSE.

Subtopics:

4.1 Changes That Affect User JCL

The following sections describe changes that might affect your JCL.

Subtopics:

4.1.1 Return Codes

Return codes have been changed in some circumstances, and JCL that is affected by them must be changed accordingly (or else the CEEBXITA exit must be customized to emulate the old return codes). C/370 return codes were from 0 to 999. This is no longer true; C/VSE now uses the LE/VSE return codes which have a different range. The LE/VSE return codes are documented in *LE/VSE Debugging Guide and Run-Time Messages*. The return

code for an abort is now 2000; it was 1000. The return code for an unhandled SIGFPE, SIGILL, or SIGSEGV is now 3000; it was 2000.

4.1.2 Differences in Standard Streams

C run-time messages and perror() messages are directed to the stderr standard stream output device. Under LE/VSE, the default destination for stderr output is the MSGFILE filename. In the batch environment, you can control the destination of stderr output by using the LE/VSE MSGFILE run-time option. The default filename for MSGFILE in batch is SYSLST. In the CICS environment, the MSGFILE run-time option is ignored, and run-time output is directed to the CESE transient data queue.

For more information about the MSGFILE run-time option, see the *LE/VSE Programming Guide*.

4.1.3 Specifying Run-Time Options on the EXEC Statement

In C/370, when passing only run-time options to a C/370 program in the PARM parameter of the JCL EXEC statement, you did not have to end the options with a slash (/). With LE/VSE, you must end the options with a slash.

With LE/VSE, if you have no run-time options but the input arguments passed to main() contain a slash, you must prefix the arguments with a slash.

JCL that is affected by the slash must be changed accordingly.

4.1.4 Passing Command-Line Parameters to a Program

In C/370, if an error was detected with the parameters being passed to the main program, the program terminated with a return code of 8 and a message indicating the reason the program was not run. For example, if there was a redirection error in the parameters, the message would indicate that the program had terminated because of a redirection error.

Under LE/VSE, the same message will be displayed, but the program will also terminate with a 4093 abend, reason code 52 (hexadecimal 34). For more information about the abend codes and messages, see *LE/VSE Debugging Guide and Run-Time Messages*.

4.2 Run-Time Options

The following sections describe changes to run-time option processing that might affect the way your applications behave.

Subtopics:

4.2.1 Compatibility Mapping

The following table shows which C/370 run-time options are mapped to LE/VSE run-time options for compatibility. Although the C/370 options are mapped to their LE/VSE equivalents, if you do not change them in your #pragma runopts directives, you will get an informational message at compile time. If you do not change them in your run-time JCL, you will get an informational message at run time.

For more information about the LE/VSE run-time options, see the *LE/VSE Programming Reference*.

Figure 3. C/370 and LE/VSE Run-Time Option Mapping

C/370 Option	**LE/VSE Equivalent**	**Notes**
ISAINC(*incr_size*)	STACK(,*incr_size*)	The C/370 ISAINC run-time option is mapped to the LE/VSE STACK run-time option for compatibility. It affects all languages in the enclave.
ISASIZE(*init_size*)	STACK(*init_size*)	The C/370 ISASIZE run-time option is mapped to the LE/VSE STACK run-time option for compatibility. It affects all languages in the enclave.
LANGUAGE	NATLANG	The C/370 LANGUAGE run-time option is mapped to the LE/VSE NATLANG run-time option for compatibility. It affects all languages in the enclave.

REPORT	RPTSTG(ON)	The C/370 REPORT run-time option is mapped to the LE/VSE RPTSTG(ON) run-time option for compatibility. It affects all languages in the enclave.
NOREPORT	RPTSTG(OFF)	The C/370 NOREPORT run-time option is mapped to the LE/VSE RPTSTG(OFF) run-time option for compatibility. It affects all languages in the enclave.
SPIE	TRAP(ON)	The C/370 SPIE run-time option is mapped to the LE/VSE TRAP(ON) run-time option for compatibility. It affects all languages in the enclave. LE/VSE requires TRAP(ON) to be in effect for applications to run successfully. For information about how mapping of the SPIE run-time option is affected by other run-time options, see Figure 4.
NOSPIE	TRAP(OFF)	The C/370 NOSPIE run-time option is mapped to the LE/VSE TRAP(OFF) run-time option for compatibility. It affects all languages in the enclave. For information about how mapping of the NOSPIE run-time option is affected by other run-time options, see Figure 4.
STAE	TRAP(ON)	The C/370 STAE run-time option is mapped to the LE/VSE TRAP(ON) run-time option for compatibility. It affects all languages in the enclave. LE/VSE requires TRAP(ON) to be in effect for

		applications to run successfully. For information about how mapping of the STAE run-time option is affected by other run-time options, see Figure 4.
NOSTAE	TRAP(OFF)	The C/370 NOSTAE run-time option is mapped to the LE/VSE TRAP(OFF) run-time option for compatibility. It affects all languages in the enclave. For information about how mapping of the NOSTAE run-time option is affected by other run-time options, see Figure 4.

As shown in Figure 3, the C/370 SPIE, NOSPIE, STAE, and NOSTAE run-time options map to the LE/VSE TRAP run-time option.Figure 4 describes how these options are mapped when specified in combination with each other.

Figure 4. TRAP Run-Time Option Settings

If...	**then...**
a single instance of SPIE|NOSPIE or STAE|NOSTAE is specified in input,	TRAP is set according to that option, TRAP(ON) for SPIE or STAE, TRAP(OFF) for NOSPIE or NOSTAE.
both SPIE|NOSPIE and STAE|NOSTAE are specified in input,	TRAP is set ON, unless both NOSPIE and NOSTAE are specified, then TRAP is set to OFF.
STAE|NOSTAE is specified in one #pragma runopts statement, and SPIE|NOSPIE in another,	the option in the last #pragma runopts statement determines the setting of TRAP.
multiple instances of STAE|NOSTAE are specified,	TRAP is set according to the last instance only. All others are ignored.
multiple instances of SPIE|NOSPIE are specified,	TRAP is set according to the last instance only. All others are ignored.
an options string has TRAP(ON) or TRAP(OFF) together with SPIE|NOSPIE and/or STAE|NOSTAE,	the TRAP setting takes preference over all others.

4.2.2 HEAP Option

The following changes have been made to the HEAP run-time option:

- The default values for the initial size and increment size of the HEAP run-time option have changed. If you do not change or override the default values, heap storage will be allocated differently when running under LE/VSE. The defaults in C/370 were 4K initial size and 4K increment size. The defaults under LE/VSE are 32K initial size and 32K increment size.

- In C/370, heap storage was always allocated below the 16MB line, regardless of whether the ANYWHERE or BELOW keyword parameter was specified in the HEAP run-time option. In LE/VSE, the ANYWHERE or BELOW keyword is honored.

  Two new parameters have been added: initsz24 and incrsz24. They specify how much of the heap is allocated and incremented below the 16MB line.

- The parameters of the LE/VSE HEAP run-time option are all positional. In C/370, only the first two of the four parameters of the HEAP run-time option were positional; the keyword parameters could be specified without leading commas if the first two parameters were omitted. For example, to specify only the keyword KEEP, you could code HEAP(KEEP). Now, to specify only KEEP, you must code HEAP(,,,KEEP).

4.2.3 STACK Option

The following changes have been made to the STACK run-time option:

- The IBM-supplied default values for the initial size and increment size parameters of the STACK run-time option have changed. If you do not change or override the default values, stack storage will be allocated differently when your program is running under LE/VSE. The defaults in C/370 were 4K initial size and 0K increment size. The defaults under LE/VSE are 128K initial size and 128K increment size.

- In C/370, stack storage was always allocated below the 16MB line, regardless of whether the ANYWHERE or BELOW keyword parameter was specified in the STACK run-time option. In LE/VSE, the ANYWHERE or BELOW keyword is honored.

- A fourth parameter has been added to the STACK run-time option, the KEEP or FREE keyword parameter. This parameter specifies whether stack storage is kept or released after the last of the storage in the stack is freed.

- All parameters of the LE/VSE STACK run-time option are positional. In C/370, only the first two of the three parameters of the STACK run-time option were positional; the keyword parameter could be specified without leading commas if the first two parameters were omitted. For example, to specify only the keyword BELOW, you could code STACK(BELOW). Now, to specify only BELOW, you must code STACK(,,,BELOW).

4.2.4 TEST Option

In C/370 in the VSE environment, the TEST run-time option was accepted for compatibility with C/370 in the MVS environment, but was ignored. In LE/VSE, the TEST run-time option is used to invoke Debug Tool for VSE/ESA. For information about using Debug Tool for VSE/ESA to help debug your applications, see the *Debug Tool for VSE/ESA User's Guide and Reference*.

4.3 Compile-Time Options

The following sections describe changes made to compile-time options that might affect the migration of your applications. For more information about C/VSE compile-time options, see the *C/VSE User's Guide*.

Subtopics:

4.3.1 SPILL() Option

In the C/370 compiler, the NOSPILL compile-time option sets the spill area to zero. In the C/VSE compiler, NOSPILL sets the spill area to the default.

4.3.2 START Option

In the C/370 compiler in the VSE environment, the START compile-time option was accepted for compatibility with C/370 compiler in the MVS environment, but the generated object module could not be link-edited and run under VSE. NOSTART was the recommended compile-time option for VSE.

In the C/VSE compiler, the START compile-time option is the default, and is required for the compiler to generate an object module that can be link-edited and run with LE/VSE. The NOSTART compile-time option is not supported.

4.3.3 TARGET() Option

In the C/370 compiler in the VSE environment, the TARGET(TPF) compile-time option was supported for compatibility with C/370 in the MVS environment. In the C/VSE compiler, TARGET(TPF) is not supported.

For more information on compiler options, refer to the *C/VSE User's Guide*.

4.4 Locale Support

With the C/VSE compiler and LE/VSE, the support for locales has been substantially enhanced. Support for old style locales is continued for compatibility. At application run time, if old-style locales are explicitly requested by name, for example in a setlocale() reference, the old-style locale information is converted into new-style POSIX locale information. You should make use of the new locale facilities, and phase out use of old-style locale support.

Effect of setlocale() on LE/VSE: The current locale set with the setlocale() function affects some C library functions only. It does not affect the CEE functions under LE/VSE.

The enhanced locale facilities include support for different character encoding schemes, or code pages, across EBCDIC systems. It is recommended that code page IBM-1047 be used for writing source files, and for locales used at run time, especially when migrating existing applications that use locale-based facilities.

Applications might require redesign and source changes to fully use the new support for code pages other than IBM-1047. For instance:

- Assumptions about character mappings to hexadecimal values might not be valid if locales based on other code pages are used at execution time.

- Library functions, such as printf() or scanf(), might behave differently because they expect data to be in the codepage of the locale chosen at execution time.

There are special migration considerations when you are changing to code page 1047. The ¬ (logical not) glyph at code point X'5F' has been replaced with a ^ (caret), which is the correct glyph for the C character set. The ¬ glyph is at code point X'B0' in code page 1047. C/VSE does not support the ¬ glyph in code page 1047. That is, X'B0' will not be considered as an alternate code point to X'5F'.

All code points that were supported in C/370 compiler will continue to be supported while compiling with NOLOCALE. The European left and right braces will not be supported when compiling with the LOCALE option.

Refer to the *LE/VSE C Run-Time Programming Guide* for detailed information on new locale support.

4.5 SIGTERM, SIGINT, SIGUSR1, and SIGUSR2 Exceptions

In C/370, the default behavior for SIGINT, SIGUSR1, and SIGUSR2 exceptions was to ignore the signals. In LE/VSE, the default is to terminate the program and return a return code of 3000. For SIGTERM, the default has always been to terminate the program, but the return code is now 3000, whereas before it was 0.

The following table lists C conditions, condition types (such as operation exception or data exception), and default run-time messages.

Figure 5. Hardware and Software Exceptions and SIG_DFL Actions

C Condition	Origin	Default Run-Time Message	SIG_DFL Action
SIGILL	Operation exception Privileged operation Execute exception raise(SIGILL)	CEE3201 CEE3202 CEE3203 EDC6001	Abnormal termination (return code=3000)
SIGSEGV	Protection exception Addressing exception Specification exception raise(SIGSEGV)	CEE3204 CEE3205 CEE3206 EDC6002	Abnormal termination (return code=3000)
SIGFPE	Data exception Fixed point divide Decimal overflow Decimal divide Exponent overflow Floating point divide raise(SIGFPE)	CEE3207 CEE3209 CEE3210 CEE3211 CEE3212 CEE3212 EDC6000	Abnormal termination (return code=3000)
SIGABND	raise(SIGABND)	EDC6003	Abnormal termination (return code=3000)
SIGTERM	raise(SIGTERM)	EDC6004	Abnormal termination

			(return code=3000)
SIGINT	raise(SIGINT)	EDC6005	Abnormal termination (return code=3000)
SIGABRT	abort() raise(SIGABRT)	EDC6006	Abnormal termination (return code=2000)
SIGUSR1	raise(SIGUSR1)	EDC6007	Abnormal termination (return code=3000)
SIGUSR2	raise(SIGUSR2)	EDC6008	Abnormal termination (return code=3000)
SIGIOERR	raise(SIGIOERR)	EDC6009	Abnormal termination (return code=3000)

4.6 atexit List during Abnormal Termination

Unlike C/370, applications that terminate abnormally, including applications terminated by an abort() call, do not drive the atexit list.

4.7 CICS

The following sections describe changes in the behavior of C applications in the CICS environment that might affect the migration of your applications.

Subtopics:

4.7.1 Running CICS in 370 Mode

When running CICS with VSE/ESA 1.4 in 370 mode, the maximum size of a program that CICS can load in the dynamic storage area (DSA) is 512KB. The LE/VSE C event handler, CEEEV003, is larger than 512KB. Therefore, if you run CICS C applications in 370 mode, you must place CEEEV003 in the SVA. If you do not do so, LE/VSE will not initialise under CICS in 370 mode.

In addition, if you plan to use Debug Tool for VSE/ESA under CICS in 370 mode, you must also place the LE/VSE C I/O extensions module, EDCZ24, in the SVA.

4.7.2 CICS Abend Codes and Messages

Abend codes used by C/370 under CICS, such as ACC2, are no longer issued. An equivalent LE/VSE abend code, of the format 4nnn, is issued instead.

4.7.3 CICS Reason Codes

Reason codes that appeared in the CICS message console log have been changed. The new ones are documented in *LE/VSE Debugging Guide and Run-Time Messages*.

4.7.4 Standard Stream Support

Under CICS, records sent to the transient data queues associated with stdout and stderr with default settings take the form of a message as follows:

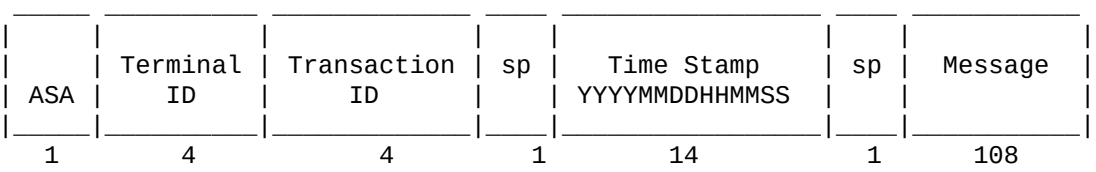


Figure 6. Format of data written to a CICS Data Queue.

ASA The American National Standard Code for Information Interchange (ASCII) carriage-control character.

Terminal ID
A 4-character terminal identifier.

Transaction ID
A 4-character transaction identifier.

sp A space.

Timestamp
The date and time displayed in the format YYYYMMDDHHMMSS.

Message
The data output to the standard stream stdout or stderr.

4.7.5 stderr Output

Output from stderr is sent to the CESE transient data queue. CESE is also used by LE/VSE for run-time error messages, dumps, and storage reports. If you previously used this file exclusively for C/370 stderr output, you should note that the output will be different. You should make changes to code that is affected by it.

4.7.6 Transient Data Queue Names

Transient data queue names are mapped as follows under LE/VSE:

Old Name	New Name
CCSI	CESI
CCSO	CESO
CCSE	CESE

4.8 Changes to User Output

The following sections describe changes to user output that might affect the migration of your applications.

Subtopics:

4.8.1 Message Contents

Because message contents and prefixes have changed, JCL that is affected by them must be changed accordingly. The prefixes on perror() and strerror() have changed. See "Prefix of perror() and strerror() Messages" in topic 2.12.1 for examples.

4.8.2 Dump Content

The contents of the dumps produced by LE/VSE are different from those of the dumps produced by C/370. The description of the contents of the dumps is in *LE/VSE Debugging Guide and Run-Time Messages*.

4.8.3 Storage Report

The format of the run-time storage report generated by the LE/VSE RPTSTG run-time option is different from the format of the storage reports produced by the C/370 REPORT run-time option. For more information about the RPTSTG run-time option, see the *LE/VSE Programming Reference*.

4.9 Working with Cross System Product

As in C/370, control can be passed between Cross System Product (CSP) and an LE/VSE-enabled program in three ways: XFER, DXFR, and CALL. You can pass control from CSP to an LE/VSE-enabled program, and then pass control back to CSP. However, when running under LE/VSE, if you pass control from an LE/VSE-enabled program to CSP, you cannot then pass control to another LE/VSE-enabled program from that CSP application. LE/VSE must appear only once in the chain of calls.

4.10 Compiler Listings

Compiler listings have changed between the C/370 compiler and the C/VSE compiler. For information about listings for the C/VSE compiler, see the *C/VSE User's Guide*. Do not build dependencies on the structure or content of listings.

BIBLIOGRAPHY Bibliography

Subtopics:

BIBLIOGRAPHY.1 IBM C for VSE/ESA Publications

Licensed Program Specifications, GC09-2421

Installation and Customization Guide, GC09-2422

Migration Guide, SC09-2423

User's Guide, SC09-2424

Language Reference, SC09-2425

Diagnosis Guide, GC09-2426

BIBLIOGRAPHY.2 IBM Language Environment for VSE/ESA Publications

Fact Sheet, GC33-6679

Concepts Guide, GC33-6680

Debugging Guide and Run-Time Messages, SC33-6681

Installation and Customization Guide, SC33-6682

Licensed Program Specifications, GC33-6683

Programming Guide, SC33-6684

Programming Reference, SC33-6685

Run-Time Migration Guide, SC33-6687

Writing Interlanguage Communication Applications, SC33-6686

C Run-Time Programming Guide, SC33-6688

C Run-Time Library Reference, SC33-6689

BIBLIOGRAPHY.3 Related Publications

Debug Tool for VSE/ESA

User's Guide and Reference, SC26-8797

Installation and Customization Guide, SC26-8798

Fact Sheet, GC26-8925

BIBLIOGRAPHY.4 Softcopy Publications

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

VSE Collection, SK2T-0060

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

INDEX Index

Special Characters

A

C

D

E

F

H

I

J

L

M

P

R

S

T

U

X

XFER to pass control to CSP, 4.9

BACK_1 Communicating Your Comments to IBM

IBM C for VSE/ESA
Migration Guide
Release 1

Publication No. SC09-2423-00

If there is something you like--or dislike--about this book, please let us know. You can use one of the methods listed below to send your comments to IBM. If you want a reply, include your name, address, and telephone number. If you are communicating electronically, include the book title, publication number, page number, or topic you are commenting on.

The comments you send should only pertain to the information in this book and its presentation. To request additional publications or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give it to the local IBM branch office or IBM representative for postage-paid mailing.

° If you prefer to send comments by mail, use the RCF at the back of this book.

° If you prefer to send comments by FAX, use this number:

- United States and Canada: 416-448-6161

- Other countries: (+1)-416-448-6161

° If you prefer to send comments electronically, use the network ID listed below. Be sure to include your entire network address if you wish a reply.

- Internet: torrcf@vnet.ibm.com
- IBMLink: toribm(torrcf)
- IBM/PROFS: torolab4(torrcf)
- IBMMAIL: ibmmail(caibmwt9)

COMMENTS Readers' Comments -- We'd Like to Hear from You

IBM C for VSE/ESA
Migration Guide
Release 1

Publication No. SC09-2423-00

Overall, how satisfied are you with the information in this book?

Legend:

1 Very satisfied
2 Satisfied
3 Neutral
4 Dissatisfied
5 Very dissatisfied

	1	2	3	4	5
Overall satisfaction					

How satisfied are you that the information in this book is:

	1	2	3	4	5
Accurate					
Complete					
Easy to find					
Easy to understand					

Well organized					
Applicable to your tasks					

Please tell us how we can improve this book:

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 EGLINTON AVENUE EAST
NORTH YORK ONTARIO CANADA M3C 1H7

Name . . . . . _____
Company or Organization _____
Address . . . . . _____
_____
_____
Phone No. . . . . _____

IBM Library Server Print Preview

DOCNUM = SC09-2423-00
DATETIME = 10/09/96 08:33:53
BLDVERS = 1.2
TITLE = C/VSE V1R1 Migration Guide
AUTHOR =

PATH = /home/webapps/epubs/htdocs/book
