IBM COBOL for VSE/ESA

General Information

Release 1

GC26-8068-00

IBM COBOL for VSE/ESA

General Information

Release 1

GC26-8068-00

Note!

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

First Edition (April 1995)

This edition applies to Version 1 Release 1 of IBM COBOL for VSE/ESA, Program Number 5686-068, and to all subsequent releases and modifications until otherwise indicated in new editions. Changes are made periodically to this publication; consult the latest *IBM System/390, 370, 30xx, 4300, and 9370 Processors Bibliography* for current information on this product.

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

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

IBM Corporation, Department J58
P.O. Box 49023
San Jose, California, 95161-9023
U.S.A.

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

Notices

References in this publication to IBM* products or services do not imply that they will be available everywhere IBM operates, nor that only IBM products or services can be used. Functionally equivalent products or services that do not infringe legal rights held by IBM can be used instead. Operation with products or services other than those expressly designated by IBM is your responsibility.

IBM may have patents or pending patent applications covering subject matter described herein. This document neither grants nor implies any license or immunity under any IBM or third-party patents, patent applications, trademarks, copyrights, or other similar rights, or any right to refer to IBM in any marketing activities. Other than responsibilities assumed via the Agreement for Purchase of IBM Machines and the Agreement for IBM Licensed Programs, IBM assumes no responsibility for any infringement of third-party rights that may result from use of the subject matter disclosed in this publication or from the manufacture, use, lease, or sale of machines or programs described herein.

Licenses under utility patents held by IBM are available on reasonable and nondiscriminatory terms. IBM does not grant licenses under its appearance design patents. Direct licensing inquiries in writing to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, U.S.A..

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. **This disclaimer does not apply in the United Kingdom or elsewhere if inconsistent with local law.**

Programming Interface

This book is intended to help you evaluate and plan for IBM COBOL for VSE/ESA (COBOL/VSE).

This book also documents General-Use Programming Interface and Associated Guidance Information.

General-Use programming interfaces allow the customer to write programs that obtain the services of COBOL/VSE.

General-Use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

┌──────── General-Use Programming Interface ────────┐

General-Use Programming Interface and Associated Guidance Information...

└──────── End of General-Use Programming Interface ────────┘

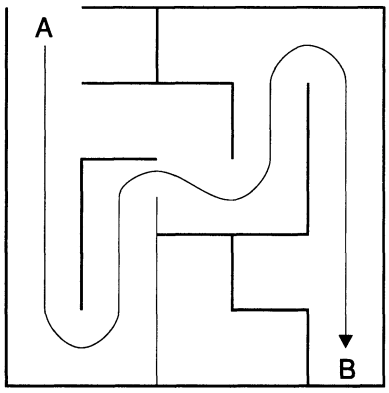
Trademarks

The following terms, denoted by an asterisk (*) on their first occurrences in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

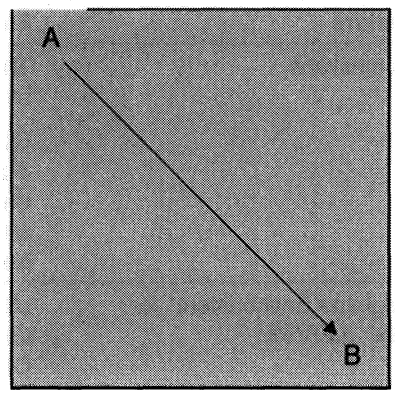
BookManager
CICS/VSE
Language Environment
Operating System/400
OS/400
Systems Application Architecture
SQL/DS
CICS
IBM
Operating System/2
OS/2
System/370
SAA
VSE/ESA

Why COBOL/VSE?

COBOL/VSE helps you ***streamline*** your application development business.



Existing Application Development



Application Development with COBOL/VSE

But, what is COBOL/VSE?

What It Is

COBOL/VSE is an implementation of the COBOL language—the most widely used application programming language in the world today. COBOL/VSE presents the language at its best. The full-function language and compiler make your applications much easier to develop and maintain, and portable to many different operating environments.

This increase in application effectiveness can save you time and money, two of the most important assets of any business.

What It Is Not

COBOL/VSE is not a revolution; you can move toward it with evolutionary steps. Most of your existing DOS/VS COBOL and VS COBOL II programs continue to run without change while you develop new applications with COBOL/VSE. In addition, you can selectively update existing applications to take advantage of advanced COBOL/VSE functions. Migration tools supplied with the product help pave the way for a smooth transition to COBOL/VSE.

COBOL/VSE—A Composite of New and Existing Function

COBOL/VSE adds new function, but it also builds on function introduced by DOS/VS COBOL and VS COBOL II.

New Function

New features of COBOL/VSE include support for the functions listed in Table 1.

Table 1. New Features Included in COBOL/VSE Support

New Feature	Feature Summary
Intrinsic functions	The intrinsic function support provides mathematical, statistical, financial, string handling, and date and time functions. COBOL/VSE supports the Intrinsic Function Module of the *COBOL 85 Standard*. Note that this book uses the term COBOL 85 Standard to refer to the International Standards Organization (ISO) and American National Standards Institute (ANSI) standards listed in Appendix A, "Industry Standards" on page 47.
IBM Language Environment* for VSE/ESA* (LE/VSE)	LE/VSE is a prerequisite product for COBOL/VSE. It provides a common run-time environment for all conforming high-level languages, including COBOL/VSE. LE/VSE adds improved interlanguage communication (ILC), a common set of callable services and tuning capability at execution time. Note that the LE/VSE product is required in order to run COBOL/VSE programs.
Procedure-pointer support	COBOL/VSE adds support for a new procedure-pointer data type. A procedure-pointer data item can be set to the address of a procedure entry point and can be passed as an argument to LE/VSE callable services or other programs. The procedure-pointer support enables you to use COBOL programs as condition handlers for such conditions as program checks, abends, or software generated signals.
GLOBAL support in Linkage section	Linkage section data items can now be declared GLOBAL, which means that a single declaration makes the name in one program available to all its subprograms.
Reference modification for double-byte character set (DBCS)	Reference modification, which enables you to refer to substrings of data, can now be applied to DBCS items.
General-use programming interface to compilation data	The ADATA compiler option directs the compiler to collect Associated Data information and write ADATA records to the SYSADAT file. The information expands on, and increases, the data that is available to you via the compiler listing. The SYSADAT file now provides a general-use programming interface for access to compilation data.
Displayable softcopy publications	COBOL/VSE offers the capability to use softcopy versions of *COBOL/VSE Language Reference*, *COBOL/VSE Programming Guide*, and *COBOL/VSE Migration Guide*. The softcopy information for COBOL/VSE is available as BookManager*/Read files.

Existing Function

COBOL/VSE includes many features that were first introduced by DOS/VS COBOL and VS COBOL II. For example, the ability to automatically check for valid data ranges, first introduced by DOS/VS COBOL, is retained by COBOL/VSE's support of LE/VSE.

Other features, first introduced by VS COBOL II and available when you develop and run COBOL/VSE programs, include support for the functions listed in Table 2 on page 3.

Table 2. Existing VS COBOL II Features Included in COBOL/VSE Support

Existing Feature	Feature Summary
Industry standards	The industry standard support provided by VS COBOL II is maintained, and extended, by COBOL/VSE. For a complete list of the industry standards met by COBOL/VSE, including the new intrinsic function support, see Appendix A, "Industry Standards" on page 47.
Structured programming	COBOL/VSE supports nested programs, in-line PERFORM statements, explicit scope terminators, and the EVALUATE statement. These constructs simplify the development of applications that conform to top-down design, modular program development, and structured programming concepts.
SAA* flagging	COBOL/VSE can flag language constructs to assist in producing programs that conform to SAA conventions for supporting multiple environments. Note that flagging is available only for SAA COBOL Level 1.
Improved CICS* COBOL interface	The CICS COBOL interface eliminates the need for a number of tasks including coding BLL cell addressing. Several COBOL language constructs are supported in the CICS environment, including the GOBACK statement and the ADDRESS OF special register.
Floating point	Floating point can be used for calculating values of mathematical expressions, including exponentiation.
Double-byte character support (DBCS)	Double-byte characters are supported for both application data and for characters in COBOL program user names.
Flexible initialization of values	When initializing data items, a single INITIALIZE statement can be used to replace multiple COBOL MOVE statements.
Improved file handling	File handling support includes using variable-length records, new file status codes, and external files.
Improved string handling	The addition of reference modification, which provides a substring function, makes it easier to process character strings.
Variety of compiler listing options	You can include any or all of the following in a compiler listing: source listing, cross-reference information, Data Division map, nested program map, and a listing of diagnostic messages.
Automatic checking for valid data ranges	Values of subscripts and indexes can be checked before data is inappropriately stored.
Code optimization	The OPTIMIZE compiler option causes the compiler to generate more efficient code.
Flexible numeric-sign processing	Numeric sign processing can be optimized for high-performance programs, and data with nonconforming signs can be processed.
Improved performance for SORT/MERGE	The FASTSRT compiler option results in faster sorting because the movement of data is reduced.
Support for reentrancy	COBOL/VSE programs can be reentrant; that is, a single copy of a program can satisfy several, simultaneous requests.
Support for extended architecture	COBOL/VSE programs and their data can reside either above or below the 16-megabyte line to take advantage of 31-bit addressing.

The Entire Product and this Book

This book describes the COBOL/VSE product as a whole, including the functions first introduced by DOS/VS COBOL and VS COBOL II. The new and old functions incorporated in COBOL/VSE help your organization streamline the application development process.

How COBOL/VSE Streamlines Application Development

With COBOL/VSE, you can cut the cost and time required to:

- Code
- Debug
- Communicate between languages
- Maintain applications
- Port to new environments

Coding

COBOL/VSE is a powerful language that greatly simplifies and reduces the code required for your applications.

COBOL/VSE provides a variety of intrinsic functions that you can invoke directly from a COBOL statement to perform mathematical, statistical, financial, string handling, or date and time functions.

Using the structured programming features of the COBOL/VSE language adds discipline, clarity, and organization to application programs, thereby increasing programmer productivity.

COBOL/VSE supports LE/VSE, thereby reducing coding requirements for all kinds of application development projects—both simple and complex. With the interlanguage communication (ILC) support provided by LE/VSE, mixed-language applications are easier to implement. You can also use the CALL statement to take advantage of LE/VSE services in the areas of storage management, message handling, date and time processing, mathematical calculations, national language processing, and condition handling. The condition handling support makes it possible to respond to errors and exceptions consistently, uniformly, and predictably, with minimal disruption to your operating environment.

Debugging

COBOL/VSE minimizes debugging time through support for the LE/VSE dump services.

LE/VSE provides a new, improved formatted dump that clearly labels COBOL data areas, making it easier to locate the source of program errors. The formatted dump is available from an LE/VSE service that may be called directly from a COBOL program.

Communicating between Languages

COBOL/VSE works with LE/VSE to simplify communication between the COBOL language and other languages that support LE/VSE.

The improved interlanguage communication (ILC) provided by LE/VSE makes it possible to develop a program once, and only once, in the single best language for the task. Then the program can be called with the same interface from applications that may employ one or more other languages. Reusing already developed code instead of repeating the function in more than one language saves application development time and reduces the potential for introducing errors.

Instead of handcrafting an application using a large amount of unique code in a single language, you can now use a building block approach to application development. You can easily combine heterogeneous programs that may include:

- Programs already written in another language
- Programs generated by tools such as Cross System Product (CSP)
- Vendor-written programs that you want to incorporate into your applications

This building block approach enables you to create applications at considerably less cost.

Maintaining Applications

Of course, the English-like quality of COBOL makes programs easy to read and understand. Add to that the structured programming language features supported by COBOL/VSE and you have a clear-cut maintenance advantage. In addition, the common run-time environment, and common callable services supplied by LE/VSE, make maintenance consistent, and productive, across and within applications.

Porting to New Environments

COBOL/VSE supports Systems Application Architecture* (SAA) COBOL Common Programming Interface (CPI). Programmers can produce applications once to be run in multiple operating environments: VM/CMS, MVS, Operating System/2* (OS/2*), and Operating System/400* (OS/400*). This helps increase productivity by freeing the COBOL application programmer from many system-specific concerns. SAA support also allows wider use of COBOL applications.

COBOL/VSE and LE/VSE

IBM Language Environment for VSE/ESA is a common run-time environment for high-level languages, including COBOL.

LE/VSE and its Benefits

With LE/VSE come common conventions, common run-time facilities, common callable services, and conforming language products, such as COBOL/VSE. These yield benefits such as uniformity and consistency, improved interlanguage communication (ILC), reusable libraries, and simplified and more efficient application development, as shown in Figure 1.

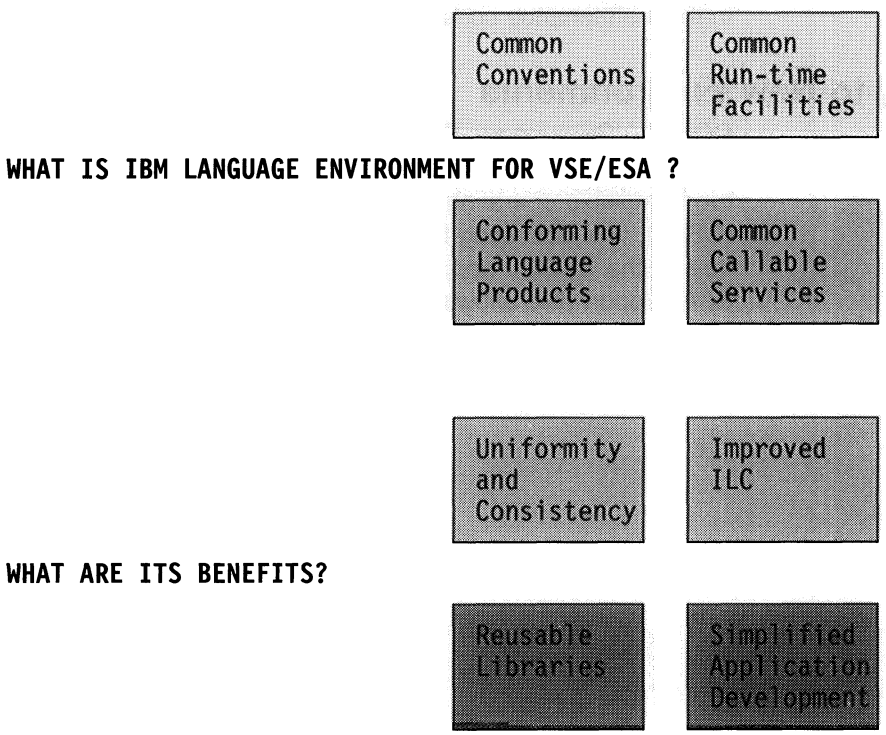


Figure 1. LE/VSE and Its Benefits

LE/VSE is consistent across supported operating systems and subsystems, and across supported languages. Library function is provided by a combination of common library routines and those provided especially for each language.

When you build applications, you want to be able to use program function wherever and however it exists, regardless of the language of implementation. The improved ILC provided by LE/VSE makes it possible to integrate already existing program packages to form new or updated applications. With LE/VSE you can move from separate, single-language applications with voluminous unique code to integrated, multi-language applications with high use of common code.

Figure 2 depicts the application structure before LE/VSE, and the application structure you can realize as you move toward LE/VSE.

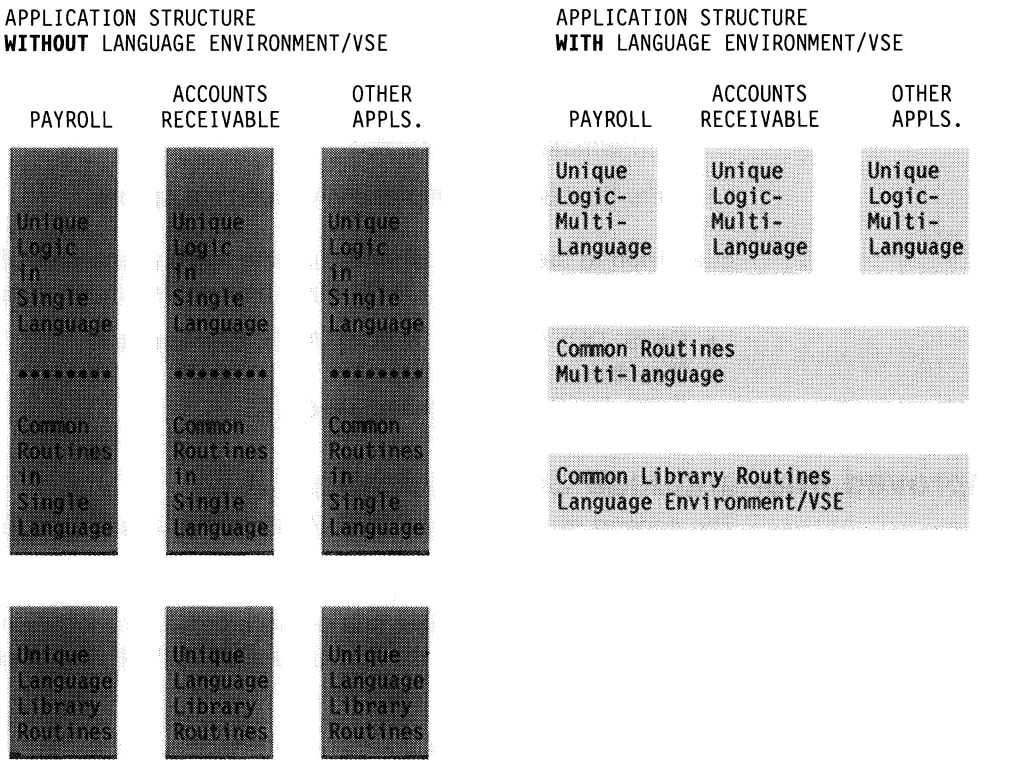


Figure 2. LE/VSE Application Structure

Because applications can easily share code written in different languages, the unique code that is required shrinks while the percentage of common code expands: Application development becomes more efficient.

LE/VSE and COBOL Application Development

The support of LE/VSE by COBOL/VSE provides these new benefits for COBOL application development:

- Improved communication between COBOL and other languages
- Alternative ways to use assembler programs to keep the LE/VSE run-time environment initialized for COBOL programs
- Tuning capability available at execution time

- Powerful callable services using the CALL statement, including:
  - Condition handling
  - Dynamic storage allocation
  - Date and time services
  - Mathematical calculations
  - Message handling
  - National language support
  - Formatted dump service
- More flexible and comprehensive run-time options

In addition to these new benefits, running applications with LE/VSE maintains function provided by the VS COBOL II run-time environment, including support for:

- CICS, DOS/VS DL/I, DFSORT/VSE, Sort/Merge II
- Reentrancy
- Extended System Architecture (ESA)

Improved Interlanguage Communication (ILC)

The support of LE/VSE by COBOL/VSE makes interlanguage communication (ILC) easier and faster than with existing ILC implementations.

A single run unit can contain one or more programs written in any language enabled by LE/VSE. For the first release of LE/VSE, the enabled languages are PL/I and COBOL.

A COBOL run unit is equivalent to an LE/VSE *enclave*. An enclave is an independent collection of routines, one of which is designated as the MAIN program. The MAIN program may, or may not be, a COBOL program.

LE/VSE provides a defined linkage convention to allow programs to call one another at run time. These protocols lessen the burden of writing an interlanguage communication (ILC) application. The ILC support, coupled with the common condition handling function of LE/VSE, makes error recovery predictable for multilanguage applications.

Example of ILC

Table 3 shows a COBOL program calling a PL/I routine.

Table 3. COBOL Program COBRTN Calling PL/I Program PENTRY

COBOL program COBRTN

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. 'COBRTN'.
   .
   .
   .
 WORKING-STORAGE SECTION.
 01  STRUCT-OUT.
     05  SUBI       PIC S9(9) COMP.
     05  SUBJ       PIC S9(9) COMP.
 PROCEDURE DIVISION.
 STARTS.
*********************************
*  Set up parameters for call     *
*********************************
     MOVE 11 TO SUBI.
     MOVE 22 TO SUBJ.
*********************************
*  Call PL/I Program PENTRY       *
*   parameters by reference       *
*********************************
     CALL 'PENTRY' USING STRUCT-OUT.
     GOBACK.
```

PL/I program PENTRY

```
/************************************/
/* Set up COBOL linkage for PENTRY   */
/************************************/
PENTRY: PROC(STRUCT) OPTIONS(MAIN);
/* Define Data Structure            */
DCL 1 STRUCT,
      2 SUBI  FIXED BINARY(31),
      2 SUBJ  FIXED BINARY(31);
/* Add SUBI to SUBJ                 */
STRUCT.SUBJ = STRUCT.SUBJ + STRUCT.SUBI;
END PENTRY;
```

Improved ILC Performance

When a COBOL program calls a non-COBOL program without using LE/VSE, the COBOL environment must be saved and restored—a costly process.

With LE/VSE, however, only *one* environment exists. Programs follow common ILC conventions, including standard parameter and return code processing. The single environment and the common conventions make possible a straightforward, faster process.

Figure 3 shows the ILC process without LE/VSE, and the improved, streamlined process available with LE/VSE.

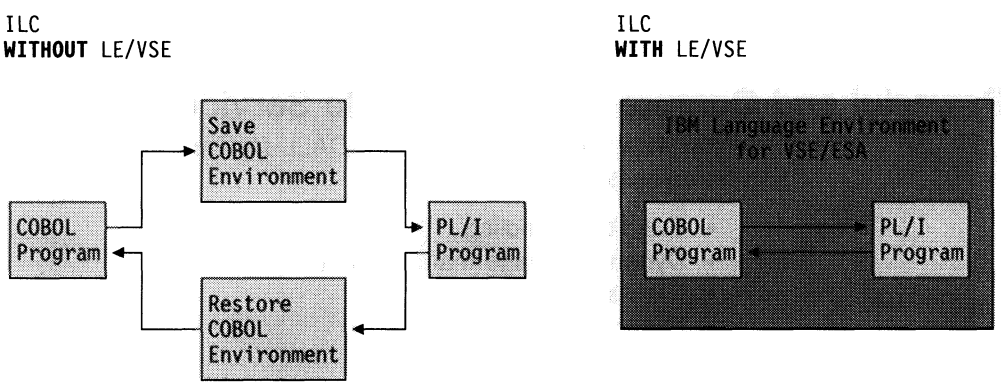


Figure 3. ILC Process—Improved with LE/VSE

Using Assembler Programs with COBOL

Your applications can contain assembler programs that interact with one or more COBOL programs. If you have a non-CICS application in which an assembler program invokes COBOL programs repeatedly, you can make the application run faster. Invoking COBOL from an assembler program enables you to retain the LE/VSE run-time environment between COBOL program invocations. To do this, you can use either of the following:

- An LE/VSE-conforming assembler program
- The LE/VSE pre-initialization function

Saving Processing Time: LE/VSE-conforming Assembler Program

LE/VSE supplies macros that you can use to develop an LE/VSE-conforming assembler program that may invoke COBOL programs. Because the LE/VSE run-time environment for the programs remains when the COBOL programs return control, you can save computing time by limiting initialization and termination processing.

Saving Processing Time: LE/VSE Pre-initialization Function

The pre-initialization function can be used from an assembler program that may, or may not, conform to LE/VSE conventions. The assembler program can interface with the LE/VSE pre-initialization function to invoke COBOL/VSE programs, which can, in turn, call either VS COBOL II or other COBOL/VSE programs. The LE/VSE run-time environment remains initialized for the COBOL programs until the assembler program is terminated.

Improved Storage Tuning

The LE/VSE support provided by COBOL/VSE makes storage tuning easy and flexible. You can now perform storage tuning at run time in addition to using link-edited tuning tables. This means that you can change storage parameters until you are satisfied with how storage is being allocated.

LE/VSE provides run-time options that control the allocation of storage for both user and system requirements, including space for COBOL working storage and external data. Run-time options enable you to specify the size of initial allocations and the size of subsequent incremental storage requests. In addition, storage reports let you analyze just how storage is being allocated for each application. Through the additional storage management function provided with LE/VSE, you can more easily control the allocation of storage for your COBOL applications.

Powerful and Comprehensive Callable Services

Now available, via the standard CALL statement, are more than 80 services that help you perform:

- Condition handling
- Dynamic storage management
- Date and time calculations
- Mathematical calculations
- Message handling
- National language support
- General services such as obtaining an LE/VSE formatted dump

Using the services from a COBOL program is easy, as demonstrated by the following example of calling a service that requires two parameters:

```
 77  Parm-string       pic  x(80) display.
 77  Feedback-code     pic  x(12) display.

Call "CEE5PRM" using parm-string, feedback-code.
```

Figure 4. Example of Using a Callable Service from COBOL

Parm-string contains the particular parameters required by a service, and LE/VSE indicates whether the service completed successfully in *Feedback-code*.

Condition Handling

LE/VSE condition handling provides facilities that allow COBOL/VSE applications to react to unexpected errors.

You can use language constructs or run-time options to select the level at which you want to handle each condition. For example, you can decide to handle a particular error in your COBOL program, let the LE/VSE condition handler take care of it, or percolate the error so that it is handled by the operating system. Only a truly catastrophic failure need disrupt your application environment.

In support of LE/VSE condition handling, COBOL/VSE introduces language extensions for procedure pointers, as described in "COBOL/VSE Programs and LE/VSE" on page 29.

Dynamic Storage Services

These services enable you to get, free, and reallocate storage. In addition, you can create your own user-defined storage pools.

Date and Time Services

With the date and time services, you can get the current local time and date in several formats, and perform date and time conversions. Two callable services, CEEQCEN and CEESCEN, provide a predictable way to handle 2-digit years, such as 94 for 1994 or 02 for 2002.

Mathematical Services

Calculations that are easy to perform with this type of callable service include logarithmic, exponential, trigonometric, square root, and integer functions.

Note: COBOL/VSE also supports a set of intrinsic functions that include some of the same mathematical and date functions. The LE/VSE callable services and intrinsic functions provide equivalent results for the same functions. See "Intrinsic Functions" on page 23 for an overview of intrinsic functions and *COBOL/VSE Programming Guide* for an explanation of the differences between COBOL/VSE intrinsic functions and LE/VSE date and mathematical services.

Message Handling

Message handling services include getting, dispatching, and formatting messages. Messages for non-CICS applications can be directed to files or printers, while CICS messages are directed to a CICS transient data queue. LE/VSE takes care of splitting the message to accommodate the record length of the destination, and presenting the message in the correct national language, such as Japanese or English.

National Language Support Services

These services make it easy for your applications to support the language wanted by application users. You can set the language and country, and obtain default date, time, number, and currency formats. For example, you might want dates to appear as 23 June 99, or 6,23,99.

General Callable Services

LE/VSE also offers a set of general callable services, which include the capability to get an LE/VSE formatted dump.

Depending upon the options you select, the LE/VSE formatted dump may contain the names and values of variables, and information about conditions, program tracebacks, control blocks, storage, and files. All LE/VSE dumps have a common, well-labeled, and easy-to-read format.

Sample List of LE/VSE Callable Services

The following table gives examples of a few callable services available with LE/VSE.

Note: Many more services are available than those listed in the table. For a complete list, see *LE/VSE Programming Guide*.

Table 4 (Page 1 of 2). LE/VSE Callable Services—Types and Sample List.

Function Type	For Example:	
Date and Time	**CEEQCEN, CEESCEN**	To query and set the century. These two callable services are valuable when one or more programs use two digits to express a year. That is, 03, can easily be interpreted as 2003 and not 1903.
	CEEGMTO	To calculate the difference between the local system time and Greenwich Mean Time.
	CEELOCT	To get the current local time in your choice of three formats.
Mathematical Services	**CEESIABS**	To calculate the absolute value of an integer.
	CEESSNWN	To calculate the nearest whole number for a single-precision floating-point number.
	CEESSCOS	To calculate the cosine of an angle.
Dynamic Storage Services	**CEEGTST**	To get storage.
	CEECZST	To change the size of a previously allocated storage block.
	CEEFRST	To free storage.
Condition Handling Services	**CEEHDLR**	To register a user condition handler.
	CEESGL	To raise or signal a condition.
	CEEMRCR	To specify where execution resumes after the condition handler has completed.

Table 4 (Page 2 of 2). LE/VSE Callable Services—Types and Sample List.

Function Type	For Example:	
Message Handling Services	**CEEMOUT**	To dispatch a message.
	CEEMGET	To retrieve a message.
National Language Support Services	**CEE5LNG**	To change or query the current national language.
	CEE5CTY	To change or query the current national country.
	CEE5MCS	To obtain the default currency symbol for a given country.
General Services	**CEE5DMP**	To obtain an LE/VSE formatted dump.

Using LE/VSE Callable Services—an example

Many callable services offer the COBOL programmer entirely new function that would require extensive coding using previous versions of COBOL. Two such services are CEEDAYS and CEEDATE, which you can use effectively when you want to format dates for output.

Figure 5 on page 14 shows a sample COBOL program that uses LE/VSE services to format and display a date from the results of a ACCEPT statement.

```
 ID DIVISION.                                                          B0000010
 PROGRAM-ID. HOHOHO.                                                   B0000020
*************************************************************        B0000030
* FUNCTION:  DISPLAY TODAY'S DATE IN THE FOLLOWING FORMAT: *          B0000040
*           WWWWWWWWW, MMMMMMMM DD, YYYY                   *          B0000050
*                                                          *          B0000060
*           I.E.   SUNDAY, DECEMBER 25, 1994               *          B0000070
*                                                          *          B0000080
*************************************************************        B0000090
 ENVIRONMENT DIVISION.                                                 B0000100
 DATA DIVISION.                                                        B0000110
 WORKING-STORAGE SECTION.                                              B0000120
                                                                       B0000130
 01   CHRDATE.                                                         B0000140
     05 CHRDATE-LENGTH     PIC S9(4) COMP VALUE 10.                   B0000150
     05 CHRDATE-STRING     PIC X(10).                                 B0000160
 01   PICSTR.                                                          B0000170
     05 PICSTR-LENGTH      PIC  S9(4) COMP.                           B0000180
     05 PICSTR-STRING      PIC  X(80).                                B0000190
                                                                       B0000200
 77   LILIAN PIC           S9(9) COMP.                                B0000210
 77   FORMATTED-DATE       PIC X(80).                                 B0000220
 77   DAYSFC               PIC X(12).                                 B0000230
 77   DATEFC               PIC X(12).                                 B0000240
                                                                       B0000250
 PROCEDURE DIVISION.                                                   B0000260
*****************************************************************    B0000270
*    USE  LE DATE/TIME CALLABLE SERVICES TO PRINT OUT         *      B0000280
*    TODAY'S DATE FROM COBOL ACCEPT STATEMENT.                *      B0000290
*****************************************************************    B0000300
     ACCEPT CHRDATE-STRING FROM DATE.                                  B0000310
                                                                       B0000320
     MOVE "YYMMDD" TO PICSTR-STRING.                                   B0000330
     MOVE 6 TO PICSTR-LENGTH.                                          B0000340
     CALL "CEEDAYS" USING CHRDATE , PICSTR , LILIAN , DAYSFC.          B0000350
                                                                       B0000360
     MOVE " WWWWWWWWWZ, MMMMMMMMMZ DD, YYYY " TO PICSTR-STRING.       B0000370
     MOVE 50 TO PICSTR-LENGTH.                                         B0000380
     CALL "CEEDATE" USING LILIAN , PICSTR , FORMATTED-DATE ,           B0000390
          DATEFC.                                                      B0000400
                                                                       B0000410
     DISPLAY "******************************".                       B0000420
     DISPLAY FORMATTED-DATE.                                           B0000430
     DISPLAY "******************************".                       B0000440
                                                                       B0000450
     STOP RUN.                                                         B0000460
                                                                       B0000470
```

Figure 5. Example of Using LE/VSE Callable Services

Using CEEDAYS and CEEDATE drastically reduces the code required without LE/VSE.

Comprehensive Run-Time Options

LE/VSE supports a full-function set of run-time options, offering greater flexibility, while maintaining compatibility with previous COBOL run-time options. Run-time options determine the conditions under which your application runs. You can use them to specify:

- How conditions are handled
- What kind of debugging is activated

- How storage management is to be handled for your application
- Whether an application has 31-bit addressing
- The national language and country settings
- How messages are processed

Condition Handling

Run-time options enable you to specify the conditions under which an application is abnormally terminated and what information you want when a condition occurs. You can specify, by abend code, which failures you want the operating system to process as abnormal terminations and which you want handled by the LE/VSE condition handler. Upon abnormal termination, you can specify what level of information you want; that is, whether you want a dump, a trace, messages, or no information.

Debugging

Run-time options also specify if debugging is activated. You can specify the DEBUG run-time option to activate the WITH DEBUGGING feature of COBOL.

You can also use the CHECK run-time option in conjunction with the SSRANGE compiler option to automatically check for valid data references. This enables you to make sure that indexes, subscripts, and reference modifiers have values within an acceptable range to avoid storing data at incorrect storage locations.

Storage Management

In addition to callable services, you can use run-time options to set storage options. These options enable you to conduct storage tuning at run time instead of via link-edited tables. In addition, you can specify a run-time option to obtain a storage report.

Indicating that an Application has 31-bit Addressing

With LE/VSE and COBOL/VSE, you can use the ALL31(ON) run-time option, in conjunction with the DATA(31) compiler option, to specify that an application is running with 31-bit addressing and that external data can be allocated space above the 16-megabyte line.

National Language

Run-time options specify the initial national language for your application and country-specific defaults for formats such as date, time, and currency.

Message Handling

You can use run-time options to specify where messages are directed and how many control blocks to allocate for the messages.

New and Compatible Function

Some run-time options offer entirely new function that is unavailable without COBOL/VSE, and some provide the equivalent function to that available with VS COBOL II.

One new function is the capability to perform storage tuning at run time. DOS/VS COBOL does not support storage tuning, while VS COBOL II supports it only at link-edit time. With COBOL/VSE, you can try various parameter combinations when you run an application and get comprehensive storage reports to determine

optimal storage management parameters. Once established, the tuning parameters can then be link-edited.

An example of a function that is compatible with one provided by VS COBOL II, but which is unavailable with DOS/VS COBOL, is the LE/VSE CHECK option. This option gives you equivalent function to that provided by the VS COBOL II run-time option, SSRANGE. The CHECK option, in conjunction with the SSRANGE COBOL/VSE compiler option, can save your organization a considerable amount of debugging effort for a common, but extremely difficult to debug, programming error. One cause for an enormous amount of debugging effort occurs when indexes, subscripts, or reference modifiers are incorrectly set to large numbers, and data is inappropriately stored. Program instructions may even get modified, and the symptom of the problem may bear no obvious relation to the subscript that is out-of-range. If the CHECK run-time option is in effect, the problem is automatically caught, the program terminated, and a message like the following issued:

```
The reference to table 'ALL-FIXED' by verb number '01' on line
'008000' addressed an area outside the region of the table.
```

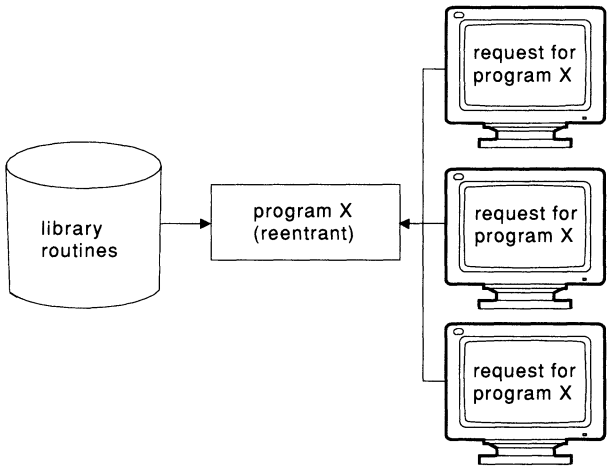
Support for Reentrancy

LE/VSE supports programs that are *reentrant*. If a reentrant program (or a reentrant subroutine) is placed in a shared area of virtual storage, a single copy of the program will satisfy all requests for the program—even simultaneous requests.

With the improved ILC of LE/VSE, reentrant applications can now include both PL/I and COBOL subprograms.

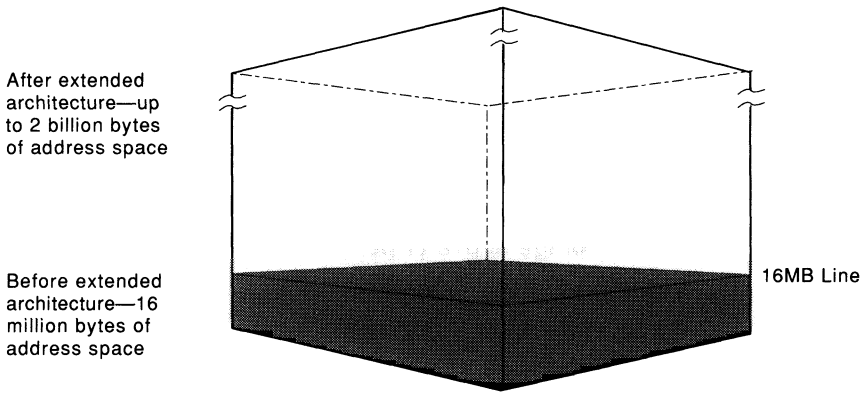
In addition to its support for reentrancy, LE/VSE supports programs that are not reentrant. However, if a program is not reentrant, a separate copy of the program is used in response to each request. Many copies of the program may be in use at the same time, consuming extra storage and wasting processing time.

Most LE/VSE library routines are reentrant, and LE/VSE supports reentrant programs generated by the COBOL/VSE compiler. As illustrated by the following figure, you can save storage in two ways: by sharing reentrant programs and by sharing library routines.



Support for Extended Architecture

COBOL/VSE can be used with the VSE/ESA operating system or CICS/VSE* subsystem in IBM processors that operate in extended architecture mode. As the following figure illustrates, extended architecture stretches the working area—the address space—available to a program. A program and its data are therefore not constrained by a 16-million-byte address space.



Using 31-bit Addressing

When operating under an environment that exploits extended addressing, COBOL/VSE can use 31-bit addressing, allowing you to take advantage of address space above the 16-megabyte line. COBOL/VSE extends the limits on the size of a program's data area and lets you develop applications that can run in the extended address space above 16 megabytes.

Thus, you can construct large applications that use extensive tables of data without resorting to techniques like segmentation to fit large programs into available address space.

The flexibility of 31-bit addressing allows you to run programs either above or below the 16-megabyte line, depending on the application size.

You need not recompile programs compiled with RENT that run on a System/370* processor (and its 24-bit addressing mechanism) to exploit 31-bit addressing on an extended architecture processor.

With COBOL/VSE, 31-bit addressing offers advantages in the following:

CICS: CICS COBOL programs can be run in 31-bit mode and must be compiled to be reentrant. If you place a program in a shared area of virtual storage (for instance, the shared virtual area of VSE/ESA), multiple tasks in a processor can share it; transactions in separate CICS regions can share it; and transactions in a single CICS region can share one copy of a program placed in that region.

DOS/VS DL/I: COBOL/VSE programs that run with the latest versions of DL/I can reside and run above 16 megabytes, enabling them to take advantage of the extended address space offered by extended architecture processors. Data areas used by the application to communicate with DL/I can reside above 16 megabytes; this is accomplished by specifying the compiler option DATA(31). Note that the DATA(31) option is applicable only if you are using a level of DL/I that supports data areas above the line.

VSAM buffers: COBOL/VSE locates its VSAM buffers above 16 megabytes when appropriate. 31-bit addressing under VSE/ESA provides as much as 2 gigabytes of available storage for these large buffers.

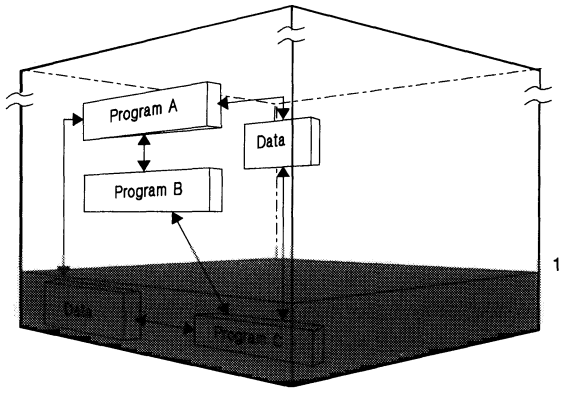
SORT/MERGE: COBOL/VSE programs with SORT/MERGE statements can be run above 16 megabytes to take advantage of the extended address space offered in extended architecture processors.

The COBOL/VSE compiler itself, and most LE/VSE library routines, can run in 31-bit mode and be loaded above 16 megabytes.

As illustrated in the following figure:

- The operating system can load a COBOL/VSE application at an address in virtual storage above or below the 16-megabyte line.
- The COBOL/VSE application program can call programs that run above 16 megabytes—even if the calling program is loaded below the 16-megabyte line.
- The COBOL/VSE application can access data that is loaded above or below 16 megabytes.

The DATA compiler option lets you allocate space for data below 16 megabytes, even if the program that uses the data runs above the 16-megabyte line.



COBOL/VSE and Advanced Language Features

COBOL/VSE is a comprehensive language that maintains the features introduced by VS COBOL II and adds still more. The COBOL/VSE language is powerful; it encourages structured programming, offers a wide variety of data types, and makes it easy to use other products such as DFSORT/VSE or CICS. New with COBOL/VSE is a set of intrinsic functions that enable you to perform mathematical, statistical, and date and time functions with simple invocations from COBOL statements.

This chapter describes only a subset of the advanced language features available with COBOL/VSE. These include:

- Structured programming language constructs
- Intrinsic functions
- Connectivity to other products such as CICS, DOS/VS DL/I, DFSORT/VSE, Sort/Merge II, and LE/VSE
- A wide variety of data types and character sets
- Other language features such as improved file handling and character string manipulation

Structured Programming

The COBOL 85 Standard (the ISO and ANSI standards identified in Appendix A, "Industry Standards" on page 47), supported by COBOL/VSE, provides efficient language constructs. These constructs include nested programs, in-line PERFORM statements, nested statements using explicit scope terminators, and the EVALUATE statement. Using these constructs will aid in the development of applications that conform to top-down design, modular program development, and structured programming concepts. Maintenance of structured programs is easier and less costly.

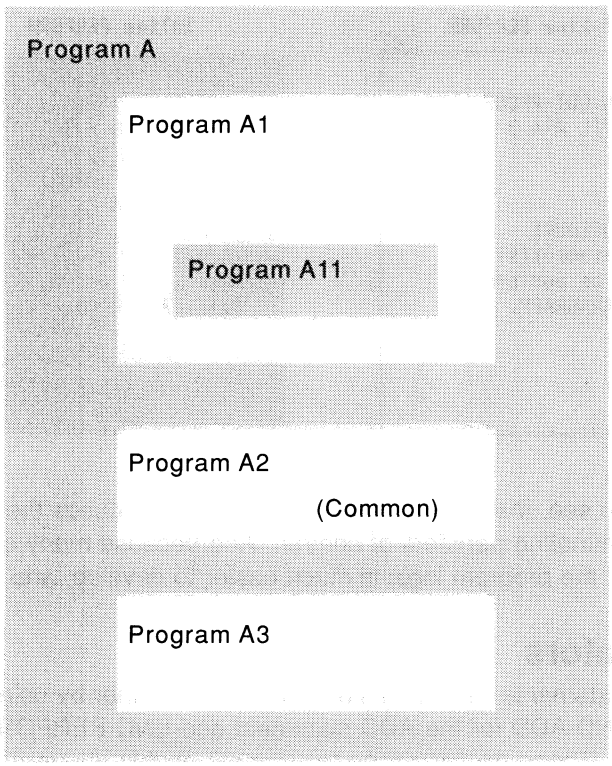
Coding without Structured Programming

Without structured programming support, GO TO statements are frequently necessary, program flow is difficult to follow, and maintenance becomes a headache. Consequently, the cost of using an application is ever-increasing.

Nested Programs

The COBOL 85 Standard allows you to develop nested COBOL programs, which help you in top-down design. This means that a COBOL source program can contain another COBOL source program, and a contained COBOL source program can in turn contain other COBOL source programs.

The following example illustrates a nested program structure. In this example, Program A1 can *only* be called by Program A. Program A2 may be called by Program A, and, because of the COMMON attribute, can also be called by Programs A1, A11, and A3.



Nested COPY Statements

The COPY statement has always allowed you to develop program functions that can be stored separately from other program code, then be included in the program at compile time without the run-time overhead that CALL statements require.

With COBOL/VSE, as with VS COBOL II, COPY statements can be nested. A COPY statement can include a segment of code that contains COPY statements. And the code included by the nested COPY statement(s) can contain COPY statements that are nested even deeper.

Nested COPY statements can be used to conveniently build nested programs. This encourages modular application development.

Inline PERFORM Statements

The inline PERFORM statement, when paired with the END-PERFORM scope terminator, allows performed procedures to be coded directly inline with the PERFORM statement that invokes them:

Out-of-Line PERFORM

```
PERFORM OUT-PROCEDURE
    UNTIL A = B
    .
    .
    .
OUT-PROCEDURE
   (statement(s)
    to be performed)
NEXT-PARAGRAPH.
    .
    .
    .
```

Inline PERFORM

```
PERFORM
    UNTIL A = B
(statement(s)
 to be performed)
END-PERFORM
    .
    .
    .
NEXT-PARAGRAPH.
    .
    .
    .
```

As you can see, the inline PERFORM statement reduces the complexity of program flow by eliminating transfers of control. And because many control transfers are eliminated, the program logic is much easier to develop, and understand.

Explicit Scope Terminators

COBOL statements can be nested within one another by using *scope terminators* such as END-ADD for the ADD statement and END-PERFORM for the PERFORM statement. These explicit scope terminators enable you to define precisely the range of each statement. By making the end of scope visually evident, explicit scope terminators make the logic of a program much easier to follow.

The EVALUATE Statement

The EVALUATE statement is another construct that allows you to develop and code structured programs. Just one EVALUATE statement allows several alternative paths of execution, clarifying program logic and simplifying program coding.

In addition, one EVALUATE statement can take the place of several IF statements. For example:

IF Statements

```
If NUMERIC-ID=1
   Move "one" to ALPH-ID
Else
If NUMERIC-ID=2
   Move "two" to ALPH-ID
Else
If NUMERIC-ID=3
   Move "three" to ALPH-ID
Else
   Move "zero" to ALPH-ID.
    .
    .
    .
```

Corresponding EVALUATE Statement

```
Evaluate NUMERIC-ID
  When 1
    Move "one" to ALPH-ID
  When 2
    Move "two" to ALPH-ID
  When 3
    Move "three" to ALPH-ID
  When Other
    Move "zero" to ALPH-ID
End-Evaluate.
    .
    .
    .
```

Intrinsic Functions

COBOL/VSE offers a set of functions that you can invoke to provide values at execution time. These values might require complex calculations such as the calculation of present value or the average of a large number of values. Function invocations may replace identifiers or arithmetic expressions in many language constructs. Intrinsic functions and the ALL subscript are a powerful combination, making it possible to reduce the code required for applications that require mathematical, statistical, financial, or date and time calculations. For example, you might say

```
IF FUNCTION MEAN(SALARY(ALL)) IS > 30000
```

All values of SALARY, a one-dimensional array, are summed, averaged, and the result compared to $30,000. The ALL specification indicates that all values of SALARY should be considered.

In the previous version of COBOL, you would have to calculate the average salary (without the benefit of a function), place the value in an identifier, and then test the identifier.

COBOL/VSE intrinsic functions help you make mathematical, statistical, financial, character string, and date calculations. The following table presents examples of a few of the intrinsic functions available with COBOL/VSE.

Note: Many more intrinsic functions are available than are listed in the table. For a complete list, see Appendix B, "COBOL/VSE Intrinsic Functions" on page 50.

Table 5. Intrinsic Functions—Types and Sample List.

Function Type	For Example:	
Mathematical	**SIN**	To calculate the sine of an angle.
	ATAN	To calculate the arctangent; that is to find the angle, given the tangent.
	MAX	To calculate the maximum value.
	SQRT	To calculate the square root.
Statistical	**MEAN**	To calculate the average.
	RANDOM	To generate a random number.
	STANDARD-DEVIATION	To calculate the standard deviation.
Financial	**ANNUITY**	To perform ANNUITY calculations.
	PRESENT-VALUE	To calculate the present value.
Date and Time	**CURRENT-DATE**	To get the current date and the difference from Greenwich Mean Time.
	DAY-OF-INTEGER	To find the Julian date equivalent (YYYYDDD) of an integer date.
Character String	**LOWER-CASE**	To convert a string to lowercase.
	NUMVAL	To find the numeric value of a simple numeric string.
	REVERSE	To reverse the order of the characters in a string.
Miscellaneous	**ORD-MAX**	To find the ordinal position n, of a maximum value. For example, you could find that the nth person in the list has the highest salary, and use n as a subscript to find the person's name.
	LENGTH	To find the length of an argument.
	WHEN-COMPILED	To find the date and time a program was compiled.

Intrinsic functions can be nested. If, for example, you want to calculate the log of the factorial of 2X+1 and store the result in Z, you can use this statement:

```
Compute Z =
  function log(function factorial(2 * X + 1))
```

Assume that you require a program to calculate the total payroll, the average salary, the salary and name of the employee with the maximum salary, and the salary range.

Table 6 shows the Procedure Division calculations required with COBOL/VSE, and Table 7 on page 25 shows the Procedure Division calculations required without COBOL/VSE.

Table 6. Intrinsic Function Example with COBOL/VSE

```
compute max-salary = function max(emp-salary(all)).
compute i = function ord-max(emp-salary(all)).
move emp-name(i) to highest-paid.
compute avg-salary = function mean(emp-salary(all)).
compute salary-range = function range(emp-salary(all)).
compute total-payroll = function sum(emp-salary(all)).
```

Table 7. Intrinsic Function Example without COBOL/VSE

```
compute max-salary = salary(1).
compute min-salary = salary(1).
move emp-name(1) to highest-paid.
perform varying x from 1 by 1
        until x > emp-count
   if salary(x) > max-salary then
      compute max-salary = salary(x)
      move emp-name(x) to highest-paid
   else
      if salary(x) < min-salary then
         compute min-salary = salary(x)
      end-if
   end-if
   compute total-payroll = total-payroll + salary(x)
end-perform.
compute avg-salary = total-payroll / emp-count.
compute salary-range = max-salary - min-salary.
```

Using Other Products from COBOL/VSE Programs

The COBOL/VSE language makes it easy to use the services of other products.

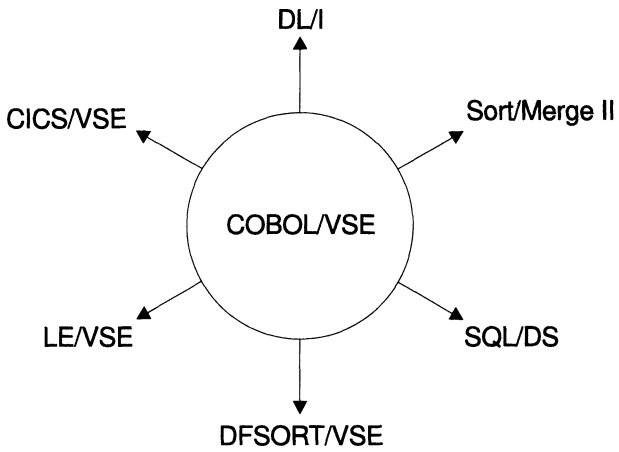


Figure 6. COBOL/VSE Connects to Other Products

Among the products that you can use from COBOL/VSE programs are:

- Customer Information Control System (CICS)
- DOS/VS DL/I
- DFSORT/VSE
- Sort/Merge II
- Structured Query Language/Data System (SQL/DS*)
- IBM Language Environment for VSE/ESA (LE/VSE)

COBOL/VSE Programs and CICS

CICS provides facilities for real-time, transaction-oriented database/data communications applications. These facilities include the management of programs, tasks, storage, files, and transient data that may be used in applications such as message switching, inquiry, data collection, and conversational data entry.

The support of CICS COBOL applications by COBOL/VSE, in combination with LE/VSE, can boost productivity and aid in top-down design and modular program development. Because COBOL/VSE subprograms can contain CICS commands, modular design and development principles are easier to implement.

In addition, it is no longer necessary to perform the time-consuming task of BLL (Base Locator for Linkage) cell addressing since based addressing is now done using pointer variables and the ADDRESS OF special register.

Improved CICS COBOL Application Development

COBOL/VSE–CICS applications are now much easier to design and code, because many of the tasks required for previous versions of COBOL have been eliminated. These tasks include:

- Defining and manipulating lists of pointers (BLL cells) for COBOL programs in locate mode. (In locate mode, the CICS commands for data access specify the SET option rather than the INTO or FROM option.)
- Issuing SERVICE RELOAD statements to re-initialize BLL cells.
- Specifying the length of the data for many CICS commands.
- Taking special actions if the size of a data record exceeds 4 Kilobytes.

Additionally, as part of this simplified COBOL-CICS interface first introduced with VS COBOL II, you can use the following language elements:

GOBACK statement: Returns control to another COBOL program or to CICS.

EXIT PROGRAM statement: Returns control from a COBOL subprogram to a COBOL main program.

STOP RUN statement: Returns control to CICS.

CALL statement: When operating under CICS, COBOL/VSE programs can call, or be called by, VS COBOL II or other COBOL/VSE programs. Both static call and the CALL *identifier* form of dynamic call are supported by COBOL/VSE. All VS COBOL II and COBOL/VSE subprograms that are called can also contain CICS commands.

In addition, with LE/VSE, ILC within a run unit is now supported for CICS applications.

Data manipulation statements: The use of the STRING, UNSTRING, and INSPECT statements is supported in the CICS environment.

ADDRESS OF special register: COBOL/VSE allows the use of the ADDRESS OF special register when operating under CICS. You can use the ADDRESS OF special register to hold the address of a variably-located record area, such as an area to hold input or output records.

LENGTH OF special register: COBOL/VSE allows the use of the LENGTH OF special register when operating under CICS. A reference to the LENGTH OF special register returns the current length of an associated data item. For example, the statement `MOVE LENGTH OF A TO B` places the current length of item A into item B.

CICS uses the LENGTH OF special register to determine the length of a referenced data item in a CICS command if the length is not explicitly stated in the command.

Pointer data type: An extension of the USAGE clause, USAGE IS POINTER, defines a type of data item that can be used to hold addresses. It allows you to move addresses received from other programs or licensed products to a part of a record, then pass this record to another program.

Using pointer data items, you can also set up or process chained lists within a program.

Example of CICS-COBOL Code Simplification

The following example illustrates the coding required by DOS/VS COBOL compared with COBOL/VSE in a program that interfaces with CICS. The program reads data from, and writes data to, a file in a CICS environment. Note the simplified coding in the COBOL/VSE program.

COBOL–CICS Interface DOS/VS COBOL

```
WORKING-STORAGE SECTION.
77 LRECL-REC1   PIC S9(4) COMP.

LINKAGE SECTION.

01 BLLCELLS.
   02 FILLER    PIC S9(8) COMP.
   02 BLL-REC1A PIC S9(8) COMP.
   02 BLL-REC1B PIC S9(8) COMP.
   02 BLL-REC2  PIC S9(8) COMP.

01 REC-1.
   02 CTR       PIC  9(4) COMP.
   02 FLD       PIC  X(100)
        OCCURS 1 TO 50 TIMES
          DEPENDING ON CTR.

01 REC-2.
   02 ...

PROCEDURE DIVISION.

EXEC CICS READ UPDATE ...
  SET (BLL-REC1A)
  LENGTH (LRECL-REC1)
  END-EXEC.

SERVICE RELOAD REC-1.

IF LRECL-REC1 > 4096
   THEN ADD 4096 TO BLL-REC1A
         GIVING BLL-REC1B.

EXEC CICS REWRITE ...
   FROM (REC-1)
   LENGTH (LRECL-REC1)
   END-EXEC.
```

COBOL–CICS Interface COBOL/VSE

```
LINKAGE SECTION.

01 REC-1.
   02 CTR       PIC  9(4) COMP.
   02 FLD       PIC  X(100)
        OCCURS 1 to 50 TIMES
          DEPENDING ON CTR.

01 REC-2.
   02 ...

PROCEDURE DIVISION.

EXEC CICS READ UPDATE ...
  SET (ADDRESS OF REC-1)
  END-EXEC.

EXEC CICS REWRITE ...
   FROM (REC-1)
   END-EXEC.
```

COBOL/VSE Programs and DOS/VS DL/I

DOS/VS DL/I provides both batch and online processing services for hierarchical databases. These services include:

- Database services: retrieving, inserting, deleting, and replacing data, and the processing of statistical requests
- System services: basic and symbolic checkpoint, extended restart, and rollback

Your COBOL programs can use DOS/VS DL/I services with calls to the DL/I interface. The following example shows a COBOL statement that calls DOS/VS DL/I using the CBLTDLI interface to replace (REPL) a data segment with data stored in *MAST-SEG-IN*.

```
CALL 'CBLTDLI' USING FUNC-REPL, DB-PCB-MAST, MAST-SEG-IN.
```

New with COBOL/VSE via LE/VSE is the CEETDLI call interface, which takes advantage of the LE/VSE condition handling services. When DL/I is executing on behalf of an application and an exception occurs, the LE/VSE condition handling services will percolate the condition handling to DL/I. This will allow DL/I to perform database rollback to provide data integrity at abnormal termination of the application.

Advantages of using CEETDLI are:

- You can run your program with TRAP(ON)
- Take advantage of exception handling services
- Percolate DL/I conditions to DL/I to ensure data integrity

COBOL/VSE Programs and DFSORT/VSE

DFSORT/VSE is a fast and efficient data processing tool that sorts, merges, and copies files. DFSORT/VSE is as effective for small amounts of data and simple jobs, such as alphabetizing a list of names, as it is for very large amounts of data and complex jobs, such as running a billing system.

COBOL/VSE allows you to use the COBOL SORT/MERGE feature for sequential, relative, and indexed files, except for those whose access mode is random.

The COBOL/VSE language supports DFSORT/VSE with these constructs:

- The SORT statement to sort records from one or more files
- The MERGE statement to combine two or more already sorted files into one, newly sorted file
- Sort Description entries to define sort and merge files in the File Section of a COBOL program
- Several special registers to pass information to DFSORT/VSE

Following is a SORT statement that sorts the file EMPLOYEE-FILE first by last name and then by first name.

```
SORT EMPLOYEE-FILE
  ON ASCENDING KEY LAST-NAME FIRST-NAME
```

COBOL/VSE Programs and SQL/DS

The relational database services of SQL/DS are available via SQL (Structured Query Language) statements embedded in your COBOL program. SQL is both simple and powerful. Single SQL statements can perform the same functions as many lines of conventional code. SQL/DS includes support for these relational database functions:

- Defining, creating, and deleting databases
- Retrieving, adding, deleting, and updating data from a database

- Checkpoint and restart
- Sorting and grouping data
- Built-in functions and mathematical calculations

An SQL statement that updates the manager number (MGRNO) column in the DSN8130.TDEPT table follows.

```
EXEC SQL
  UPDATE DSN8130.TDEPT
    SET MGRNO = :MGR-NUM
    WHERE DEPTNO = :INT-DEPT
END-EXEC
```

Please note that the SQL statement is enclosed by EXEC SQL and END-EXEC. A COBOL program with SQL statements must be precompiled by a preprocessor before it can be compiled by the COBOL compiler. The preprocessor accepts the mixture of COBOL and SQL as input, and produces:

- A source module that can be compiled by COBOL/VSE
- SQL information, obtained from the embedded SQL statements, that is required for SQL/DS processing

With COBOL/VSE, as with VS COBOL II, your SQL/DS applications can be reentrant; that is, only one copy is required, but that copy can be used by many users or programs concurrently. Under DOS/VS COBOL, SQL/DS programs cannot be reentrant.

Also, with COBOL/VSE, as with VS COBOL II, you can use all features of dynamic SQL; that is, you can construct SQL statements dynamically at run time. Under DOS/VS COBOL, dynamic SQL is severely limited.

COBOL/VSE Programs and LE/VSE

LE/VSE, the common run-time environment for COBOL/VSE programs, provides over 80 services that you can call directly from COBOL/VSE programs, using the CALL statement. COBOL/VSE introduces language extensions to support these LE/VSE services.

For LE/VSE callable services that require the address of a procedure to be passed as an argument, COBOL/VSE adds new extensions for both the USAGE clause and the SET statement. Now, you can specify in the USAGE clause that a data item is being used as a procedure pointer; that is, it contains the address of a procedure entry point. With the SET extension you can assign the address of a procedure entry point to the data item. Then, if you want, you can use the name of the data item as a parameter when you call LE/VSE services such as CEEHDLR, which establishes a user condition handler.

These language extensions enable you to use COBOL programs as condition handlers for such conditions as program checks, abends, or software generated signals. LE/VSE condition handling as supported by COBOL/VSE enables your applications to continue processing for all but the most catastrophic system failures.

For more information about the benefits of LE/VSE in general, see "COBOL/VSE and LE/VSE" on page 6; for a complete list of LE/VSE callable services, see *LE/VSE Programming Guide*.

Data Types and Character Sets

In addition to standard data types such as alphabetic, numeric, and alphanumeric, COBOL/VSE supports floating-point calculations, which frequently can improve the accuracy realized by application program calculations.

COBOL/VSE supports both the single-byte character set (SBCS) and the double-byte character set (DBCS). The inclusion of floating-point and DBCS widens the applicability of COBOL to your organization's application requirements.

Floating-Point Data

COBOL/VSE provides highly accurate floating-point exponentiation and a highly accurate conversion algorithm for floating-point calculation, and lets you choose when to use floating-point data types to enhance program performance.

For example, fixed-point exponentiations with large exponents can be evaluated more quickly and accurately when the operands force the exponentiations to be evaluated in floating-point. (Floating-point exponentiations often return more accurate results than fixed-point exponentiations.)

Calculation of arithmetic expressions evaluated in floating-point mode are most efficient when the operands involved require the least amount of conversion. Therefore, if you define the items in an expression as floating-point, COBOL/VSE efficiently and more accurately computes the expression.

Double-Byte Character Set

COBOL/VSE accepts characters in the Double-Byte Character Set (DBCS) and the standard COBOL set of characters. DBCS support makes it easier to develop COBOL applications that require a DBCS character set—such as applications using Kanji data.

To handle DBCS data, COBOL/VSE offers:

- The USAGE DISPLAY-1 clause
- PICTURE clause symbols that define 2-byte characters (as opposed to 1-byte EBCDIC characters)
- Formats for DBCS literals and for nonnumeric literals containing double-byte characters

In addition to using DBCS for application data, you can also use double-byte characters in COBOL program user names.

Other Language Features

Other language features include:

- Flexible ways of initializing values
- Improved file handling
- Reference modification

Flexible Ways of Initializing Values

The INITIALIZE statement, first introduced with VS COBOL II, enables you to set certain types of data items to predetermined values. It is functionally equivalent to one or more MOVE statements. For example, assume that you have a structure G, defined as follows:

```
01 G.
 03 A PIC A.
 03 AN PIC X.
 03 N PIC 9.
 03 ANE PIC XB.
 03 NE PIC 9,999.
 03 FLT USAGE COMP-1
 03 BIN USAGE BINARY.
```

If you want to use standard initializations of blank for alphabetic or alphanumeric data items and zero for numeric data items, COBOL/VSE makes it especially easy. If you are restricted to DOS/VS COBOL constructs, you need several COBOL statements.

Table 8. Initializing via DOS/VS COBOL and via COBOL/VSE

DOS/VS COBOL	COBOL/VSE
MOVE SPACES TO A.	INITIALIZE G.
MOVE SPACES TO AN.	
MOVE ZERO TO N.	
MOVE SPACES TO ANE.	
MOVE ZERO TO NE.	
MOVE ZERO TO FLT.	
MOVE ZERO TO BIN.	

Yes, with COBOL/VSE it takes only *one* statement.

Improved File Handling

Already one of the strong suits in the COBOL language, file-handling improvements, introduced in VS COBOL II, are part of COBOL/VSE. COBOL/VSE enables you to use variable-length records, new file status codes, and external files.

Variable-Length Records

In the past, using DOS/VS COBOL, variable-length record programming techniques required:

- Storing the record length into an explicit field in the record, or
- Calling assembly language routines, or
- Illegally accessing the data management-supplied record length field.

COBOL/VSE enables you to use:

```
RECORD IS VARYING...DEPENDING ON dataname
```

When a COBOL program successfully reads a record, *dataname* is automatically set to the length of the record. With READ INTO and RETURN INTO, the length of the record in the implicit move is automatically provided.

File Status Codes for VSAM Input and Output Requests

Using the FILE-STATUS clause, you can obtain detailed information in response to your VSAM input/output requests.

As the following example demonstrates, you can specify `FS-CODE`, a 2-byte area that holds status information, plus `VSAM-CODE`, a 6-byte area to receive information from VSAM.

```
FILE-STATUS IS FS-CODE, VSAM-CODE
```

External Files and Data

COBOL/VSE enables you to share files and data across an entire run unit. Two programs can do input and output on one file, whereas with DOS/VS COBOL, data has to be passed through CALL USING and the LINKAGE section.

Reference Modification

Reference modification gives you the capability to operate upon substrings of data items, which can reduce the need for subordinate data definitions. For example,

```
MOVE WHOLE-NAME(1:25) TO LAST-NAME
```

transfers the first 25 characters in the variable *WHOLE-NAME* to the variable *LAST-NAME*.

Together with other statements such as STRING and UNSTRING, the COBOL/VSE language makes it easy to process character strings.

Note: This chapter has highlighted only a few of the language features of COBOL/VSE. For a complete description of the language, see *COBOL/VSE Language Reference*.

COBOL/VSE and Advanced Compiler Features

COBOL/VSE is a full-function compiler that can:

- Assist in migration, compatibility, and conformance to standards
- Produce easy-to-use listings
- Generate code that is set up for debugging
- Optimize generated code
- Provide flexible numeric sign processing
- Help you manage storage
- Offer performance improvements in sorting

You can control the operation of the compiler with over 40 compiler options.

Support for Migration, Compatibility, and Conformance to Standards

The use of one or more of the compiler options listed in Table 9 can help provide compatibility with programs written for previous versions of COBOL and can help ensure conformance to industry standards and SAA conventions.

Table 9 (Page 1 of 2). Compiler Options That Assist in Migration, Compatibility, and Standards Conformance

Compiler Option	Description
NUMPROC(MIG)	The NUMPROC(MIG) compiler option causes numeric sign handling in COBOL/VSE programs to be similar to numeric sign handling in DOS/VS COBOL. This saves programming effort and alleviates many incompatibility problems. Specifying NUMPROC(MIG) for a COBOL/VSE program makes migration from DOS/VS COBOL much easier.
CMPR2	The CMPR2 compiler option lets you continue to use your valid VS COBOL II Release 2 programs by providing the same run-time results. The NOCMPR2 compiler option allows you to take full advantage of the COBOL 85 Standard support for your new COBOL/VSE programs.
FLAGMIG	The FLAGMIG compiler option can be used in combination with the CMPR2 compiler option to help identify language elements that exhibit different behavior between VS COBOL II Release 2 and COBOL/VSE.
TRUNC(BIN)	The TRUNC(BIN) compiler option tells COBOL/VSE to perform truncation on binary numeric fields based on the size of the binary field in storage, rather than on the number of digits specified by the PICTURE clause. Binary truncation is more consistent with the handling of binary data by other System/370 software such as CICS.
FLAG	You can identify prior IBM COBOL code that needs modification by recompiling existing application programs with the COBOL/VSE compiler and specifying the FLAG option. The FLAG option determines what level of error messages you want to receive. Specifying that you want to receive all error messages, even those at the warning level, can help you identify which code requires change.

Table 9 (Page 2 of 2). Compiler Options That Assist in Migration, Compatibility, and Standards Conformance

Compiler Option	Description
FLAGSTD	The FLAGSTD compiler option can help ensure that your programs conform to the COBOL 85 Standard, and to the Federal Information Processing Standards Publication (FIPS PUB 21-3). FLAGSTD flags any statement in a program that does not conform to a specific level of the COBOL 85 Standard. You can also request the flagging for optional COBOL 85 Standard subsets and for obsolete language elements.
FLAGSAA	The FLAGSAA compiler option can help ensure that your programs conform to the Systems Application Architecture (SAA) COBOL interface definition. FLAGSAA flags statements not conforming to that interface definition, making it easy to spot nonportable language elements. **Note:** This flagging support is available only for SAA COBOL Level 1.

Availability of Easy-To-Use Compiler Listings

The COBOL/VSE compiler produces, on request, a wide variety of information about a program and its compilation. Optionally, you can include in the listing generated by the COBOL/VSE compiler any of the information described in Table 10.

Table 10 (Page 1 of 2). Description of the Information Available in a Compiler Listing

Type of Information	Description
Source listing	Provides a list of the COBOL source statements in your program. You can tailor source statement information to meet your specific needs. A variety of information can be embedded right in your source listing: • Diagnostic messages immediately following the relevant statement • Cross-reference information • Data Division map information
Cross-reference information	Shows where COBOL/VSE verbs, procedures, data names, and programs are defined and used. Cross-reference lists are sorted alphabetically for ease of use.
Data Division map	Illustrates the structure of defined data in a program, and provides other pertinent information such as where the data is defined and what its attributes are.
Nested program map	Displays, using indentation, the hierarchy of nested programs. In addition, the map indicates the statement number where each nested program is defined and lists the attributes of each nested program.
Diagnostic message listing	Provides a list of diagnostic messages generated by the compiler. You can use compiler options to control the severity level of the messages to be included in the list.

Table 10 (Page 2 of 2). Description of the Information Available in a Compiler Listing

Type of Information	Description
Procedure Division list	Provides information about the Procedure Division, including information about program verb usage, global tables, working storage, and literals, together with a complete list of the assembler code generated by the compiler. Optionally, if you do not want the full assembler listing, you can get a condensed list.

For complete information about the listing options that are available, see *COBOL/VSE Programming Guide*.

Many of the listings you generate from the compiler can help you debug but the compiler helps you debug in other ways, as well.

Other Ways to Use the Compiler to help Debugging

You can use compiler options to generate object code that is set up to:

- Get a symbolic dump of the Data Division of COBOL application programs
- Automatically check for an out-of-range condition

Preparing Your Program to Get a Symbolic Dump

Also with the TEST compiler option, the object code can contain the names of all symbols defined in the Data Division of your COBOL program. LE/VSE uses this information to generate a symbolic dump, which makes it easy to locate the values of all program variables. You can get a dump automatically when a program terminates abnormally, or you can explicitly request a dump from your COBOL program by calling the LE/VSE dump service.

Getting Set Up to Check for Valid Data Ranges

By using the SSRANGE compiler option, you can generate information in the program object code that makes possible automatic range checking. When you specify the CHECK option at run time, values of subscripts, indexes, and reference modifiers are checked automatically before data is stored. For more information about the value of range checking, see "New and Compatible Function" on page 15.

Optimize the Generated Code

To reduce the run time of an object program, the COBOL/VSE compiler can optimize code if you specify the OPTIMIZE compiler option.

COBOL/VSE does some of this optimization automatically. For instance, the compiler ensures that the code it produces for table handling is efficient and that code is addressed efficiently. You can use the OPTIMIZE compiler option to:

- Eliminate unnecessary transfers of control (branches) or simplify inefficient branches.
- Simplify the compiled code for PERFORM or internal CALL statements and their associated procedures. Where it can, the compiler places procedure code inline, eliminating the need for linkage code.

- Streamline redundant computations by substituting the results of initial computations for later repetitions.
- Eliminate constant computations by performing them when the program is compiled.
- Aggregate moves of contiguous, equal-sized items into a single move.
- Delete code that will never be run and identify the deleted code with a warning message.

Flexible Numeric Sign Processing

The NUMPROC compiler option enables you to:

- Speed execution for high-performance programs
- Process data with nonconforming signs
- Make numeric sign handling in COBOL/VSE programs similar to numeric sign handling in DOS/VS COBOL (see the description of NUMPROC(MIG) in Table 9 on page 33)

Speeding Execution for High-Performance Programs

If high performance is a prime consideration, you can choose the NUMPROC(PFD) compiler option. This option tells the COBOL/VSE compiler that numeric signs in packed and zoned decimal fields are in the "preferred-sign" format. If the compiler can make this assumption, it can generate more efficient code.

Processing Data with Nonconforming Signs

For programs processing input data that does not conform to the "preferred-sign" conventions, the NUMPROC(NOPFD) option performs invalid sign processing for all numeric input data.

Compiler Options that Help You with Storage Management

You can use compiler options to:

- Generate reentrant code, with the RENT option.
- Determine whether data areas for reentrant code are obtained from above or below the 16-megabyte line, or from unrestricted storage, with the DATA option.

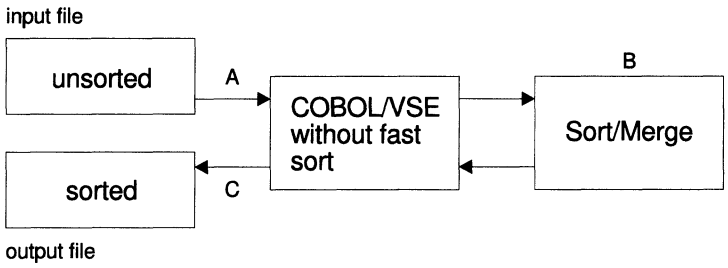
For information about the value of reentrant code and the use of virtual storage above the 16-megabyte line, see "Support for Reentrancy" on page 16 and "Support for Extended Architecture" on page 17, respectively.

Performance Improvement When You Use SORT/MERGE

The performance of programs that sort data via the COBOL SORT/MERGE feature can be improved by using the FASTSRT compiler option. First, consider the process required without the FASTSRT option.

Without Fast Sort

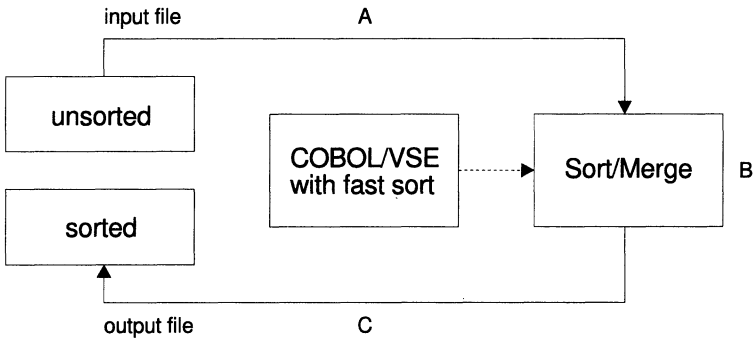
The following figure shows the process required if you do not use the FASTSRT compiler option.



COBOL/VSE will call the SORT/MERGE product to do the actual sorting of data (B in the figure), but COBOL/VSE itself will process the related input and output data (A and C). Thus, for every file to be sorted, COBOL/VSE must perform the steps necessary to release all of the input records needed for the SORT/MERGE product, and to return the sorted records when the process is complete.

With Fast Sort

The following figure illustrates how the SORT/MERGE feature works if you specify the FASTSRT option.

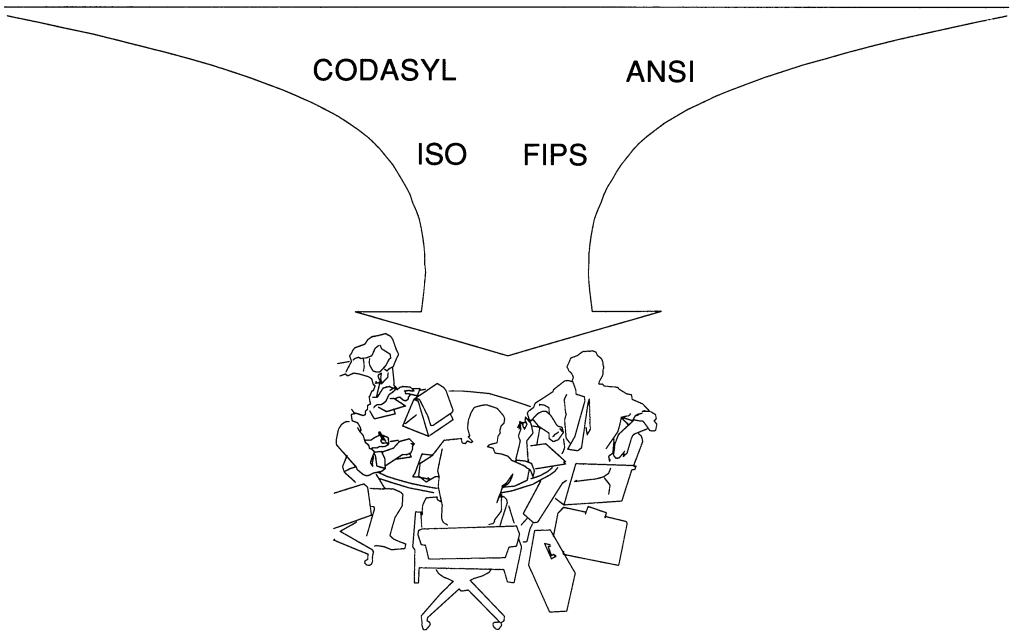


COBOL/VSE can speed up sorting because it can call the SORT/MERGE product to do both the sorting and the processing of the related input/output. This reduces the number of times the data must be moved for processing. DFSORT/VSE provides this SORT/MERGE capability.

If you request fast sorting but the object program cannot perform it, you receive a message indicating which conditions were not met. If you do not request fast sorting and your program meets the fast-sorting conditions, you receive a message to that effect.

COBOL/VSE and Industry Standards

COBOL/VSE supports the COBOL 85 Standard, as defined in Appendix A, "Industry Standards" on page 47. Many organizations contribute to industry standards and practices for the COBOL language:



The complete names for the organizations are:

- American National Standards Institute (ANSI)
- Conference On Data Systems Languages (CODASYL)
- Federal Information Processing Standard (FIPS)
- International Standards Organizations (ISO)

The COBOL 85 Standard met by COBOL/VSE is the accepted standard of the industry. Tools and other products that interface with COBOL applications use this standard as their base. For a list of the industry standards met by COBOL/VSE, see Appendix A, "Industry Standards" on page 47 and for a complete description of the COBOL/VSE language, see *COBOL/VSE Language Reference*.

What Must I Do to Move toward COBOL/VSE?

This chapter provides:

- An overview of the history of IBM COBOL products
- A list of the hardware and software supported by COBOL/VSE
- An overview of the compatibility you can expect between COBOL/VSE and previous System/370 COBOL products
- A short description of the migration aids that are available
- A list of steps to take to move toward COBOL/VSE

IBM COBOL Products—A History

COBOL/VSE builds upon the functions that IBM introduced with DOS/VS COBOL and VS COBOL II. Figure 7 gives an overview of the functions available with the last releases of DOS/VS COBOL and VS COBOL II. In addition, Figure 7 shows that the new functions added by COBOL/VSE include the following:

- Intrinsic functions
- Language extensions, such as the new procedure-pointer support
- Support for LE/VSE, which provides: improved ILC, common services, condition handling, and tuning at run time
- A general-use programming interface for access to compilation data in the SYSADAT file (via the ADATA compiler option)

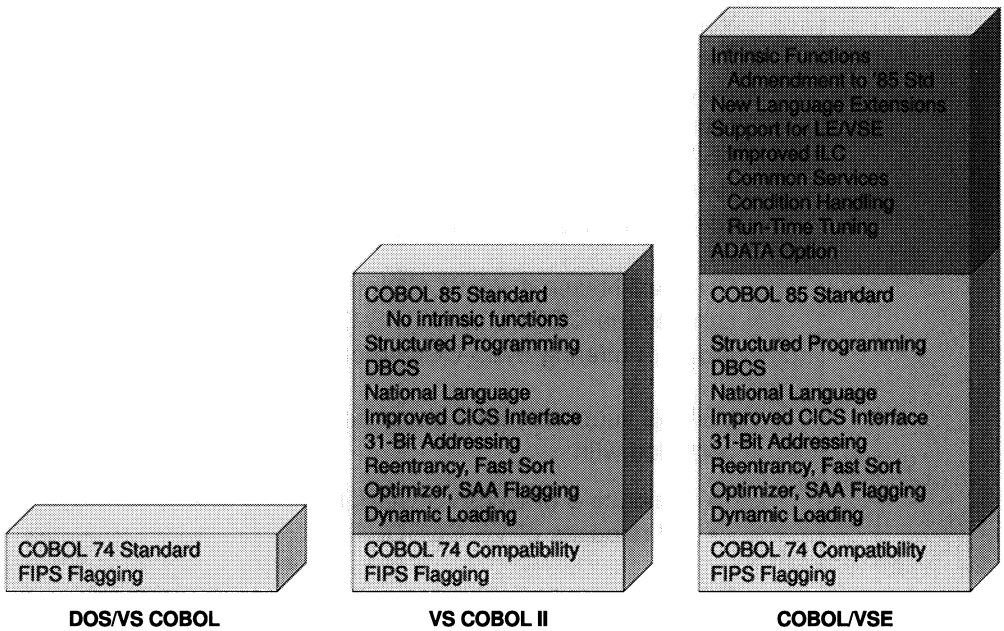


Figure 7. COBOL-A History and COBOL/VSE-The Future

Because COBOL/VSE offers many advantages over previous COBOL products, you may want to begin preparing now to use COBOL/VSE for your organization's applications. First, consider which operating systems and subsystems you can use with COBOL/VSE.

What Hardware and Software Environments Are Supported?

You can run COBOL/VSE application programs in all environments supported by LE/VSE. With LE/VSE, you can run COBOL/VSE generated object programs under, or with, the following products:

- VSE/ESA Version 1 Release 4
- VSE/ESA Version 2 Release 1
- CICS/VSE Version 2 Release 3
- DL/I DOS VS Version 1 Release 10
- DFSORT/VSE Version 3 Release 1
- DOS/VS Sort/Merge II Version 2 Release 5
- High Level Assembler/MVS & VM & VSE Release 1
- High Level Assembler/VSE Release 2
- SQL/DS (VSE) Version 3 Release 4

IBM Language Environment for VSE/ESA Release 1 will run on the System/370 and its follow-on machines supporting the above programming systems.

Will Existing Applications Still Run?

Yes, you can run DOS/VS COBOL and VS COBOL II programs in the new run-time environment; in fact, DOS/VS COBOL, VS COBOL II, and COBOL/VSE programs can all coexist. You can have a single application that mixes programs from all three compilers.

COBOL/VSE provides upward source program compatibility with VS COBOL II Version 1 Release 4 for VSE with certain necessary exceptions that are detailed in the *COBOL/VSE Migration Guide*. This means that, in general, you can take an existing COBOL source program that runs under VS COBOL II Release 4 or later, recompile it without change under COBOL/VSE, and run it successfully under LE/VSE. Or, you can take the source program, make whatever changes you want to make, including taking advantage of new COBOL/VSE language function, compile it under COBOL/VSE, and run it under LE/VSE.

For pre-Release 4 VS COBOL II programs and for DOS/VS COBOL programs, you can use the COBOL and CICS Command Level Conversion Aid for VSE (CCCA) program offering 5785-CCC to convert the program source to COBOL/VSE source.

LE/VSE supports existing and new programs, as either phases or object modules, compiled with:

- Release 3 of DOS/VS COBOL
- All releases of VS COBOL II

In most cases both phase and object module compatibility are provided for existing COBOL code that is run under LE/VSE. Therefore, you can continue to run your old COBOL phases under LE/VSE, without recompiling and link-editing the programs. The phases may consist of COBOL compiled code or assembler code that is written to the standards documented in the *COBOL/VSE Programming Guide*. Also, you can link-edit existing COBOL object modules or assembler object modules with the LE/VSE library, and then run them under LE/VSE and get equivalent results.

Note: If any part of a phase is recompiled with COBOL/VSE, the phase must be relinked with the LE/VSE library.

What Migration Aids Are Available?

COBOL/VSE supports a number of compiler options that help you migrate, as described in "Support for Migration, Compatibility, and Conformance to Standards" on page 33.

In addition, you may want to use one of these IBM products that can help smooth migration to COBOL/VSE:

Table 11. Migration Aids

Migration Product	Description
COBOL and CICS Command Level Conversion Aid for VSE (CCCA) Program Offering 5785-CCC	Converts DOS/VS source code or CICS COBOL programs into COBOL/VSE source code.
IBM COBOL Structuring Facility Version 3 (COBOL/SF) Program Product 5696-737	Running under MVS or VM, COBOL/SF transforms unstructured DOS/VS COBOL or VS COBOL II to structured COBOL/VSE programs, automatically invoking conversion using CCCA. Please note that COBOL/SF is **not available under VSE**.
COBOL Report Writer Precompiler Program Offering 5798-DYR	Converts DOS/VS Report Writer statements into COBOL/VSE source code or runs as a preprocessor for COBOL/VSE. This tool supplements CCCA source code conversions.

For more information about these migration features and how you might use them in your migration process, see *COBOL/VSE Migration Guide*.

Where Can I Get More Information to Help Me Use or Move toward COBOL/VSE?

You can find helpful information in the COBOL/VSE and LE/VSE publication libraries.

The structure of these libraries is designed to support the basic programming tasks shown in the following figure:

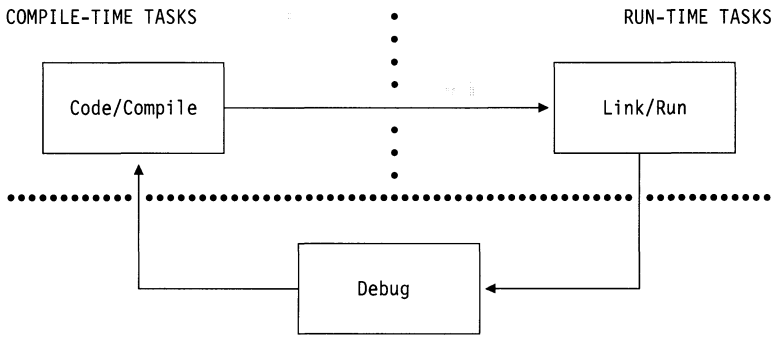


Figure 8. Division of Programming Tasks by Library

Bear in mind that:

- Publications supporting coding and compiling tasks are available with the language products you use.
- Publications supporting linking and running tasks are available with LE/VSE

There are exceptions to this method of dividing tasks, but it is a good model to use in finding information. Note also that interlanguage communication is discussed primarily in the LE/VSE publications.

Publications Provided with COBOL/VSE

Publications provided with the COBOL/VSE product include the following:

Table 12. IBM COBOL for VSE/ESA Publications

Task	Publication	Order number
Evaluation and Planning	*General Information*	GC26-8068
	Migration Guide	GC26-8070
	Installation and Customization Guide	SC26-8071
Programming	*Programming Guide*	SC26-8072
	Language Reference	SC26-8073
	Reference Summary	SX26-3834
Diagnosis	*Diagnosis Guide*	SC26-8528
Warranty	*Licensed Program Specifications*	GC26-8069

COBOL/VSE General Information
: Contains high-level information designed to help you evaluate the COBOL/VSE product. This book describes new compiler and language features, application development with LE/VSE, and product support for industry standards.

COBOL/VSE Migration Guide
: Contains detailed migration and compatibility information for current users of DOS/VS COBOL and VS COBOL II who wish to migrate to, or reuse their existing applications on, COBOL/VSE. This book also describes several migration aids or tools to help you plan a migration path for your installation.

COBOL/VSE Installation and Customization Guide

Provides information you will need in order to install and customize the COBOL/VSE product in VSE/ESA or CICS/VSE environment. Detailed planning information includes:

- Systems and storage requirements for COBOL/VSE
- Information about changing compiler option defaults

COBOL/VSE Programming Guide

Contains guidance information for writing and compiling application programs using COBOL/VSE, including information on the following topics:

- Programming using new product features, such as intrinsic functions
- Processing techniques for VSAM and SAM files
- Debugging techniques using compiler options and listings
- Nested programming techniques
- Subsystem considerations

COBOL/VSE Language Reference

Provides syntax and semantic information about the implementation of the COBOL language, including rules for writing source programs and descriptions of IBM language extensions. This book is meant to be used in conjunction with the COBOL/VSE Programming Guide, which provides programming task-oriented information.

COBOL/VSE Reference Summary

Contains a convenient summary of the COBOL/VSE language syntax—including new intrinsic functions—and syntax for compiler options, compiler-directing statements, and the COBOL/VSE reserved word list.

COBOL/VSE Diagnosis Guide

Provides instructions for diagnosing failures in the COBOL/VSE compiler product that are not caused by user error. This book will help you construct a keyword string that allows you or IBM Service to search the product failure database for previously documented problems and appropriate corrections.

COBOL/VSE Licensed Program Specifications

Contains a product description and product warranty information for the COBOL/VSE compiler.

Other Publications: LE/VSE

The publications provided with the LE/VSE product are listed in Table 13.

Table 13. IBM Language Environment for VSE/ESA Publications

Task	Publication	Order number
Evaluation and Planning	*Fact Sheet*	GC26-8062
	Concepts Guide	GC26-8063
	Installation and Customization Guide	SC26-8064
Programming	*Programming Guide*	SC26-8065
	Debugging Guide and Run-Time Messages	SC26-8066
Diagnosis	*Diagnosis Guide*	SC26-8060
Warranty	*Licensed Program Specifications*	GC26-8069

LE/VSE Fact Sheet
Provides a brief overview and description of LE/VSE, and COBOL/VSE.

LE/VSE Concepts Guide
Provides a detailed overview of program models and intended architecture for LE/VSE, the common run-time environment.

LE/VSE Installation and Customization Guide
Contains information needed to plan for installing and customizing the LE/VSE product.

LE/VSE Programming Guide
Provides detailed information on the following topics:

- Directions for linking and running programs that use LE/VSE services
- Information on storage management, run-time message handling, and condition handling models
- Callable services and run-time options and how to use them
- Instructions for writing programs that use interlanguage communication (ILC)

This book also contains language-specific run-time information.

LE/VSE Debugging Guide and Run-Time Messages
Provides detailed information on the following topics:

- Debugging techniques and services
- Run-time messages and their explanation
- Abend codes

LE/VSE Diagnosis Guide
Describes the procedures for creating a keyword string and reporting errors to IBM Service.

LE/VSE Licensed Program Specifications
Contains a product description and warranty information.

Softcopy Information Available with LE/VSE

The *LE/VSE Programming Guide* is also available in softcopy.

Additional Publications

Other publications that may help you move toward the COBOL/VSE product include:

COBOL and CICS Command Level Conversion Aid for VSE
: Provides detailed information about how to use the CCCA migration aid.

COBOL Structuring Facility User's Guide and Reference
: Gives the programming rules for using COBOL/SF.

COBOL Report Writer Installation and Operation Manual (VSE)
: Describes the basic functions of the precompiler, and how to install and customize it. The book also discusses various debugging problems that may arise from its use.

COBOL Report Writer Precompiler Programmers Manual
: Provides the information needed by application programmers engaged in the writing or maintenance of COBOL programs using Report Writer.

What Steps Do I Take to Get Ready to Move toward COBOL/VSE?

It is easy to get started. Just take these steps:

Read this book, and others

You have already started preparing for COBOL/VSE because you are reading this book: *COBOL/VSE General Information*. In addition, consider reading:

LE/VSE Concepts Guide
: To understand the new concepts introduced by LE/VSE.

LE/VSE Installation and Customization Guide
: To understand how to install and customize LE/VSE.

COBOL/VSE Installation and Customization Guide
: To understand how to plan for installing and customizing COBOL/VSE.

COBOL/VSE Migration Guide
: To understand the detailed requirements for migrating applications to COBOL/VSE.

Other books, as required:
: To understand more about the products you are planning to install. See "Where Can I Get More Information to Help Me Use or Move toward COBOL/VSE?" on page 41 for a list of books to consider reading.

Order products, as necessary

You can place an order for COBOL/VSE and LE/VSE, and one or more of the migration products listed in "What Migration Aids Are Available?" on page 41.

Plan, by application, how you will migrate

Consider dividing your organization's applications into these categories:

- Those you plan to run in compatibility mode; that is, those applications that, at least initially, you want to run without modification and without recompiling and relinking.
- Those you plan to modify to take advantage of one or more new COBOL/VSE features.
- New applications to be developed completely with COBOL/VSE.

Install products

Before you install the products you have chosen, follow the instructions given in *COBOL/VSE Installation and Customization Guide*, *LE/VSE Installation and Customization Guide*, and *COBOL/VSE Migration Guide*. Be sure to run several tests to make sure that the products are installed correctly.

Begin to use COBOL/VSE

After you have installed COBOL/VSE and LE/VSE, you can begin to reap some benefits immediately without disruption to existing applications. Gradually, as you migrate applications to COBOL/VSE, and create new applications to take advantage of COBOL/VSE's new features, you can continue to reap more and more benefits to help you streamline the entire application development process.

Appendix A. Industry Standards

COBOL/VSE supports the following industry standards in the VSE/ESA environment, as understood and interpreted by IBM as of September 1989:

1. ISO 1989:1985, Programming Languages - COBOL

   ISO 1989/Amendment 1, Programming Languages - COBOL - Amendment 1: Intrinsic function module.

   ISO 1989:1985 is identical to X3.23-1985, American National Standard for Information Systems - Programming Language - COBOL.
   ISO 1989/Amendment 1 is identical to X3.23a-1989, American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL.

   For supported modules, see item 2 below.

2. X3.23-1985, American National Standard for Information Systems - Programming Language - COBOL.

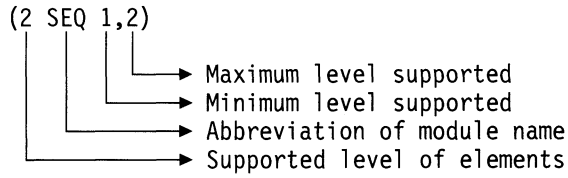
   X3.23a-1989, American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL.

   COBOL/VSE supports all required modules at the intermediate level. It also supports all required modules at the high level with the exception of the following language features:

   - EXTEND phrase of the OPEN statement
   - REVERSED phrase of the OPEN statement
   - OF/IN phrase of the COPY statement

   See Level 2 Restrictions below.

   In the following list, the shorthand notation describing module levels is shown in parentheses. For example, to summarize module information for sequential input/output, the shorthand notation is (2 SEQ 1,2).



   - Nucleus (2 NUC 1,2)
   - Sequential I-O (1 SEQ 1,2) file.

     Level 2 restriction: the EXTEND phrase of the OPEN statement is not supported except for VSAM sequential files.

     Level 2 restriction: the REVERSED phrase of the OPEN statement does not cause file positioning, and is only applicable to tape files.

   - Relative I-O (2 REL 0,2)
   - Indexed I-O (2 INX 0,2)
   - Sort-Merge (1 SRT 0,1)
   - Inter-Program Communication (2 IPC 1,2)
   - Source Text Manipulation (1 STM 0,2)

Level 2 restriction: the OF/IN phrase of the COPY statement is treated as documentation.

In addition, the following levels of optional modules are supported:

- Intrinsic Functions (1 ITR 0,1)
- Debug (1 DEB 0,2)
- Segmentation (2 SEG 0,2)

The following optional modules of the standard are not supported:

- Report Writer
- Communications
- Debug (2 DEB 0,2)

3. FIPS Publication 21-3, Federal Information Processing Standard 21-3 , COBOL high subset.
4. International Reference Version of the ISO 7-bit code defined in *International Standard 646, 7-Bit Coded Character Set for Information Processing Interchange*.
5. The 7-bit coded character sets defined in *American National Standard X3.4-1977, Code for Information Interchange*.

Under CICS, the following language elements of X3.23-1985 are not supported:

ACCEPT
CLOSE
DELETE
DISPLAY
MERGE
OPEN
READ
RERUN
REWRITE
SORT which requires COBOL to perform I/O
START
STOP 'literal'
WRITE
USE declaratives (except USE FOR DEBUGGING)
ENVIRONMENT DIVISION and FILE SECTION when entries relate to data management.

NOTES:

The term "COBOL 85 Standard" is used in this book to refer to the combination of the following standards:

- ISO 1989:1985, Programming Languages - COBOL

  ISO 1989/Amendment 1, Programming Languages - COBOL - Amendment 1: Intrinsic function module.

- X3.23-1985, American National Standard for Information Systems - Programming Language - COBOL.

  X3.23a-1989, American National Standard for Information Systems - Programming Language - Intrinsic Function Module for COBOL.

The term "COBOL 74 Standard" is used in this book to refer to the following standards:

- X3.23-1974, American National Standard for Information Systems - Programming Language - COBOL.
- ISO 1989:1978, Programming Languages - COBOL

Note: The ISO Standards are equivalent to the American National Standards.

The following options are **required** to support the above standards:

Compiler Options LE/VSE Run-Time Options

ADV	AIXBLD
DYNAM	TRAP(ON)
FLAGSTD(H)	
LIB	
NOCMPR2	
NOCURRENCY	
NODBCS	
NOFASTSRT	
NOFLAGMIG	
NOFLAGSAA	
NONUMBER	
NOSEQUENCE	
NUMPROC(NOPFD) or NUMPROC(MIG)	
QUOTE	
TRUNC(STD)	
NOWORD	
ZWB	

The following LE/VSE run-time options are used in support of the standards: UPSI, DEBUG, and NODEBUG.

A complete description of the COBOL/VSE language is available in *COBOL/VSE Language Reference*.

Appendix B. COBOL/VSE Intrinsic Functions

Note: This appendix contains general-use programming interfaces and associated guidance information.

Table 14 provides an overview of the argument type, function type and value returned for each of the Intrinsic Functions provided by COBOL/VSE.

Argument types and function types are abbreviated as follows:

A = alphabetic
I = integer
N = numeric
X = alphanumeric

Table 14 (Page 1 of 3). Table of Functions

Function-name	***Arguments***	***Type***	***Value returned***
ACOS	N1	N	Arccosine of N1
ANNUITY	N1, I2	N	Ratio of annuity paid for I2 periods at interest of N1 to initial investment of one
ASIN	N1	N	Arcsine of N1
ATAN	N1	N	Arctangent of N1
CHAR	I1	X	Character in position I1 of program collating sequence
COS	N1	N	Cosine of N1
CURRENT-DATE	None	X	Current date and time and difference from Greenwich Mean Time
DATE-OF-INTEGER	I1	I	Standard date equivalent (YYYYMMDD) of integer date
DAY-OF-INTEGER	I1	I	Julian date equivalent (YYYYDDD) of integer date
FACTORIAL	I1	I	Factorial of I1
INTEGER	N1	I	The greatest integer not greater than N1
INTEGER-OF-DATE	I1	I	Integer date equivalent of standard date (YYYYMMDD)
INTEGER-OF-DAY	I1	I	Integer date equivalent of Julian date (YYYYDDD)
INTEGER-PART	N1	I	Integer part of N1

Table 14 (Page 2 of 3). Table of Functions

Function-name	*Arguments*	*Type*	*Value returned*
LENGTH	A1 or N1 or X1	I	Length of argument
LOG	N1	N	Natural logarithm of N1
LOG10	N1	N	Logarithm to base 10 of N1
LOWER-CASE	A1 or X1	X	All letters in the argument are set to lowercase
MAX	A1... or I1...or N1...or X1...	X I N X	Value of maximum argument; note that the type of function depends on the arguments
MEAN	N1...	N	Arithmetic mean of arguments
MEDIAN	N1...	N	Median of arguments
MIDRANGE	N1...	N	Mean of minimum and maximum arguments
MIN	A1... or I1...or N1...or X1...	X I N X	Value of minimum argument; note that the type of function depends on the arguments
MOD	I1,I2	I	I1 modulo I2
NUMVAL	X1	N	Numeric value of simple numeric string
NUMVAL-C	X1 X1,X2	N	Numeric value of numeric string with optional commas and currency sign
ORD	A1 or X1	I	Ordinal position of the argument in collating sequence
ORD-MAX	A1... or N1...or X1...	I	Ordinal position of maximum argument
ORD-MIN	A1...or N1...or X1...	I	Ordinal position of minimum argument
PRESENT-VALUE	N1 N2...	N	Present value of a series of future period-end amounts, N2, at a discount rate of N1

Table 14 (Page 3 of 3). Table of Functions

Function-name	***Arguments***	***Type***	***Value returned***
RANDOM	I1 or none	N	Random number
RANGE	I1... or N1...	I N	Value of maximum argument minus value of minimum argument; note that the type of function depends on the arguments.
REM	N1,N2	N	Remainder of N1/N2
REVERSE	A1 or X1	X	Reverse order of the characters of the argument
SIN	N1	N	Sine of N1
SQRT	N1	N	Square root of N1
STANDARD-DEVIATION	N1...	N	Standard deviation of arguments
SUM	I1... or N1...	I N	Sum of arguments; note that the type of function depends on the arguments.
TAN	N1	N	Tangent of N1
UPPER-CASE	A1 or X1	X	All letters in the argument are set to uppercase
VARIANCE	N1...	N	Variance of arguments
WHEN-COMPILED	None	X	Date and time when program was compiled

Bibliography

Language Environment Publications

IBM Language Environment for VSE/ESA

Fact Sheet, GC26-8062

Concepts Guide, GC26-8063

Installation and Customization Guide, SC26-8064

Programming Guide, SC26-8065

Debugging Guide and Run-Time Messages, SC26-8066

Diagnosis Guide, SC26-8060

Licensed Program Specifications, GC26-8061

Reference Summary, SX26-3835

LE/VSE-Conforming Language Product Publications

IBM 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

Reference Summary, SX26-3834

Diagnosis Guide, SC26-8528

Licensed Program Specifications, GC26-8069

IBM PL/I for VSE/ESA

Fact Sheet, GC26-8052

Programming Guide, SC26-8053

Language Reference, SC26-8054

Licensed Program Specifications, GC26-8055

Migration Guide, SC26-8056

Installation and Customization Guide, SC26-8057

Diagnosis Guide, SC26-8058

Reference Summary, SX26-3836

Compile-Time Messages and Codes, SC26-8059

Related Publications

COBOL and CICS Command Level Conversion Aid for VSE, SC26-8269

COBOL Structuring Facility MVS and VM User's Guide, SC26-3278

COBOL Structuring Facility Reference Version 3.1, SC26-3411

COBOL Structuring Facility Getting Started Version 3.1, SC26-3415

COBOL Report Writer Precompiler Installation and Operation, SC26-4864

COBOL Report Writer Precompiler Programmer's Manual, SC26-4301

Softcopy Publications

These collections contain the COBOL/VSE and LE/VSE-conforming language product publications:

VSE Collection, SK2T-0060

Application Development Collection, SK2T-1237

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

Index

Numerics

A

B

C

D

E

F

G

H

I

We'd Like to Hear from You

IBM COBOL for VSE/ESA
General Information
Release 1

Publication No. GC26-8068-00

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773.
- Electronic mail—Use one of the following network IDs:
  - IBMMail: USIB2VVG at IBMMAIL
  - IBMLink: COBPUBS at STLVM27
  - Internet: COMMENTS@VNET.IBM.COM

  Be sure to include the following with your comments:
  - Title and publication number of this book
  - Your name, address, and telephone number if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

IBM COBOL for VSE/ESA
General Information
Release 1

Publication No. GC26-8068-00

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name _____ Address _____

Company or Organization _____

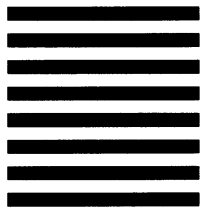
Phone No. _____

Readers' Comments
GC26-8068-00

Cut or Along

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

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Department J58
International Business Machines Corporation
PO BOX 49023
SAN JOSE CA 95161-9945

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

Cut or Along

Readers' Comments

IBM COBOL for VSE/ESA
General Information
Release 1

Publication No. GC26-8068-00

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name

Address

Company or Organization

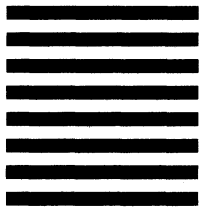
Phone No.

Readers' Comments
GC26-8068-00

Cut or
Along l

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

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Department J58
International Business Machines Corporation
PO BOX 49023
SAN JOSE CA 95161-9945

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

Cut or
Along

Program Number: 5686-086

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber

IBM COBOL for VSE/ESA Publications

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

GC26-8068-00