

IBM VisualAge PL/I for OS/390



Compiler and Run-Time Migration Guide

Version 2 Release 2.1

IBM VisualAge PL/I for OS/390



Compiler and Run-Time Migration Guide

Version 2 Release 2.1

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix, "Notices" on page 37.

Second Edition (September 2000)

This edition applies to Version 2 Release 2.1 of &prodva. for OS/390, part number 04L7217; and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

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

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

IBM Corporation, Department BWE/H3
P.O. Box 49023
San Jose, CA, 95161-9023
United States of America

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

Chapter 1. Introduction	1
Run-time environment for VisualAge PL/I	1
Using your documentation	2
Chapter 2. Installation considerations	3
Product information	3
Considerations for using assembler user exits	4
Specific considerations	4
Considerations for using high-level language user exits	4
Chapter 3. Compile-time considerations	5
Mixing Object Levels	5
Dependency on Language Environment	5
Compile-time options not supported by VisualAge PL/I	6
Compatibility considerations and restrictions	6
OS PL/I Version 1 source code	6
ENTRY statement	7
Array expressions	7
Structure expressions	8
DEFINED variables	8
DBCS	8
Stream I/O	8
Record I/O	9
Built-in functions	9
Batch compilations	9
Miscellaneous unsupported elements	10
Storage report changes	10
Compiler messages	10
Messages that PL/I issues for errors in the PLIXOPT string	10
Chapter 4. Link-edit considerations	12
Using FETCH in your routines	12
Using PLICALLA or PLICALLB entry	12
ENTRY CEESTART requirement	12
Chapter 5. Run-Time Considerations	13
Differences in PLICALLA and PLICALLB Support	13
PLICALLA considerations	13
PLICALLB considerations	14
Differences in preinitialization support	16
Differences in DATE/TIME built-in functions	16
Differences in user return codes	17
Differences in Condition Handling	17
Differences in run-time messages	19
Differences in PLIDUMP	20
Differences in run-time options	21
Differences in storage report	22
Differences in interlanguage communication support	22
Differences in assembler support	23
Differences in language element behavior	24

Differences in Descriptor Format	25
Differences in AMODE(24) Support	25
Chapter 6. Tuning your VisualAge PL/I program	26
Improving CPU utilization	26
Improving storage utilization	27
Improving performance under subsystems	27
Chapter 7. Subsystem considerations	29
CICS considerations	29
Updating CICS System Definition (CSD) file	29
Macro-level interface	29
SYSTEM(CICS) compile-time option	29
FETCHing a PL/I MAIN procedure	30
STACK run-time option	30
Run-time output	30
Abend codes used by PL/I under CICS	30
Linking VisualAge PL/I applications	30
IMS considerations	30
Interfaces to IMS	31
SYSTEM(IMS) compile-time option	31
PLICALLA Support in IMS	31
PSB language options supported	31
Assembler driving a PL/I transaction	32
Storage usage considerations	32
Coordinated condition handling under IMS	32
Performance enhancement with Library Retention(LRR)	33
DB2 considerations	33
Chapter 8. OS PL/I coexistence with Language Environment	34
Coexistence under OS/390 non-CICS	34
Coexistence under OS/390 CICS	35
Coexistence under DB/2	36
Appendix. Notices	37
Programming interface information	38
Trademarks	38
Bibliography	39
Bibliography	40
VisualAge PL/I publications	40
DB2 Version 2	40
DATABASE 2	40
VisualAge CICS Enterprise Application Development	40
Index	41

Chapter 1. Introduction

This book contains information to help you migrate applications from previous releases of PL/I to VisualAge PL/I and OS/390 Language Environment. It suggests solutions to problems that arise because of differences in support between previous releases of PL/I (OS PL/I and PL/I for MVS & VM) and VisualAge PL/I.

If you need information on support issues for previously compiled PL/I programs, see *PL/I for MVS & VM Compiler and Run-Time Migration Guide*.

Note: There is no VM support for IBM VisualAge PL/I for OS/390.

IMPORTANT

The information in this book discusses migration considerations using VisualAge PL/I V2R2M1 and OS/390 V2R8 Language Environment or later. These two products must be installed in order to take advantage of the migration enhancements discussed in this book. The use of VisualAge PL/I will always refer to Version 2 Release 2.1 unless indicated otherwise. The use of Language Environment will always refer to OS/390 V2R8 Language Environment or later unless indicated otherwise.

This book is for system programmers, application programmers, and IBM support personnel who are involved in PL/I product migration. Prerequisite knowledge for using this book is:

- A general understanding of your operating system
- Some knowledge of the PL/I language and options
- Some knowledge of how PL/I uses Language Environment for its run-time environment

Run-time environment for VisualAge PL/I

VisualAge PL/I uses Language Environment as its run-time environment. It conforms to Language Environment architecture and shares the run-time environment with other conforming languages such as C/370, C/C++, COBOL, and Fortran.

Language Environment is the common run-time environment for the following language compilers:

C/370
C/C++
COBOL for MVS & VM
COBOL for OS/390 & VM
Fortran
PL/I for MVS & VM
VisualAge PL/I

It provides a common set of run-time options and callable services. It also improves interlanguage communication (ILC) between high-level languages (HLL) and assembler by eliminating language-specific initialization and termination on each ILC invocation. Language Environment provides compatibility support for existing applications with a few restrictions.

Using your documentation

The publications provided with VisualAge PL/I are designed to help you program with PL/I. The publications provided with Language Environment are designed to help you manage your run-time environment for applications generated with VisualAge PL/I. Each publication helps you perform a different task.

The following tables show you how to use the publications you receive with VisualAge PL/I and Language Environment. You'll want to know information about both your compiler and run-time environment. For the complete titles and order numbers of these and other related publications, see "Bibliography" on page 40.

PL/I information

Table 1. How to use VisualAge PL/I publications

To...	Use...
Evaluate VisualAge PL/I	Fact Sheet
Understand warranty information	Licensed Programming Specifications
Plan for and install VisualAge PL/I	VisualAge PL/I Program Directory
Understand compiler and run-time changes and adapt programs to VisualAge PL/I and Language Environment	Compiler and Run-Time Migration Guide
Prepare and test your programs and get details on compiler options	Programming Guide
Get details on PL/I syntax and specifications of language elements	Language Reference
Diagnose compiler problems and report them to IBM	Diagnosis Guide
Get details on compile-time messages	Compile-Time Messages and Codes

Language Environment information

Table 2. How to use OS/390 Language Environment publications

To...	Use...
Evaluate Language Environment	Concepts Guide
Plan for Language Environment	Concepts Guide Run-Time Migration Guide
Install Language Environment on OS/390	OS/390 Program Directory
Customize Language Environment on OS/390	Customization
Understand Language Environment program models and concepts	Concepts Guide Programming Guide
Find syntax for Language Environment run-time options and callable services	Programming Reference
Develop applications that run with Language Environment	Programming Guide and your language Programming Guide
Debug applications that run with Language Environment, get details on run-time messages, diagnose problems with Language Environment	Debugging Guide and Run-Time Messages
Develop interlanguage communication (ILC) applications	Writing Interlanguage Applications
Migrate applications to Language Environment	Run-Time Migration Guide and the migration guide for each Language Environment-enabled language

Chapter 2. Installation considerations

This chapter contains product information you need to know at installation time. It discusses differences in user exits and effects of Language Environment Abnormal Termination Exit.

The Language Environment run-time options that you might want to consider at installation time are described in “Differences in run-time options” on page 21.

Product information

VisualAge PL/I has renamed its parts so that, if you want to, you can install it in the same SMP/E zone as OS PL/I. To help you identify the elements of each product, the following table lists the name differences:

Table 3. PL/I element names

OS PL/I	PL/I for MVS & VM	VisualAge PL/I
IEL0AA	IEL1AA	IBMZPLI
IKJEN00n	IEL1IKJn	
IEL0nn	IEL1nn	IBMZnn
PLInnnnn	IEL1Mnnn	IBMZMnnn
PLIXnnn	IEL1nnn	IBMZnnn
PLIHELP	IEL1PLIH	IBMZPLIH

Language Environment must be available before you can compile, prelink, link-edit, and run a VisualAge PL/I application. If you attempt to compile a program before installing Language Environment, the program will not compile and a message will be generated. A STEPLIB concatenation for SCEERUN must be added in the compile. The details of the data sets and modules shipped with VisualAge PL/I and Language Environment can be found in one of the documents listed below. If you want to know the names of the data sets and modules, storage requirements, or other details specifically for installation planning, refer to one of these documents:

- VisualAge PL/I Program Directory
- OS/390 Program Directory
- OS/390 V2R8 Language Environment Customization

There are additional requirements you need to be aware of before you begin to use VisualAge PL/I and Language Environment.

You must have access to Language Environment when you compile your VisualAge PL/I application. When you compile your application and you use existing JCL, be sure your STEPLIB or JOBLIB statement includes SCEERUN (Language Environment run-time library). You can use the IBMZC cataloged procedure to compile PL/I applications.

Your compile step should include the following:

```
//PLI      EXEC PGM=IBMZPLI,REGION=512K
//STEPLIB DD DSN=&LNGPRFX..SIBMZCMP,DISP=SHR
//        DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
```

Reading about the cataloged procedures provided with VisualAge PL/I can help you understand the use of SCEERUN during compilation. “Using PL/I Cataloged Procedures” is a chapter in *VisualAge PL/I for OS/390 Programming Guide*.

When you link-edit your VisualAge PL/I application with Language Environment and you use existing JCL, be sure your SYSLIB statement includes SCEELKED (Language Environment link-time library). Language Environment also provides the SIBMCAL2 library to which you can link specific PL/I functions and compatibility support. You can use this library as long as it is specified before SCEELKED.

You must specify SYSLIB if you plan to use it. Do not include SYSLIB unless you are using a LINKLIB which already includes SCEELKED.

Considerations for using assembler user exits

The only Assembler user exit supported by VisualAge PL/I is the Language Environment user exit CEEBXITA. IBMBXITA and IBMFXITA are not supported. For a detailed parameter description for CEEBXITA, see *OS/390 Language Environment Programming Guide*.

Specific considerations

- The PL1DUMP, PLIDUMP or CEEDUMP file for the dump output is treated as a process resource and must not be cleared during enclave termination.
- The OS PL/I abend exit IBMBEER is ignored under Language Environment. See “Differences in Condition Handling” on page 17 for forcing an abend under Language Environment.

For more information on assembler language user exits, see *OS/390 Language Environment Programming Guide*.

Considerations for using high-level language user exits

The OS PL/I Version 2 High-Level Language (HLL) user exit IBMBINT is not supported. VisualAge PL/I MAIN load module supports only CEEBINT. The load module always contains a copy of CEEBINT, either the application-specific one or the default one provided by Language Environment.

If you write CEEBINT in PL/I, you must write it in VisualAge PL/I. If CEEBINT calls any PL/I routines, those routines must also be written in VisualAge PL/I.

Do not use the OPTIONS(MAIN) statement in the PL/I HLL user exit.

The STOP statement terminates the application.

Chapter 3. Compile-time considerations

This chapter describes compile-time consideration when your run-time environment is Language Environment. You'll find out what OS PL/I Version 1 source code is supported in VisualAge PL/I. Also discussed are the changes affecting compiler messages.

The major factors to consider before migrating to VisualAge PL/I are:

- There is no VM support.
- Consider recompiling your OS PL/I and PL/I for MVS & VM applications with VisualAge PL/I. Support for combining object and load modules from previous PL/I releases with VisualAge PL/I object is limited.
- Multitasking is not supported; however, multithreading is supported.
- There is no support for compiles under TSO.

Mixing Object Levels

Support for mixing objects produced by VisualAge PL/I and previous versions in a single application is limited. If you attempt to mix old and new objects, it is strongly recommended that you use the following compiler options:

- CMPAT(V2) (or CMPAT(V1) if your old code was compiled with that)
- DEFAULT(LINKAGE(SYSTEM))
- LIMITS(EXTNAME(8))
- NORENT

The CMPAT(V2) (or CMPAT(V1)) option is also needed where a string, array or structure is passed to or received from non-PL/I code. This occurs, for example, when a DB2 stored procedure written in PL/I is passed a string.

The following are not supported:

- shared files
- shared controlled variables
- shared entry variables unless the NORENT compiler option is used
- passing structures or arrays unless the CMPAT(V1) or the CMPAT(V2) compiler option is used

Dependency on Language Environment

Language Environment must be available whenever you compile a PL/I application. Language Environment is the run-time environment for the VisualAge PL/I compiler.

Compile-time options not supported by VisualAge PL/I

VisualAge PL/I does not support the following compile-time options:

CONTROL	NOINCLUDE
DECK	SEQUENCE
ESD	SIZE
FLOW	SMESSAGE
LMESSAGE	

The following compile-time options have had the indicated suboptions dropped:

LANGLVL	NOSPROG/SPROG (SPROG is always in effect)
LIST	m,n
SYSTEM	CMS, CMSTPL

The meaning of the following compile-time options has changed:

INCLUDE	The old meaning of INCLUDE (enabling %INCLUDE statements without use of the macro preprocessor) is always in effect in VisualAge PL/I. The new INCLUDE option is used under OS/390 UNIX System Services to help the compiler find the right include file.
OFFSET	This option no longer produces an offset table in the listing, instead, this option determines how offsets are presented in the listing: NOOFFSET produces offsets from the compile unit, and OFFSET produces offsets from the procedure.

Compatibility considerations and restrictions

There are some behavioral differences between the VisualAge PL/I compiler and previous PL/I compilers. As a result, the VisualAge PL/I compiler imposes some restrictions or may produce different results than when using source code created for OS PL/I or PL/I for MVS & VM. These differences are described in the following sections.

OS PL/I Version 1 source code

VisualAge PL/I compatibility with OS PL/I Version 1 source code is supported with the following exceptions:

- CHARSET(48) and CHARSET(BCD) are no longer supported. Support for these options were dropped by OS PL/I Version 2; however, there is an IBM-supplied tool that will convert the source.
- Graphic DBCS varies slightly from old EGCS in that the shift-in and shift-out code points are fixed.
- Suffixes that follow string constants are not replaced by the preprocessor—whether or not these are legal PL/I suffixes—unless you insert a delimiter between the ending quotation mark of the string and the first letter of the suffix. For example:

```

%DCL (GX, XX) CHAR;
%GX=' ||FX';
%XX=' ||ZZ';
DATA = 'STRING'GX;
DATA = 'STRING'XX;
DATA = 'STRING' GX;
DATA = 'STRING' XX;

```

under Version 1 produces the source:

```

DATA = 'STRING' ||FX;
DATA = 'STRING' ||ZZ;
DATA = 'STRING' ||FX;
DATA = 'STRING' ||ZZ;

```

whereas, under VisualAge PL/I it produces:

```

DATA = 'STRING'GX;
DATA = 'STRING'XX;
DATA = 'STRING' ||FX;
DATA = 'STRING' ||ZZ;

```

- CHECK statement, CHECK prefix, and CHECK condition support was dropped by PL/I for MVS & VM.

ENTRY statement

The ENTRY statement is supported with the following restrictions:

- All parameters must be BYADDR. The default compiler options will force this action.
- RETURNS must be BYADDR if an aggregate. The default compiler options will force this action.
- Return value, if any, will be converted to the attributes in the most recent PROC or ENTRY statement with a RETURNS option.

The following example shows how returns value might be converted:

```

a: proc; /* return value is undefined */
  b: entry returns(fixed bin); /* return value converted to fixed bin */
  c: entry; /* return value converted to fixed bin */
  d: entry returns(fixed dec); /* return value converted to fixed dec */
end;

```

Array expressions

An array expression is not allowed as an argument to user functions unless it's an array of scalars of known length.

The following example shows numeric array expressions supported in calls:

```

dcl x entry, (y(10),z(10)) fixed bin(31);

call x(y + z);

```

The following unprototyped call would be flagged since it requires a string expression of unknown size:

```
dc1 a1 entry;  
dc1 (b(10),c(10)) char(20) var;
```

```
call a1(b || c);
```

However, the following prototyped call would not be flagged:

```
dc1 a2 entry(char(30) var);  
dc1 (b(10),c(10)) char(20) var;
```

```
call a2(b || c);
```

Structure expressions

Structure expressions are supported in assignment statements, including BY NAME assignments, with the following exceptions:

- Structure expressions as arguments are not supported unless both of the following conditions are true:
 - There is a parameter description.
 - The parameter description specifies all constant extents.

DEFINED variables

Support for iSUB defining is limited to arrays of scalars.

Simple DEFINED variables are supported only for the following:

- Scalars
- Structures with constant extents matching those in the base variable
- Arrays of such scalars and structures as long as the array is not based on a controlled variable

When simple defining does not apply, string-overlay defining is assumed and flagged with an E-level message (as is true with PLIOPT).

DBCS

DBCS can be used only in the following:

- G and M constants
- Identifiers
- Comments

G literals can start and end with a DBCS quote followed by either a DBCS G or an SBCS G.

Stream I/O

Stream I/O is supported with the following restrictions:

- For PUT/GET DATA, the following restrictions apply:
 - DEFINED variables are not supported if the DEFINED variable is BIT or GRAPHIC or has a POSITION attribute.
 - DEFINED variable is not supported if its base variable is an array slice or an array with a different number of dimensions than the defined variable.

Record I/O

Record I/O is supported with the following exceptions:

- EVENT clauses on READ/WRITE are not supported.
- UNLOCK statement is not supported.
- The BACKWARDS file attribute is not supported.

Built-in functions

Built-in functions are supported with the following exceptions/restrictions:

- The PLITEST built-in function is not supported.
- Pseudovariabes are not supported in:
 - The STRING options of PUT statements
- Pseudovariabes in DO loops are restricted to:
 - IMAG
 - REAL
 - SUBSTR
 - UNSPEC
- The POLY built-in function has the following restrictions:
 - The first argument must be REAL FLOAT.
 - The second argument must be scalar.
- The COMPLEX pseudovariabes is not supported.

Batch compilations

Compilation is not performed in PROCESS-delimited chunks and the following results occur:

- Options on later sets of PROCESS statements are ignored
- One TEXT deck or .o is produced
- One listing file with one set of messages is produced
- External variables with the same name must match

The following example demonstrates a batch compilation. In this case, the mismatches in **b** and **x** would be flagged.

```

*process opt(0);

a: proc;
  dcl b ext entry(1,2 char(2), 2 char(2));
  dcl
    1 x ext,
    2 x1a char(2),
    2 x1b char(2);

  call b(x);
end;

*process opt(2);

b: proc(p);
  dcl p pointer;
  dcl
    1 x ext,
    2 x1a bit(16),
    2 x1b bit(16);

end;

```

Miscellaneous unsupported elements

The following miscellaneous elements are not supported:

- NOMAP, NOMAPIN, and NOMAPOUT are accepted but ignored, even if parmlist/arguments are given.

Storage report changes

The PLIXHD variable is no longer used as the heading in storage reports. The identifier PLIXHD is no longer reserved; you can declare it and use it as you would declare and use any other variable.

Compiler messages

The messages issued by the VisualAge PL/I compiler are completely different from messages issued by previous PL/I compilers. For detailed descriptions of messages produced by VisualAge PL/I, see *VisualAge PL/I Compile-Time Messages and Codes*.

Messages that PL/I issues for errors in the PLIXOPT string

The PLIXOPT variable is a varying-length character string that contains run-time options you can specify at compile time. The messages that the compiler produces to diagnose errors in these options have changed. In most cases, the PL/I messages now list an associated Language Environment message that you should read for more information about the error.

PL/I parses the PLIXOPT string and produces the Language Environment CEEUOPT CSECT. If you explicitly include CEEUOPT in your recompiled application ahead of the compiler-generated CEEUOPT CSECT, the explicitly

included CEEUOPT CSECT overrides the one generated by the compiler for the options specified in the PLIXOPT string.

Chapter 4. Link-edit considerations

This chapter describes factors you must consider when you link-edit an object module produced by VisualAge PL/I.

Important: If you link text decks using the binder, you must place the output into a PDSE file. To use a PDS file, the prelinker must be used. But for all new styles objects, must either be all prelinked together to the PDS file or using the binder to place into the PDSE file. For more information about linking, see *VisualAge PL/I for OS/390 Programming Guide*, in the chapter titled 'Link-editing and Running.'

Using FETCH in your routines

You can FETCH VisualAge PL/I routines, OS/390 C DLLs, and Assembler routines except for the following restrictions:

- If PL/I is fetching another VisualAge PL/I routine, it must not specify OPTIONS(ASM) or OPTIONS(COBOL) in the declaration of that ENTRY; however, if it is fetching a COBOL or Assembler routine, then it should specify OPTIONS(COBOL) or OPTIONS(ASM), respectively.
- OPTIONS(FETCHABLE) must be specified on the PROCEDURE statement for the entry point of the DLL.
- PROCEDURE statements for a non-CICS program specifying OPTIONS(FETCHABLE) must be linked as a DLL. DYNAM=DLL must be specified when linking a fetchable module into a DLL. The ENTRY name in the linkage editor control statements should be specified as the fetched program's name.

For a detailed description of these restrictions, see *VisualAge PL/I for OS/390 Programming Guide*, in the chapter titled 'Link-editing and Running.'

Using PLICALLA or PLICALLB entry

For VisualAge PL/I programs that use PLICALLA or PLICALLB as the main entry point, link-edit the object modules with the SIBMICAL2 dataset in front of the SCEELKED dataset. See "PLICALLA considerations" on page 13 and "PLICALLB considerations" on page 14 for details.

ENTRY CEESTART requirement

If a VisualAge PL/I or PL/I for MVS & VM main procedure is link-edited with object modules produced by other language compilers or by assembler, and is the first module to receive control, the user must ensure that the entry point of the resulting executable program is resolved to the external symbol CEESTART. This happens automatically if the VisualAge PL/I or PL/I for MVS & VM main procedure is first in the input to the linkage editor. Run-time errors occur if the executable program entry point is forced to some other symbol by use of the linkage editor ENTRY control statement.

Chapter 5. Run-Time Considerations

Before you migrate to Language Environment or VisualAge PL/I, you should read this chapter. It discusses the functional differences between previous PL/I compilers and VisualAge PL/I and its run-time environment Language Environment. These differences should be considered before you install Language Environment or VisualAge PL/I. Other chapters in this book discuss differences you must consider during and after installation.

Factors to consider before migrating to VisualAge PL/I are:

- There is no support for IBMBSIR or IBMBHKS.
- There is no support for multitasking.

It is possible to install Language Environment together with the OS PL/I library. If you'd like to know more about this topic, see Chapter 8, "OS PL/I coexistence with Language Environment" on page 34.

Differences in PLICALLA and PLICALLB Support

The interfaces in the following sections are not recommended for use in VisualAge PL/I. They are supported only for compatibility reasons.

PLICALLA considerations

Language Environment provides support for VisualAge PL/I applications that use the PLICALLA entry point. It also provides support for recompiled OS PL/I and PL/I for MVS & VM applications that want to continue to use PLICALLA as the primary entry point. When you recompile your OS PL/I or PL/I for MVS & VM program with VisualAge PL/I, there is no need to INCLUDE Language Environment-provided PLISTART CSECT when you link your main load module. You just need to make sure the SIBMCAL2 dataset is concatenated in front of the SCEELKED dataset. If you don't do this, the linkage editor or loader issues an error message for an unresolved ENTRY PLICALLA statement.

You can also use PLICALLA as the primary entry point of a FETCHed/CALLED main load module; however, the calling routine must pass only user arguments which are passed to a subroutine. If run-time options are passed, they are treated as user arguments.

If you develop a new application in VisualAge PL/I and you want the main procedure to receive user arguments like a subroutine, do one of the following:

- Receive control directly from IMS by:
 - Using CEESTART as the primary entry point of the load module.
 - Specifying the SYSTEM(IMS) compile-time option.
- Receive control from an assembler program or a procedure using a FETCH or CALL statement by:
 - Using CEESTART as the primary entry point of the load module.
 - Specifying the NOEXECOPS option and the SYSTEM(MVS) compile-time option.

- Specifying either the BYADDR option or the BYVALUE option.

Language Environment support of PLICALLA is not available in the following environments:

- CICS environment
- Preinitialized environment
- Nested enclave environment except the PL/I FETCHable main.

Passing parameters

OPTIONS(BYADDR) passes the argument indirectly by reference and is the usual argument-passing convention. VisualAge PL/I also provides OPTIONS(BYVALUE) which passes arguments directly by value.

If the main procedure is recompiled with VisualAge PL/I using SYSTEM(CICSIMS), only POINTER data type can be passed as parameters. If a main procedure receives control from assembler via PLICALLA and is recompiled with VisualAge PL/I, the main procedure cannot be compiled with SYSTEM(CICSIMS).

Table 4 provides the expected argument passing convention (either BYADDR or BYVALUE) when the main procedure of your OS PL/I PLICALLA application is recompiled with VisualAge PL/I:

Table 4. Parameter passing for the main procedure compiled with VisualAge PL/I

System environment	Invoked from IMS ¹	Invoked from assembler program ²	Invoked by PL/I FETCH/CALL Statement ²
SYSTEM(MVS)	BYADDR	BYADDR	BYADDR
SYSTEM(CICS)	BYVALUE ³	Not supported	Not supported
SYSTEM(IMS)	BYVALUE ³	Not supported	Not supported
SYSTEM(TSO)	BYADDR	BYADDR	BYADDR

¹LANG=PL/I must be specified and it passes indirect by reference.

²It must have already passed indirect by reference or by value.

³PL/I library will convert the argument list to direct by value.

PLICALLB considerations

Language Environment provides support for recompiled OS PL/I or PL/I for MVS & VM PLICALLB applications that continue to use PLICALLB as the primary entry point. The following list shows the PLICALLB parameter mapping for Language Environment:

- Address of argument list (argument must either point to an address or be zero)
- Address of the length of ISA storage mapped to STACK(*init_size*)
- Address of ISA storage used as the initial STACK segment
- Address of the options word in which the run-time options for a program are specified. These options are: RPTSTG, TRAP, HEAP(,KEEPIFREE)|(,ANYIBELOW). The hexadecimal value for each option is defined as follows in the assembler program:

```

OPTIONS DC AL1(RPTSTG+TRAP,FREEHEAP+ANYHEAP,0,0)
*
RPTSTG     EQU X'80'
RPTSTGOFF  EQU X'40'
TRAP       EQU X'20'
TRAPOFF    EQU X'10'
*
KEEPHEAP   EQU X'20'
FREEHEAP   EQU X'10'
ANYHEAP    EQU X'08'
BELHEAP    EQU X'04'

```

- Address of HEAP storage length for a program is mapped to HEAP(*init_size*)
- Address of HEAP storage is used as the initial HEAP segment
- Address of HEAP increment for a program is mapped to HEAP(*incr_size*)
- Address of ISA increment for a program is mapped to STACK(*incr_size*) (optional) is mapped to NONIPTSTACK(*incr_size*)

When the above argument list is passed in via the PLICALLB entry point, the argument in the list must either point to an address or be zero. The high-order bit of an argument must be ON to indicate the end of the argument list. R1 must contain the address of the argument list.

With Language Environment, the run-time options passed via the PLICALLB entry point are processed as options specified on invocation of the application and have a higher precedence than CEEUOPT or PLIXOPT options. The assembler user exit cannot be used to alter the run-time options passed through the PLICALLB invocation.

To summarize, the run-time options passed in have the following precedence (from highest to lowest) among Language Environment option specification methods:

1. Options specified via the PLICALLB entry point
2. Options specified in the PLIXOPT string or in CEEUOPT

The user arguments passed to the PL/I main routine have the following precedence (from highest to lowest):

1. Output from CXIT_PARM or AUE_PARM of the assembler user exit
2. User arguments passed in via the PLICALLB entry

Note: The input to CXIT_PARM or AUE_PARM of the assembler user exit is the first argument in the PLICALLB parameter list; that is, the address of a vector of user argument addresses.

Language Environment encourages the use of above-16M-line storage. For compatibility with OS PL/I, Language Environment maps the user-supplied ISA and HEAP storage to STACK and HEAP. With this mapping, however, Language Environment still needs to issue some GETMAINS. Since user-supplied ISA/HEAP storage is usually below the 16M line, below-16M-line storage can be quickly consumed under Language Environment. How Language Environment manages storage is described in the *OS/390 Language Environment Programming Guide*.

Language Environment manages storage differently than OS PL/I. It divides storage into more categories than the OS PL/I-supported ISA and HEAP.

Language Environment allocates below-16M-line storage using the `init_sz24` and `incr_sz24` suboptions specified in the HEAP option.

When you develop new applications in VisualAge PL/I and want to pass both run-time options and arguments to a PL/I main procedure, especially to provide user-supplied stack and heap storage from an assembler program, take advantage of Language Environment's preinitialization support as described in *OS/390 Language Environment Programming Guide*.

Language Environment support of PLICALLB is not available in the following environments:

- CICS
- IMS
- Preinitialized environment
- Nested enclave environment

Passing parameters

OPTIONS(BYADDR) passes the argument indirectly by reference and is the usual argument-passing convention. VisualAge PL/I also provides OPTIONS(BYVALUE) which passes arguments directly by value.

You must use the BYADDR option when you want to pass parameters using PLICALLB. PLICALLB is invoked from assembler which passes the argument list indirectly by reference.

Table 5 provides the expected argument passing convention (either BYADDR or BYVALUE) when the main procedure of your OS PL/I PLICALLB application is recompiled with VisualAge PL/I:

Table 5. Parameter passing for the main procedure compiled with VisualAge PL/I

System environment	Invoked from assembler program ¹
SYSTEM(MVS)	BYADDR
SYSTEM(CMSICMSTPL)	BYADDR
SYSTEM(CICS)	Not supported
SYSTEM(IMS)	Not supported
SYSTEM(TSO)	BYADDR

¹It passed the argument list required by the PLICALLB entry.

Differences in preinitialization support

The PL/I preinitialized program interface is not supported for VisualAge PL/I applications. Use the Language Environment preinitialization service as described in the *OS/390 Language Environment Programming Reference*.

Differences in DATE/TIME built-in functions

The DATETIME and TIME built-in functions now return the number of milliseconds in all environments. The syntax and description of these built-in functions are in the *VisualAge PL/I for OS/390 PL/I Language Reference*.

Differences in user return codes

VisualAge PL/I and Language Environment support a FIXED BIN(31) four-byte user return code value for PLIRETC, PLIRETV, and OPTIONS(RETCODE). This support removes the restriction of maximum value 999.

The following table shows how PL/I user return code is supported:

Table 6. Return code behavior under Language Environment

Function	OS PL/I load module	OS PL/I object module linked with Language Environment	VisualAge PL/I load module
PLIRETC built-in function	2-byte value with restriction of 999	4-byte value without restriction of 999	4-byte value without restriction of 999
PLIRETV built-in function	2-byte value	Lower 2 bytes of a 4-byte value	4-byte value
RETCODE option	Lower 2 bytes of R15	Lower 2 bytes of R15	2-byte value

For PLIRETC, VisualAge PL/I and relinked OS PL/I load modules can set a 4-byte user return code value.

For PLIRETV and RETCODE, only VisualAge PL/I load modules can receive a 4-byte user return code value.

Under Language Environment, the PL/I user return code is always reset to zero upon return from the PLISRTx invocation. This is not the case with OS PL/I run-time.

Differences in Condition Handling

PL/I condition handling semantics remain supported under Language Environment; however, the timing of issuing the run-time message for an ERROR condition with respect to the ERROR ON-unit is different in the following way:

- The run-time message for an ERROR condition is issued only if there is no ERROR ON-unit established, or if the ERROR ON-unit does not recover from the condition by using a GOTO out of the ERROR ON-unit. Thus you can use a GOTO out of the ERROR ON-unit to avoid a message for a PL/I ERROR condition.

For other PL/I conditions whose implicit action includes printing a message and raising the ERROR condition, the message is issued before control is given to an established ERROR ON-unit.

Table 7 on page 18 shows when the run-time message for an ERROR condition is issued under OS PL/I with respect to the ERROR ON-unit.

Table 7. OS PL/I Version 2 Release 3 ERROR ON-unit and message for an ERROR condition

Condition	No ON-units	ERROR ON-unit No GOTO	ERROR ON-unit GOTO
ERROR condition raised ¹	Message	Message prior to ON-unit	Message prior to ON-unit
ZERODIVIDE condition raised ²	Message	Message prior to ON-unit	Message prior to ON-unit

Notes:

- 1 Taking the square root of a negative number, data exception, etc.
- 2 With no ZERODIVIDE ON-unit; thus, implicit action is taken. Message is printed, ERROR condition is raised.

Table 8 shows when the run-time message for an ERROR condition is issued under Language Environment with respect to the ERROR ON-unit.

Table 8. Language Environment ERROR ON-unit and message for an ERROR condition

Condition	No ON-units	ERROR ON-unit No GOTO	ERROR ON-unit GOTO
ERROR condition raised ¹	Message	Message after ON-unit	No message
ZERODIVIDE condition raised ²	Message	Message prior to ON-unit	Message prior to ON-unit

Notes:

- 1 Taking the square root of a negative number, data exception, etc.
- 2 With no ZERODIVIDE ON-unit; thus, implicit action is taken. Message is printed, ERROR condition is raised.

The SNAP traceback message produced by ON ERROR SNAP continues to be issued before the ERROR ON-unit receives control. The SNAP traceback message is not identical to the regular ERROR message.

Some program return code modifiers have changed under Language Environment, depending upon what compiler was used. The behaviors are:

- **OS PL/I V2R3:**

A return code of 2000 is added for the case where the ERROR condition is raised and the program terminates without returning from an ERROR or FINISH ON-unit.

- **VisualAge PL/I and PL/I for MVS & VM:**

A return code of 3000 is added for severity 3 conditions (severe error—abnormal termination).

Most PL/I conditions are severity 3, with the following severity 1 exceptions: ENDPAGE, FINISH, NAME, PENDING, STRINGRANGE, STRINGSIZE, UNDERFLOW, ATTENTION signaled, CONDITION signaled.

Note: This information is useful when using the Language Environment ERRCOUNT run-time option.

If your OS PL/I application used to force an abend for an unhandled condition under OS PL/I run-time using OS PL/I assembler user exit IBMBXITA or abend exit IBMBEER, use the following ways to force an abend under Language Environment:

- Run your application with the Language Environment ABTERMENC(ABEND) option. You cannot specify your own abend code via the run-time option.
- Use Language Environment assembler user exit CEEBXITA to force an abend with your own abend code.

For ZERODIVIDE, OVERFLOW, and SIZE, the ERROR condition is raised if the condition goes unhandled.

The FOFL condition is not raised for FIXED BIN. It is raised only for FIXED DEC and decimal PICTURE.

Language Environment provides limited support for OS PL/I IBMBXITA and IBMBEER. See “Considerations for using assembler user exits” on page 4 for details.

An UNHANDLED condition of severity 2 or higher now produces an abend U4039 and optionally a system dump if SYSUDUMP or SYSABEND ddname is present. If ABTERMENC(RETCODE) is in effect, your application continues the termination with an abend code. If you don't want to see the U4039 abend, Language Environment provides you the facilities to suppress it.

See “Abnormal Termination Exit” in *OS/390 Language Environment Customization* for ways to suppress or change the U4039 abend.

Differences in run-time messages

The format and content of run-time messages are different. If you have applications that analyze run-time messages, you must change the applications to allow for the differences. The differences include:

- The message number in the message prefix is four digits instead of three digits in the form IBMnnnx, where nnnn represents the message number and x represents the severity of the message.
- The message severity in the message prefix can be I, W, E, S, or C.
- The message text of some messages has been enhanced.

Details are provided in *OS/390 Language Environment Debugging Guide and Run-Time Messages*.

Under Language Environment, run-time messages go to the MSGFILE destination specified in the run-time option MSGFILE. The default for MSGFILE destination is SYSOUT. The user output still goes to SYSPRINT. MSGFILE(SYSPRINT) is not supported under VisualAge PL/I. For more information about the MSGFILE option, refer to *OS/390 Language Environment Programming Guide*.

Under Language Environment, run-time messages give offset values that are relative to the start of the external procedure, rather than relative to the start of the block that contains the statement. You can use these offsets to help you find the statement that is in error. To do this, match the offset provided in the message with the offset given in the pseudo-assembler listing that the compiler produces when you specify the LIST compile-time option.

Differences in PLIDUMP

PLIDUMP now produces a Language Environment-style dump. The way you use PLIDUMP and the dump output is different. The following list contains the differences in the way you use PLIDUMP and the output produced. *Compile unit* refers to the primary entry point of the external procedure and *Compile unit name* refers to the name of the external procedure.

- The ddname of the dump output file can be CEEDUMP, PLIDUMP, or PL1DUMP. If you do not define one of these files, Language Environment creates a default CEEDUMP file to contain the dump output. The LRECL of the dump output file must be at least 133 bytes to prevent dump records from wrapping, not the 121 bytes required by OS PL/I. If you write the dump output to the SYSOUT file, make sure you change the default LRECL size of 121 to 133 to prevent from wrapping. Use LRECL of 137 for variable-length files.

- When you use the hexadecimal (H) option of PLIDUMP, you must specify the ddname CEESNAP; otherwise, no SNAP dump will be produced.

When you specify the hexadecimal (H) option under OS/390, the output from SNAP includes all system control program information (SDATA=ALL). OS PL/I provides only partial information (SDATA=CB, Q, and TRT).

- When you use ILC, the dump output contains information related to other languages (for example, C/C++ or COBOL).
- The identifier character string is limited to 60 bytes rather than the 90 bytes OS PL/I supported.
- The traceback section lists the compile-unit name associated with each entry point name. When the entry point is a secondary entry point, the primary entry point name associated with the actual entry point is not listed.

The traceback section also contains offsets relative to the address of the *compile unit*, as well as offsets relative to the address of the real entry point.

- Run-time messages are in a separate section; they are no longer part of the traceback section.
- When you specify the BLOCK (B) option of PLIDUMP, the condition handler save areas appear in the block section of the dump. If you do not specify the BLOCK option of PLIDUMP, the condition handler save areas do not appear in the dump.
- If the program was compiled with the TEST compile-time option, the BEGIN blocks that are ON-units are identified as `_ON_Begin_line_Blk_number` while other BEGIN blocks are identified as `_Begin_line_Blk_number` where line is the line number where the Begin block begins and number is block block for the begin-block.
- PL/I library routines are identified by name in the dump.
- Assembler routines that conform to the rules for mimicking PL/I routines are identified by their CSECT names in the dump output.
- PLIDUMP now conforms to National Language Support standards.
- PLIDUMP can supply information across multiple Language Environment enclaves. For example, if an application running in one enclave FETCHes a main procedure (an action that creates another enclave), PLIDUMP contains information about both procedures.

Differences in run-time options

Language Environment run-time options replace OS PL/I run-time options. Most OS PL/I run-time options have an equivalent Language Environment run-time option that provides the same function. This section describes differences in the use of run-time options.

Pre-Language Environment storage was initialized to zero. By default Language Environment does not do this and it can be a problem for programs with uninitialized variables. One way to handle this situation is to use the run-time option STORAGE by using the third parameter to initialize all storage to zero. Note that the use of this method has serious performance costs, and modifying the program so that all variables are initialized is the preferred solution.

You should adapt your applications to allow for the following differences:

- The Language Environment ABTERMENC option controls which type of return/abend code your application receives at abnormal termination. ABTERMINC(RETCODE) allows your application to receive a run-time return code, which is equivalent to the way OS PL/I worked.
- The OS PL/I COUNT option is ignored.
- The Language Environment ERRCOUNT option limits the number of errors that are handled at run-time. ERRCOUNT(0) specifies that there is no limit, which is equivalent to the way OS PL/I worked.
- The Language Environment DEPTHCONDLMT option limits the extent to which conditions can be nested. To maintain compatibility, specify DEPTHCONDLMT(0), which means there is an unlimited depth.
- The OS PL/I FLOW option is ignored.
- The OS PL/I HEAP option is always in effect. This means that when you allocate storage for BASED and CONTROLLED variables, the storage always comes from HEAP storage. The storage does not come from a PL/I Initial Storage Area (ISA). HEAP(0) is not supported and, if used, is ignored.
- The Language Environment NATLANG option replaces the OS PL/I LANGUAGE option.
- The Language Environment RPTSTG option replaces the OS PL/I REPORT option.
- The Language Environment TRAP option replaces both OS PL/I SPIE and STAE options. The following table shows how the OS PL/I SPIE and STAE options map to Language Environment's TRAP option:

Table 9. Mapping of SPIE and STAE options to the TRAP option

OS PL/I	Language Environment	Action
SPIE NOSPIE	TRAP(ON,SPIE)	If either SPIE or STAE is specified or defaulted in input, TRAP is set to TRAP(ON,SPIE). If both NOSPIE and NOSTAE are specified, TRAP is set to TRAP(OFF). TRAP(ON,SPIE) is the recommended setting.
STAE NOSTAE	TRAP(OFF)	

Note: Applications performing their own condition management often conflict with Language Environment condition management. See your *OS/390*

Language Environment Programming Guide for more information on Language Environment condition handling.

- The Language Environment STACK option replaces both OS PL/I ISASIZE and ISAINC options. You do not need to change source code that contains ISASIZE and ISAINC. In addition, object modules and/or load modules containing the PLIXOPT string will run under Language Environment with the ISASIZE and ISAINC honored as before.

Use STACK(,,ANY) for your VisualAge PL/I application. Your application must run in AMODE(31) to use STACK(,,ANY).

Under CICS, ALL31(ON) and STACK(,,ANY) are the defaults.

- The Language Environment Environment XUFLOW option determines if the UNDERFLOW condition is raised when underflow occurs. XUFLOW(AUTO) preserves PL/I semantics with regard to raising the UNDERFLOW condition.

For more information about run-time options, see the *OS/390 Language Environment Programming Reference*.

For OS PL/I applications, the options specified in the PLIXOPT string are processed as the application-specific options. Do not mix PLIXOPT and CEEUOPT.

Differences in storage report

The format, contents, and destination of the run-time storage report have changed. Language Environment provides storage information equivalent to OS PL/I. The details of storage report is described in *OS/390 Language Environment Programming Reference*.

The PLIXHD declaration is no longer used to provide the heading for the run-time storage report. Instead, use Language Environment's Callable Service, CEE3RPH, to specify the heading.

Differences in interlanguage communication support

Recompilation of PL/I modules in ILC applications containing OS PL/I or PL/I for MVS & VM with VisualAge PL/I is recommended.

ILC between VisualAge PL/I and the following languages is not supported:

- Fortran (prior to Language Environment Release 5)
- OS/VS COBOL
- VS COBOL II Version 1 Release 2 or earlier releases

For more information, see *Language Environment for OS/390 & VM Writing Interlanguage Communication Applications*.

Under VisualAge PL/I, all routines are reentrant (even if they modify static variables). All procedures must be recompiled with VisualAge PL/I.

OPTIONS(COBOL) is treated like OPTIONS(ASM). There is no remapping of parameters via MAPIN or MAPOUT. This is both a compile-time difference and a run-time difference, but it will become apparent only at run-time.

The behavior of certain applications that use ILC might be different. For example:

- Condition handling might behave differently. The major causes of differences in condition handling are that the INTER option is now ignored, and that PL/I condition handling facilities can deal with conditions occurring in non-PL/I routines whether or not you specify INTER.
- Under OS PL/I, in applications that used ILC, the environment initialization and termination of the involved languages, including PL/I, could occur multiple times. With Language Environment, there is only one run-time environment, and language-specific initialization and termination occurs only once. Changes in behavior that you might see include opening and closing of files, releasing of allocated storage, and invocation of established ON-units.

Note: If you have designed your own code to manage your run-time environments, you should remove it as part of your migration efforts. This *private* code is incompatible with Language Environment and will conflict with the run-time environment.

For a complete description of how ILC works in the Language Environment run-time environment, see *Language Environment for OS/390 & VM Writing Interlanguage Communication Applications*.

Differences in assembler support

With VisualAge PL/I, the object module contains the CSECT name CEESTART. It also contains CEEMAIN if it has OPTIONS(MAIN) or CEEFMAIN if it has OPTIONS(FETCHABLE). VisualAge PL/I no longer produces PLISTART and PLIMAIN CSECTs. CEESTART, CEEMAIN, and CEEFMAIN are not supported as a standard entry point and you cannot call them directly from an assembler program. You can call CEESTART from an assembler program only when it is a CSECT name of a VisualAge PL/I routine statically linked with an assembler program. Therefore, any assembler program mimicking a OS PL/I main procedure (calling PLISTART directly as a standard entry point), must continue to use PLISTART under Language Environment.

With Language Environment, assembler programs that call a PL/I routine must follow the calling conventions defined by Language Environment. For example, Register 13 pointing to a save area, save areas properly back-chained, and the first word of the save area being zero. For detailed information, see *OS/390 Language Environment Programming Guide*.

If your OS PL/I main program is called by an assembler program and you want to convert your assembler program to use Language Environment-conforming assembler, you must recompile your OS PL/I program with VisualAge PL/I without OPTIONS(MAIN). The called VisualAge PL/I program is treated as a subroutine and runs under the same Language Environment enclave where the assembler program is the main program and the called VisualAge PL/I program is a subroutine.

Your Language Environment-conforming assembler main program must explicitly include the Language Environment-VisualAge PL/I signature CSECT, CEESG011, when calling a PL/I subroutine to ensure the Language Environment-PL/I-specific run-time environment is initialized. There are three ways Language

Environment-conforming assembler routines can pass control to a VisualAge PL/I subroutine:

1. Branch to a statically-linked VisualAge PL/I subroutine.
2. Use the Language Environment macro CEEFETCH to branch to a separately-linked VisualAge PL/I subroutine.
3. Use assembler instructions such as LOAD and BALR to branch to a separately-linked VisualAge PL/I subroutine.

When you use method 1 or 2 with VisualAge PL/I, you don't need to include CEESG011 with your assembler program. If your assembler program uses instructions as described in method 3, you must always include CEESG011 with your assembler program.

Condition handling of the LINK from assembler is now clearly defined. For detailed information, see *OS/390 Language Environment Programming Guide* and *VisualAge PL/I Programming Guide*.

Differences in language element behavior

There are also some language elements that can cause your program to run differently under VisualAge PL/I than it does under PL/I for MVS & VM due to differences in the hardware or in the implementation of the language by the compiler. Each of the following items is described in terms of its VisualAge PL/I behavior.

FIXED BIN(p) maps to one byte if p <= 7

If you have any variables declared as FIXED BIN with a precision of 7 or less, they occupy one byte of storage under VisualAge PL/I instead of two as under PL/I for MVS & VM and earlier. If the variable is part of a structure, this usually changes how the structure is mapped, and that could affect how your program runs. For example, if the structure were read in from a file created on the mainframe, fewer bytes would be read in on the workstation than would be on the mainframe.

To avoid this difference, you could change the precision of the variable to a value between 8 and 15 (inclusive).

INITIAL attribute for AREAs is ignored

To keep VisualAge PL/I products from ignoring the INITIAL attribute for AREAs, convert INITIAL clauses into assignment statements.

For example, in the following code fragment, the elements of the array are not initialized to a1, a2, a3, and a4:

```
dc1 (a1,a2,a3,a4) area;  
dc1 a(4) area init( a1, a2, a3, a4 );
```

However, you can rewrite the code as follows so that the array is initialized as desired:

```
dc1 (a1,a2,a3,a4) area;  
dc1 a(4) area;
```

```
a(1) = a1;  
a(2) = a2;  
a(3) = a3;  
a(4) = a4;
```

ADD, DIVIDE, and MULTIPLY do not return scaled FIXED BIN

Under the RULES(IBM) compile-time option, which is the default, variables can be declared as FIXED BIN with a nonzero scale factor. Infix, prefix, and comparison operations are performed on scaled FIXED BIN as with the mainframe. However, when the ADD, DIVIDE, or MULTIPLY built-in functions have arguments with nonzero factors or specify a result with a nonzero scale factor, the VisualAge PL/I, PL/I for MVS & VM and earlier compilers evaluate the built-in function as FIXED DEC rather than as FIXED BIN as the mainframe compiler.

For example, the workstation compilers would evaluate the DIVIDE built-in function in the assignment statement below as a FIXED DEC expression:

```
dc1 (i,j) fixed bin(15);
dc1 x      fixed bin(15,2);
      :
x = divide(i,j,15,2);
```

Differences in Descriptor Format

The descriptor format in VisualAge PL/I is different from previous versions of PL/I. The CMPAT(V2) and CMPAT(V1) compiler options are available to cause VisualAge PL/I to use the previous descriptor format. This is particularly useful, for example, in applications where DB2 passes strings to VisualAge PL/I, or where Assembler programs pass descriptors in the previous format, or where Assembler programs directly read and expect the old descriptor format.

Differences in AMODE(24) Support

AMODE(31) and RMODE(ANY) are the default settings for the VisualAge PL/I application. Amode(24) applications are only supported for compatibility reasons. To use the AMODE(24) feature, the application program has to be compiled with the PL/I compiler option NORENT and run with the Language Environment option ALL31(OFF).

Chapter 6. Tuning your VisualAge PL/I program

After you migrate to Language Environment, you should retune your applications to maximize the performance. When you retune an application, it is not always possible to maximize CPU and storage at the same time. Often you will find that, in order to obtain better CPU, you need to use more storage, or vice versa. This chapter provides general tips to help you to retune your applications under Language Environment.

For more information on tools you can use to improve performance for your applications, see *OS/390 Language Environment Customization* or *OS/390 V2R8 Language Environment Customization*, and *VisualAge PL/I for OS/390 Programming Guide*.

Improving CPU utilization

The following discussion shows ways to help you obtain better CPU utilization:

- Reduce the number of GETMAINS and FREEMAINS issued by Language Environment.

Use the Language Environment RPTSTG(ON) option to produce the storage report. Specify the reported storage amount in the corresponding Language Environment storage run-time options.

- Reduce the number of LOADs and DELETEs issued by Language Environment.

Put the commonly used Language Environment library routines in (E)LPA. The following lists the recommended candidates for VisualAge PL/I:

- CEEBINIT (LPA)
- CEEPLPKA (ELPA)
- CEEEV011 (ELPA)

See *OS/390 Language Environment Customization* for a complete list of library routines that can be put in (E)LPA.

- Avoid AMODE switching between library routines.

Use AMODE(31) for your application, if possible, so you can specify Language Environment ALL31(ON) option. If ALL31(ON) is in effect, there is no AMODE switching among library routines.

Avoid the following PL/I conditions because they might cause a slower performance to your application:

- STRINGSIZE
- AREA
- OVERFLOW

- Use DFSMS®-provided system-determined BLKSIZE.

On OS/390, use BLKSIZE(0) for an output file that can be blocked. DFSMS determines the optimal block size for you which can improve the file performance.

- Use Language Environment Library Routine Retention facility (LRR).

You can get a better CPU performance if you use LRR. When LRR is used, Language Environment keeps certain Language Environment resources in storage when an application ends. Subsequent invocations of programs that use LRR is much faster because the Language Environment resources left in storage are reused.

For example, you can use LRR for your IMS/DC environment to improve performance.

Note that because LRR leaves LE resources in the storage for a long period of time, you must assess your storage availability to accommodate the situation.

Improving storage utilization

The following discussion helps you to obtain better storage utilization:

- Use Language Environment option HEAP(,ANY) option, if possible.

For VisualAge PL/I, Language Environment will allocate the heap storage above the 16M line if the following is true:

- The requestor is in AMODE(31)
- HEAP(,ANY) is in effect
- The main program is in AMODE(31)

- Use Language Environment STACK(,ANY) option, if possible.

Your application must be in AMODE(31). Language Environment allocates the stack storage above the 16M line when your application is recompiled with VisualAge PL/I and linked with Language Environment.

- Reduce the IBM-supplied default values in Language Environment storage options.

If you use a smaller value, Language Environment will allocate less storage each time, but it could result in more GETMAINS and FREEMAINS being issued.

- Put commonly used Language Environment library modules in (E)LPA.

The library routines in (E)LPA do not occupy storage in your application region, so your application has more storage to use. See the recommended library routines for (E)LPA in “Improving CPU utilization” on page 26.

Improving performance under subsystems

The following discussion helps you to obtain better performance under specific subsystems:

- Under CICS

Use the PL/I FETCH/CALL statement instead of EXEC CICS LINK. The PL/I FETCH/CALL statement has a much shorter path length than the path length of EXEC CICS LINK.

- Under IMS

Use Language Environment Library Routine Retention (LRR) facility to reduce the number of LOADS/DELETES and GETMAINS/FREEMAINS issued by Language Environment for each transaction.

Preload commonly used Language Environment library modules and frequently used top-level applications.

Chapter 7. Subsystem considerations

This chapter discusses subsystem-specific considerations that you need to know when you migrate your applications running under CICS, IMS, and DB2.

CICS considerations

Language Environment provides the same level of OS PL/I object and load module support as for non-CICS, if you are running your OS PL/I or PL/I for MVS & VM programs with Language Environment. Before running VisualAge PL/I with Language Environment, you must first recompile and relink your source code.

The CICS Storage Protect facility was introduced under CICS 3.3. This provides more data integrity and security for the application program and especially for the entire CICS region. Because of the new feature, you might discover that some of the successfully running OS PL/I applications start to fail with ASRA(0C4) abend and the CICS message DFHSR0622.

If the above problem is happening in your VisualAge PL/I application program, set the CICS system initialization parameter `RENTPGM=NOPROTECT`. This sets the protection of the user program in user key. The default for `RENTPGM` is `PROTECT`.

If the stream output function is used in your OS PL/I CICS application, especially the `PUT DATA` statement, it might trigger the above error. PL/I stream output function is intended for debugging purposes only. For performance reasons, we recommend that you don't use it in production programs.

Updating CICS System Definition (CSD) file

When you bring up a CICS region with Language Environment, you must ensure the module names listed in Language Environment `CEECCSD` are defined in the CSD. You can locate `CEECCSD` in `SCEESAMP`. If you use CICS Version 4 autoinstall facility, you do not need to define Language Environment modules manually in the CSD.

In order to run a VisualAge PL/I CICS application, you need to define the VisualAge PL/I member event handler `CEEEV011` in the CICS CSD definition table:

```
DEFINE PROGRAM(CEEEV011) GROUP(CEE) LANGUAGE(ASSEMBLER)
```

Macro-level interface

The CICS macro-level interface is not supported.

SYSTEM(CICS) compile-time option

If you compile with `SYSTEM(CICS)` compile-time option, PL/I enforces the `OPTIONS(BYVALUE)` procedure option for `MAIN` procedures. `OPTIONS(BYVALUE)` is the default. If you specify `OPTIONS(BYADDR)`, the compiler diagnoses it as an error and applies `OPTIONS(BYVALUE)` instead.

FETCHing a PL/I MAIN procedure

CICS does not support PL/I FETCHing in a PL/I MAIN procedure.

STACK run-time option

Language Environment supports VisualAge PL/I applications that use the run-time option STACK(,ANY). Language Environment also supports STACK(,ANY) for OS PL/I applications that have been relinked with Language Environment as long as the applications meet the following conditions:

- Contains no edited stream I/O (for example, EDIT was not used in a PUT statement)
- Specifies AMODE(31)

Run-time output

Run-time output is now transmitted to the CICS transient data queue CESE. Language Environment ignores the MSGFILE option under CICS. Figure 1 shows format of the output data queue.

ASA	Terminal id	Transaction id	B	DateTime YYYYMMDDHHMMSS	B	Data
-----	----------------	-------------------	---	----------------------------	---	------

Figure 1. CESE output data queue

In addition, PL/I transient queues CPLI and CPLD are no longer used. As a result, you do not need to specify entries for the CPLI and CPLD in the CICS Destination Control Table (DCT).

Abend codes used by PL/I under CICS

The APLx abend codes that were issued under OS PL/I Version 2 are no longer issued. Instead, Language Environment-defined abend codes are issued. For more information about Language Environment abend codes, see *OS/390 Language Environment Debugging Guide and Run-Time Messages*.

Linking VisualAge PL/I applications

You are no longer required to take special actions when you link a VisualAge PL/I object module under CICS. The CEESTART CSECT is the entry point for programs compiled with OPTIONS(MAIN) or OPTIONS(FETCHABLE). If a subroutine that was **not** compiled with OPTIONS(FETCHABLE) is FETCHed or called, you must code the linkage editor ENTRY statement so that it nominates the actual entry point.

IMS considerations

Interfaces to IMS

Language Environment supports the PLITDLI, ASMTDLI, and EXEC DLI interfaces from a PL/I routine. It also supports CEETDLI interface from a VisualAge PL/I routine running under IMS/ESA® Version 4.

Under Language Environment, CEETDLI is the recommended interface. CEETDLI supports calls that use an Application Interface Block (AIB) or a Program Communication Block (PCB). CEETDLI is available under IMS/ESA Version 4. For more information about AIB and a complete description of the CEETDLI interface, see *IMS/ESA Version 4 Application Programming Guide*.

When you recompile your PL/I routine with VisualAge PL/I and you want to replace PLITDLI with CEETDLI, you must replace the parameters in the CALL statement with the actual blocks, instead of the pointer to the blocks as required in the CALL PLITDLI statement.

SYSTEM(IMS) compile-time option

The SYSTEM(IMS) option, available in OS PL/I Version 2, was supported for PL/I IMS applications only. The main procedure of an IMS application must use the POINTER data type for its parameters.

When you recompile your main procedure with VisualAge PL/I, the object module assumes that the parameters are passed as BYVALUE. Language Environment converts the parameters to the BYVALUE style for you, if necessary, so the parameters are always passed correctly. If you specify OPTIONS(BYADDR) when you recompile your main procedure with VisualAge PL/I, you receive an error message and the compiler applies BYVALUE instead.

PLICALLA Support in IMS

The OS PL/I PLICALLA entry point is supported under Language Environment; however, it is **not** a recommended interface for IMS. Instead, use the SYSTEM(IMS) compile-time option and the CEESTART entry point.

Language Environment provides the same support for OS PL/I PLICALLA applications; however, when you recompile your main load module with VisualAge PL/I and want to continue to use PLICALLA, you must follow additional rules. See “PLICALLA considerations” on page 13 for details.

PSB language options supported

Language Environment supports PL/I applications with the following PSBGEN LANG options in the supported releases of IMS:

IMS/ESA Version 4

Table 10 shows support for PSB LANG options in IMS/ESA Version 4.

Table 10 (Page 1 of 2). PSB LANG options for IMS/ESA Version 4 Release 1

SYSTEM option	Entry point	LANG=
IMS	CEESTART	PLI or other values except PASCAL
IMS	PLICALLA ¹	PLI
Omitted	CEESTART	Illegal

Table 10 (Page 2 of 2). PSB LANG options for IMS/ESA Version 4 Release 1

SYSTEM option	Entry point	LANG=
Omitted	PLICALLA ¹	PLI

Note: ¹Supported only for compatibility.

IMS/ESA Version 3 Release 1

Table 11 shows support for PSB LANG options in IMS/ESA Version 3 Release 1.

Table 11. PSB LANG options for IMS/ESA Version 3 Release 1

SYSTEM option	Entry point	LANG=
IMS	CEESTART	PLI
IMS	PLICALLA ¹	PLI
Omitted	CEESTART	Illegal
Omitted	PLICALLA ¹	PLI

Note: ¹Supported only for compatibility.

Assembler driving a PL/I transaction

If an assembler program is driving a transaction program written in PL/I, and assuming the PSBGEN LANG= option remains unchanged, you must use the SYSTEM(MVS) compile-time option when you recompile the PL/I main program with VisualAge PL/I. In this case, no changes to the assembler program are required.

Storage usage considerations

With IMS/ESA Version 3 Release 1, the parameters passed to the IMS interfaces are no longer restricted to the area below the 16M line. The parameters will most likely be placed above the 16M line if you use the following methods:

- Use the ANYWHERE suboption of the HEAP run-time option. It applies to variables with the CONTROLLED or BASED attribute because their storage is obtained from the heap.
- Use the ANYWHERE suboption of the STACK run-time option. If you relink your OS PL/I application with Language Environment and your application does not use any edited stream I/O, or you recompile your application with VisualAge PL/I, you can use STACK(,ANYWHERE) if your application is AMODE(31). In this case, the variables in automatic storage are placed above the 16M line.
- Place parameters in static storage and make sure the load module attribute used is AMODE(31).

Coordinated condition handling under IMS

Language Environment and IMS condition handling is coordinated, meaning that if a program interrupt or abend occurs when you application is running in an IMS environment, the Language Environment condition manager is informed whether the problem occurred in your application or in IMS. If the problem occurs in IMS, Language Environment, as well as any invoked HLL-specific condition handler, percolates the condition back to IMS.

With Language Environment run-time option TRAP(ON), Language Environment continues to support coordinated condition handling for the PLITDLI and ASMTDLI interface invoked from a PL/I routine.

With IMS/ESA Version 3 with PTF UN4928 or IMS/ESA Version 4, Language Environment also supports the coordinated condition handling for CEETDLI, CTDLI from a C routine, CBLTDLI from a COBOL program, AIBTDLI from a PL/I program, and ASMTDLI from a non-PL/I program.

Note that if a program interrupt or abend occurs in your application outside of IMS, or if a software condition of severity 2 or greater is raised outside of IMS, the Language Environment condition manager takes normal condition handling actions described in OS/390 Language Environment Programming Guide. In this case, in order to give IMS a chance to do database rollback, you must do one of the following:

- Resolve the error completely so that your application can continue.
- Issue a rollback call to IMS, and then terminate the application.
- Make sure that the application terminates abnormally by using the ABTERMENC(ABEND) run-time option to transform all abnormal terminations into system abends in order to cause IMS rollbacks.
- Make sure that the application terminates abnormally by providing a modified assembler user exit (CEEEXITA) that transforms all abnormal terminations into system abends in order to cause IMS rollbacks.

The assembler user exit you provide should check the return code and reason code or the CEEAUE_ABTERM bit, and requests an abend by setting the CEEAUE_ABND flag to ON, if appropriate. See OS/390 Language Environment Programming Guide for details.

Performance enhancement with Library Retention(LRR)

If you use LRR, you can get an improvement in performance. See “Improving CPU utilization” on page 26 for details.

DB2 considerations

There are no special considerations for using DB2 other than the considerations described in “IMS considerations” on page 30.

Chapter 8. OS PL/I coexistence with Language Environment

This chapter discusses how you can run your OS PL/I applications under either OS PL/I or Language Environment. This coexistence gives you the flexibility to migrate your OS PL/I applications to VisualAge PL/I and/or Language Environment gradually. It's important that you understand how Language Environment supports OS PL/I object and load modules before you consider the coexistence. For rules and information on Language Environment's support of OS PL/I object and load modules, see chapter on object and load modules in the *PL/I for MVS & VM Compiler and Run-Time Migration Guide*.

Coexistence under OS/390 non-CICS

Under OS/390, Language Environment can coexist with the OS PL/I library in the same SMP/E zone. This enables you to have the Language Environment and OS PL/I library in your environment at the same time and allows you to use either run-time by specifying each library in a certain order in your JCL. Which run-time is used depends on the sequence in which they appear in your JCL and what type of application you are running.

The following library search order rules apply to non-CICS applications and are valid only for pure PL/I applications with the same type of load modules.

Table 12. OS PL/I and Language Environment coexistence rules for non-CICS environment

Type of load module	Search sequence	Run-time used
OS PL/I load module, no Shared Library	1. OS PL/I 2. Language Environment	OS PL/I
	1. Language Environment 2. OS PL/I	Language Environment
OS PL/I load module, with OS PL/I Shared Library	1. OS PL/I 2. Language Environment	OS PL/I
	1. Language Environment 2. OS PL/I	Not supported
OS PL/I load module, with replaced Shared Library	1. OS PL/I 2. Language Environment	Not supported
	1. Language Environment 2. OS PL/I	Language Environment
OS PL/I object module, linked with Language Environment	1. OS PL/I 2. Language Environment	Language Environment
	1. Language Environment 2. OS PL/I	Language Environment
VisualAge PL/I	1. OS PL/I 2. Language Environment	Language Environment
	1. Language Environment 2. OS PL/I	Language Environment

If a pure PL/I application contains a mixture of PL/I load modules, the OS PL/I library and Language Environment library must appear in such order that only one run-time environment is used by all load modules in the application. For example, if a PL/I application contains a mixture of any of the following load modules:

- OS PL/I load module, no Shared Library

- OS PL/I object module, linked with Language Environment
- VisualAge PL/I load module

Language Environment must always be placed before the OS PL/I library in the search order.

If your application contains ILC, the search order of Language Environment must be correct for all ILC programs in the application.

Coexistence under OS/390 CICS

CICS allows multiple language run-time environments to coexist in the same CICS region. As a result, you can bring up the CICS region with both OS PL/I and Language Environment; however, if Language Environment exists and PL/I in Language Environment is enabled, Language Environment is used for all PL/I transactions, including OS PL/I. It is only when Language Environment does not exist, or the PL/I component in Language Environment is not enabled, that the OS PL/I environment is used for OS PL/I transactions. In this case, you cannot run VisualAge PL/I transactions because it requires Language Environment with the PL/I component enabled. The following table summarizes the coexistence support for OS PL/I transactions:

Table 13. Summary of coexistence support under OS/390 CICS

PPT	OS PL/I	Language Environment	Environment Used
PLI	Yes	Yes, with PL/I enabled	Language Environment
PLI	Yes	Yes, without PL/I enabled	OS PL/I
PLI	No	Yes, with PL/I enabled	Language Environment
PLI	No	Yes, without PL/I enabled	Abend APCL
LE370	Yes	Yes, with PL/I enabled	Language Environment
LE370	Yes	Yes, without PL/I enabled	Abend APCL
LE370	No	Yes, with PL/I enabled	Language Environment
LE370	No	Yes, without PL/I enabled	Abend APCL

The following shows how you can disable the PL/I component from Language Environment:

- If the CICS region is not up, do the following:
 - Delete CEEEV011 module from CEECCSD located in Language Environment SCEESAMP.
 - Run the DFHCSDUP utility with CEECCSD.
- If the CICS region is up, do the following:
 - CEDS DELETE PROGRAM(CEEV011), bring down CICS, and start CICS cold.

You must concatenate OS PL/I in front of Language Environment SCEERUN in DFHRPL in order to use the OS PL/I environment.

Coexistence under DB/2

If you write a user-defined function in PL/I for MVS & VM or OS PL/I, DB/2 passes some string-locator descriptors to the PL/I procedure. These descriptors have a different format than the descriptors used by VisualAge PL/I. In order for PL/I to interpret them correctly, the PROCEDURE statement must specify OPTIONS(CMPAT) if you declare any of these string parameters with an asterisk (*) for its length, as shown in the following example:

```
udf: proc( a, b, c ) options(cmpat);
```

```
  dcl (a,b,c) char(*) varying;
```

This is also true for any assembler code passing string-locator descriptors to PL/I.

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

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 NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4

555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

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

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This book is intended to help the customer migrate from previous releases of PL/I to VisualAge PL/I and OS/390 V2R8 Language Environment. This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of VisualAge PL/I.

Trademarks

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

AIX	IMS
CICS	IMS/ESA
CICS/ESA	Language Environment
DB2	OS/2
DFSMS	OS/390
DFSORT	VisualAge
IBM	

Windows is a trademark of Microsoft Corporation in the United States and/or other countries.

Bibliography

Bibliography

VisualAge PL/I publications

Fact Sheet, GC26-9470
Programming Guide, SC26-9473
Language Reference, SC26-9476
Messages and Codes, SC26-9478
Diagnosis Guide, SC26-9475
Compiler and Run-Time Migration Guide,
SC26-9474
Installation and Customization, GC26-9472
Building Graphical User Interfaces on OS/2,
GC26-9180-01

DB2 Version 2

Information and Concepts Guide, S20H-4664
Administration Guide, S20H-4580
Database System Monitor Guide and Reference,
S20H-4871
Command Reference, S20H-4645
API Reference, S20H-4984
SQL Reference, S20H-4665
Application Programming Guide, S20H-4643
Call Level Interface Guide and Reference,
S20H-4644

Messages Reference, S20H-4808
Problem Determination Guide, S20H-4779
DDCS User's Guide, S20H-4793
DRDA Connectivity Guide, SC26-4783

DATABASE 2

Application Programming and SQL Guide,
SC26-4377
SQL Reference, SC26-4380

VisualAge CICS Enterprise Application Development

Installation, GC34-5356
Customization, SC34-5357
Operation, SC34-5358
Reference Summary, SX33-6109
Intercommunication, SC34-5359
Problem Determination, GC34-5360
Performance, SC34-5363
Application Programming, SC34-5361

Index

A

- abend codes
 - CICS considerations 30
- AREAs and INITIAL attribute 24
- array expressions restriction 7
- ASMTDLI IMS interface 31
- assembler driving PL/I transaction, IMS considerations 32
- assembler invocation of PL/I 23
- assembler language options, IMS considerations 31
- assembler support
 - PLIMAIN entry point 23
 - PLISTART entry point 23
- assembler user exits
 - specific considerations 4

B

- batch restrictions 9
- built-in function restriction 9

C

- CEEBXITA user exit 4
- CEESTART, using 23
- CICS considerations
 - abend codes used by PL/I 30
 - CSD file, updating 29
 - discussion of 29
 - linking VisualAge PL/I applications 30
 - macro-level interface 29
 - run-time output 30
 - STACK run-time option, using 30
 - SYSTEM compile-time option 29
- coexistence, OS PL/I with Language Environment 34—36
 - under DB/2 36
 - under OS/390 CICS 35
 - under OS/390 non-CICS 34
- compatibility considerations 6
 - PLICALLA entry point 13
 - PLICALLB entry point 14
- compile unit definition 20
- compile-time considerations 5—11
 - installing Language Environment 5
 - mixing object levels 5
 - storage reports 10
- compiler messages 10
 - discussion of changes 10
- condition handling
 - IMS considerations 32

- Condition Handling Differences 17
- considerations
 - before migrating
 - Condition Handling 17
 - DATE/TIME built-in functions 16
 - ILC differences 22
 - OS PL/I coexistence with Language Environment 34
 - PLIDUMP 20
 - preinitialized program 16
 - run-time message 19
 - run-time options 21
 - storage report 22
 - user return code 17
 - compile-time 5
 - installation
 - High-Level Language user exit 4
 - OS/390 requirements 3
 - product configuration 3
 - product configuration, SCEELKED 3
 - product configuration, SCEERUN 3
 - link-edit
 - ENTRY CEESTART requirement 12
 - PLICALLA and PLICALLB 12
 - using FETCH 12
 - Run-Time 13
 - subsystem
 - CICS 29
 - DB2 33
 - IMS 30
- COUNT run-time option 21
- CPU utilization, improving 26
- CSD file, updating 29

D

- data sets
 - new, OS/390 3
- DATE/TIME built-in functions 16
- DB2 considerations 33
- DBCS restriction 8
- DEFINED variable restriction 8
- DEPTHCONDLMT run-time option 21
- Descriptors clause 10

E

- ENTRY statement restriction 7
- ERRCOUNT run-time option 21
- EXEC DLI interface 31

F

FETCH

considerations for 12

FIXED

BINARY, mapping and portability 24

FLOW run-time option 21

H

HEAP run-time option 21

High-Level Language user exits, using 4

I

IBMBEER user exit, installation considerations 4

IBMBXITA user exit 4

IBMFXITA user exit 4

ILC (interlanguage communication)

differences in 22

enabled languages 22

IMS considerations

assembler driving PL/I transaction 32

assembler language options support 31

condition handling 32

discussion of 30

interfaces 31

interfaces to 31

PLICALLA support 31

PSB language options 31

storage usage 32

SYSTEM compile-time option 31

INITIAL attribute 24

installation considerations

user exits 4

installing Language Environment, compile-time

considerations 5

interlanguage communication (ILC)

differences in 22

enabled languages 22

introduction

Language Environment library 2

PL/I run-time environment 1

user information 1

VisualAge PL/I for OS/390 library 2

ISASIZE run-time option 21

L

Language Environment library 2

LANGUAGE run-time option 21

link-edit considerations

ENTRY CEESTART requirement 12

FETCH 12

using FETCH

discussion of 12

using PLICALLA entry 12

link-edit considerations (*continued*)

using PLICALLB entry 12

linking applications under CICS 30

M

macro-level interface, CICS considerations 29

messages

compiler 10

PLIXOPT string errors

discussion of 10

mixing object levels, compile-time considerations 5

N

NOMAP 10

notices 37

O

OS PL/I

version 1

source code compatibility 6

P

performance

CPU utilization 26

retuning for 26

storage utilization 27

under CICS, improving 27

under IMS, improving 27

PL/I dependency on Language Environment 5

PL/I mixing object levels 5

PLICALLA Entry Point

IMS considerations 31

passing parameters 14

Support for 13

PLICALLB entry point

passing parameters 16

support for 14

PLIDUMP

output produced by 20

PLIDUMP differences 20

PLIMAIN entry point 23

PLISTART entry point 23

PLITDLI IMS interface 31

PLIXOPT string

messages issued

discussion of 10

portability

language elements 24

preinitialized program 16

product configuration

data sets

new 3

OS/390 3

- product configuration (*continued*)
 - discussion of 3
- programs, preinitialized 16
- PSB language options, IMS considerations 31
- pseudovisible restriction 9

R

- record I/O restriction 9
- REPORT run-time option 21
- restrictions under VisualAge PL/I 6
- retuning applications
 - CPU utilization 26
 - storage utilization, improving 27
 - under IMS, improving 27
- run-time
 - behavior differences
 - Amode(24) Support. 25
 - Descriptor Format 25
 - INITIAL attribute for AREAs is ignored 24
 - language elements 24
 - using variables declared as FIXED BIN 24
- run-time environment for PL/I 1
- run-time message differences 19
- run-time options differences 21
- run-time output, CICS considerations 30

S

- SCEELKED configuration 3
- SCEERUN configuration 3
- SPIE run-time option 21
- STACK run-time option 21, 30
- STAE run-time option 21
- storage
 - reports, compile-time considerations 10
 - usage
 - IMS considerations 32
 - retuning for 26
- storage report differences 22
- storage utilization, improving 27
- stream I/O restrictions 8
- structure expression restriction 8
- subsystem considerations
 - CICS 29
 - DB2 33
 - IMS 30
- subsystem performance, improving 27
- SYSTEM compile-time option
 - CICS considerations 29
 - IMS considerations 31

T

- TRAP run-time option 21

U

- user exits
 - assembler
 - specific considerations 4
 - CEEBINT 4
 - CEEBXITA 4
 - High-Level Language 4
 - IBMBEER 4
 - IBMBXITA 4
 - IBMXITA 4
 - installation considerations 4
 - user return code differences 17

V

- VisualAge PL/I library 2

We'd Like to Hear from You

IBM VisualAge PL/I for OS/390
Compiler and Run-Time Migration Guide
Version 2 Release 2.1
Publication No. SC26-9474-02

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:
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 VisualAge PL/I for OS/390
Compiler and Run-Time Migration Guide
Version 2 Release 2.1
Publication No. SC26-9474-02**

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	<input type="checkbox"/>				
Complete	<input type="checkbox"/>				
Easy to find	<input type="checkbox"/>				
Easy to understand	<input type="checkbox"/>				
Well organized	<input type="checkbox"/>				
Applicable to your tasks	<input type="checkbox"/>				
Grammatically correct and consistent	<input type="checkbox"/>				
Graphically well designed	<input type="checkbox"/>				
Overall satisfaction	<input type="checkbox"/>				

May we contact you to discuss your comments? Yes No

Would you like to receive our response by E-Mail?

Your E-mail address

Name

Address

Company or Organization

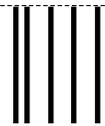
Phone No.



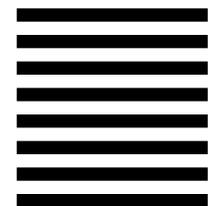
Fold and Tape

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

IBM Corporation
Department HHX/H3
PO Box 49023
San Jose, CA 95161-9945



Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5655-B22



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

Common VisualAge PL/I Library

SC26-9476 Language Reference

VisualAge PL/I for OS/390 Library

GC26-9471 Licensed Program Specifications
SC26-9473 Programming Guide
SC26-9474 Compiler and Run-Time Migration Guide
SC26-9475 Diagnosis Guide
SC26-9478 Compile-Time Messages and Codes

VisualAge PL/I Enterprise Version 2.1 Library

GC26-9178 Language Reference (OS/2 and Windows)
GC26-9177 Programming Guide (OS/2 and Windows)
GC26-9179 Messages and Codes (OS/2 and Windows)
GC26-9180 Building Graphical User Interfaces on OS/2

SC26-9474-02





IBM VisualAge PL/I for OS/390

Compiler and Run-Time Migration Guide

Version 2 Release 2.1