IBM COBOL for Linux on x86 1.2

Migration Guide

SC28-3454-00

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 49.

Third Edition (December 2025)

This edition applies to IBM® COBOL for Linux® on x86 1.2 or later compilers until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

You can view or download softcopy publications free of charge in the COBOL for Linux on x86 library.

Contents

Chapter 1. Introduction

This document presents topics to help you migrate COBOL source programs to COBOL for Linux on x86 1.2.

If you are upgrading from a previous version of COBOL for Linux on x86, see Chapter 2, “Migrating from an earlier version of COBOL for Linux on x86 to the current version,” on page 3 for information on changes you may need to be aware of when compiling and running your COBOL applications with COBOL for Linux on x86 1.2.

COBOL for Linux on x86 is a part of the IBM COBOL compiler family. Migrating COBOL programs from Enterprise COBOL for z/OS® or COBOL for AIX® to COBOL for Linux on x86 should be very straightforward. However, you might have to update some of your compiler options and environment variables and take into consideration differences in data representation between the platforms. From a COBOL language perspective, all the IBM COBOL compilers conform to the ISO 1989:1985, *Programming languages – COBOL* standard, therefore the majority of your COBOL source code should compile successfully with COBOL for Linux on x86. Enterprise COBOL on z/OS is the most advanced of the IBM COBOL compilers and supports several features in the COBOL 2002 and COBOL 2014 language standards. If you are migrating from Enterprise COBOL for z/OS 6 and your COBOL source contains usage of new language features, see “Language elements” on page 36 to confirm if those features are available in COBOL for Linux on x86. For more information, see Chapter 4, “Migrating from Enterprise COBOL for z/OS to COBOL for Linux on x86,” on page 31 and Chapter 5, “Migrating from COBOL for AIX to COBOL for Linux on x86,” on page 45.

COBOL programs created using non-IBM COBOL compilers will normally have some source code differences that will need to be modified to become compatible with COBOL for Linux on x86. These differences are often the result of implementer extensions or deviations from standard COBOL but might also be due to items in the COBOL standard that are specified as "implementor defined". The effort to migrate to COBOL for Linux on x86 will depend largely on how many differences to Standard COBOL were coded in the COBOL source files that you are migrating. COBOL for Linux on x86 includes a Source Conversion Utility (scu) to assist with this migration effort. Differences in data representation between the platforms and COBOL compilers may also need to be considered. For more information, see the Chapter 3, “Migrating from non-IBM COBOL compilers to COBOL for Linux on x86,” on page 7.

In this document, the term COBOL 2002 refers to the ISO/IEC 1989:2002, *Information technology - Programming languages - COBOL* language standard. This does not imply that COBOL for Linux on x86 has implemented the new features found in COBOL 2002 although some have been implemented. We are simply using COBOL 2002 to help you to identify extensions to COBOL that you might encounter as you do your migration, and to help you to determine the changes that you might need to make to resolve the extensions you encounter. It will be useful for you to refer to the COBOL for Linux on x86 *Language Reference* and *Programming Guide* when modifying COBOL code. You can find these documents at the COBOL for Linux on x86 documentation.

Chapter 2. Migrating from an earlier version of COBOL for Linux on x86 to the current version

This information describes some areas you might need to consider if migrating programs from COBOL for Linux on x86 1.1 to 1.2.

Option changes

This information describes changes to options in COBOL for Linux on x86 1.2.

ADDR

The default option is changed from `ADDR(32)` to `ADDR(64)`. The change means that if you want to generate a 32-bit program in COBOL for Linux on x86 1.2, you must explicitly specify `ADDR(32)`, otherwise you generate a 64-bit program.

Migration of 32-bit applications to 64-bit mode

Execution results can change in 64-bit mode. If you migrate 32-bit applications to 64-bit mode, you might need to do some recoding to accommodate the differences.

Although the main consideration in migrating to 64-bit mode is the change in size of data items, other areas of an application could potentially be impacted, as described below.

`LENGTH OF` special register:

In 32-bit mode, the implicit definition of the `LENGTH OF` special register is `PICTURE 9(9) USAGE IS BINARY`. In 64-bit mode, the implicit definition is `PICTURE 9(18) USAGE IS BINARY`.

LENGTH intrinsic function:

In 32-bit mode, `FUNCTION LENGTH` returns a 9-digit integer. In 64-bit mode, it returns an 18-digit integer.

Address data items and index data items:

The storage allocation for data items that contain addresses or indexes is 4 bytes in 32-bit mode, and 8 bytes in 64-bit mode. The data items that are affected are those that are defined with any of the following usages:

- `POINTER`
- `FUNCTION-POINTER`
- `PROCEDURE-POINTER`
- `INDEX`

The special register `ADDRESS OF` is implicitly defined as `USAGE POINTER`. Therefore storage is allocated for it as described above for `USAGE POINTER` data items.

Layout of group items:

The change in allocation size of pointer data items and index data items between 32-bit mode and 64-bit mode programs, described above, affects the layout of group items. In particular, data items that are located after a pointer or index data item could be located at different offsets within a group. Also, the number of *slack bytes* (bytes inserted by the compiler to ensure correct alignment) could differ, and thus the total size of a group could change.

Restriction: Your COBOL programs should not rely on the layout of group items.

`SYNCHRONIZED` data items:

If the SYNCHRONIZED clause is specified for pointer data items or index data items, the items are aligned on a fullword boundary in 32-bit mode and on a doubleword boundary in 64-bit mode.

Because the size of pointers is different between 32-bit and 64-bit applications, the offsets of data items that are subsequent to them in a group, as well as the overall size of the group, will change. This is true regardless of whether or not slack bytes are inserted by the compiler when SYNCHRONIZED is specified. However, specifying SYNCHRONIZED will further affect the offsets of pointers and subsequent data items in a group, as well as the group size due to the difference in the alignment for pointers between 32-bit and 64-bit applications.

Redefining pointer data items:

Because the size of a pointer is different between 32-bit and 64-bit applications, using it as the object of a REDEFINES clause might yield undesired results unless the size of the redefining data item (that is, the REDEFINES subject) is also changed accordingly. However, even if you change the redefining data item to match the size of the pointer, carefully consider why you are redefining a pointer in the first place. The value of a pointer is an address and generally speaking your program should only use pointers in the COBOL statements that explicitly support using pointers such as the SET statement. Using a REDEFINES clause to inspect or set the value of a pointer is not recommended.

Using the IGY-ADDR conditional compilation variable

In general, it is recommended that you code your programs so that they are not affected by the switch from 32-bit to 64-bit applications. For example, there is no dependency on the layout of data items in a group if you code your programs in this way.

However, if for some reason your programs are sensitive to the addressing mode, there is a predefined conditional compilation variable named IGY-ADDR, which can be used to have different code paths compiled based on the setting of the ADDR option. This allows you to maintain a single source file that will work for both 32-bit and 64-bit compilations.

Data files:

Data files created by 32-bit mode programs can be processed by 64-bit mode programs, and data files created by 64-bit mode programs can be processed by 32-bit mode programs. No addressing mode is attached to a data file. However, if a record definition contains an address data item or index data item, migration to 64-bit mode could cause the length of the record, and thus the length of the file, to change.

Middleware subsystems:

If a program depends on a middleware subsystem to do data-processing tasks, make sure that that subsystem works with 64-bit programs.

Restriction: CICS® TXSeries® supports only 32-bit COBOL applications.

DB2® Version 11.5 supports either 32-bit or 64-bit COBOL applications.

Other considerations:

Linux does not support mixing 32-bit mode and 64-bit mode programs within an application, and the linker does not support linking object files of different addressing modes. If a 64-bit program makes interlanguage calls, or calls routines in a user library, make sure that 64-bit versions of the called routines are available.

Linux distribution support

Linux distributions that have reached end of support such as RHEL 7.8 are not supported with COBOL for Linux on x86 1.2. Check the *Installation Guide* for a list of Linux distributions supported by COBOL for Linux on x86 1.2.

Optional programs support

Optional programs that have reached end of support, such as Db2® 11.1, are not supported with COBOL for Linux on x86 1.2. Check the *Installation Guide* for a list of optional programs supported by COBOL for Linux on x86 1.2.

Runtime compatibility

When you install COBOL for Linux on x86 1.2, the runtime library will be installed in the location `/opt/ibm/cobol/rte`, upgrading previous versions of the runtime library that might have been installed in that location. As the COBOL runtime library is upward compatible, any COBOL programs created using COBOL for Linux on x86 1.1 will continue to run with the runtime library provided in COBOL for Linux on x86 1.2.

Chapter 3. Migrating from non-IBM COBOL compilers to COBOL for Linux on x86

This information describes some areas you might need to consider if migrating programs from non-IBM COBOL compilers to COBOL for Linux on x86.

Format changes

To run your source programs on IBM COBOL for Linux on x86, make the required format changes to your source programs.

File suffix .cob for COBOL source

You can now compile a COBOL program by using a `.cob` suffix in addition to the `.cbl` suffix.

IBM COBOL for Linux on x86 supports the `.cob` suffix for COBOL source files.

See the following example:

```
$ cob2 -o tcCobol tcCobol.cob
IBM COBOL for Linux 1.2.0 compile started
End of compilation 1, program TCCOBOL, no statements flagged.
```

Free-format COBOL source files

IBM COBOL for Linux on x86 suppresses some messages that are related to enforcing fixed-format source code.

When you use IBM COBOL for Linux on x86 to compile source code that contains free-format COBOL source, you will receive the error messages in the following example:

```
$ cat tcFreeFormat.cbl
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. FreeFormat.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500
000600 PROCEDURE DIVISION.
000700 BEGIN-MY-PROGRAM.
000800 DISPLAY "COBOL for Linux on x86 1.2.0...".
000900 STOP RUN.

$ cob2 tcFreeFormat.cbl
IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      8  IGYPS0009-E   "DISPLAY" should not begin in area "A".  It was processed as if found
                       in area "B".
      9  IGYPS0009-E   "STOP" should not begin in area "A".  It was processed as if found in
                       area "B".
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       2                                    2
End of compilation 1,  program FREEFORMAT,  highest severity: Error.
Return code 8
```

You can rewrite your source in fixed format, or use `scu` to convert your source from free-format source to fixed-format source:

```
$ cat sol-tcFreeFormat.cbl
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. FreeFormat.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500
000600 PROCEDURE DIVISION.
000700 BEGIN-MY-PROGRAM.
```

```
000800     DISPLAY "COBOL for Linux on x86 1.2.0...".
000900     STOP RUN.
```

Alphanumeric literals longer than 160 characters in COBOL source files

The length of an alphanumeric literal, excluding the separators that delimit the literal, must be greater than zero and less than or equal to 160 alphanumeric character positions.

When you use IBM COBOL for Linux on x86 to compile source code that contains one or more alphanumeric literals longer than 160 characters, you will receive the error message in the following example:

```
$ cat tcAlphanumeric160.cbl
   000100 IDENTIFICATION DIVISION.
   000200 PROGRAM-ID. TEST-CASE.
   000300 ENVIRONMENT DIVISION.
   000400
   000500 DATA DIVISION.
   000600 WORKING-STORAGE SECTION.
   000700
   000800 01 VARIABLE PIC N(300) VALUE "BEGIN = = = = = = =
   000900-       " = = = = = = = = = = = = = = = = = = = = = =
   001000-       " = = = = = = = = = = = = = = = = = = = = = =
   001100-       " = = = = = = = = = = = = = = = = = = = = = =
   001200-       "= = = = = = = = = = = = = = = = END".
   001300
   001400 PROCEDURE DIVISION.
   001500     DISPLAY VARIABLE.
   001600     STOP RUN.

$ cob2 -o tcAlphanumeric160 tcAlphanumeric160.cbl IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      8  IGYDS0026-E   An alphanumeric literal that was longer than 160 characters was
                       specified.  The literal was truncated to 160 characters.
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       1                                    1
End of compilation 1,  program TEST-CASE,  highest severity: Error.
Return code 8
```

According to Standard COBOL 2002, the length of an alphanumeric literal, excluding the separators that delimit the literal, must be greater than zero and less than or equal to 160 alphanumeric character positions.

Change your COBOL source to follow Standard COBOL 2002. You can change your code to split long literals. As shown in the following example, `MOVE “longliteral” to ITEM-1` can be split:

```
MOVE “long” to ITEM-1(1:4).
MOVE “literal” to ITEM-1(5:).
```

Extra and misplaced periods in COBOL source

You must remove extra or misplaced periods in the fixed-format source.

When you use IBM COBOL for Linux on x86 to compile source code that contains extra and misplaced periods, you will receive the error messages in the following example:

```
$ cat tcPeriods.cbl
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. TEST-CASE.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050
000060 PROCEDURE DIVISION.
000070     PERFORM GREETING.
000080    .
000090
000100     STOP RUN.
000110
000120 GREETING.
000130 .
000140
000150     DISPLAY "HELLO FROM IBM.".
```

```
$ cob2 -o tcPeriods tcPeriods.cbl
IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      8  IGYPS0019-W   No COBOL statement was found between periods.
                       Same message on line:     13
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       2                          2
End of compilation 1,  program TEST-CASE,  highest severity: Warning.
Return code 4
```

According to Standard COBOL 2002, fixed-format source cannot include extra or missing periods in the source.

Change your COBOL source to follow Standard COBOL 2002, or use `scu` to remove extra and misplaced periods:

```
$ cat sol-tcPeriods.cbl
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. TEST-CASE.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050
000060 PROCEDURE DIVISION.
000070     PERFORM GREETING.
000080
000090
000100     STOP RUN.
000110
000120 GREETING.
000130
000140
000150     DISPLAY "HELLO FROM IBM.".
```

User-defined words

Within a source element, a user-defined word can be used as only one type of source element.

Example 1

In the following example, the procedure name is the same as the program identifier:

```
$ cat tcSameName.cbl
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. GREETING.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050
000060 PROCEDURE DIVISION.
000070
000080 MY-PARAGRAPH.
000090     PERFORM GREETING.
000100     PERFORM END-PARAGRAPH.
000110
000120 GREETING.
000130     DISPLAY "WELCOME TO IBM COBOL for Linux 1.2.0...".
000140
000150 END-PARAGRAPH.
000160     DISPLAY "HAVE A NICE DAY!".

$ cob2 -o tcSameName tcSameName.cbl
IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      9  IGYPS3007-S   "GREETING" was not defined as a procedure-name.
                       The statement was discarded.
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       1                                              1
End of compilation 1,  program GREETING,  highest severity: Severe.
Return code 12
```

Example 2

In the following example, `GREETING` is used as both a data name and a procedure name:

```
$ cat tcSameName2.cbl
000010 IDENTIFICATION DIVISION.
```

```
000020 PROGRAM-ID. TEST-CASE.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050
000060 WORKING-STORAGE SECTION.
000070
000080 01 GREETING PIC X(20) VALUE "FROM IBM.".
000090 PROCEDURE DIVISION.
000100 MY-PROGRAM.
000110     PERFORM GREETING.
000120     STOP RUN.
000130
000140 GREETING.
000150     DISPLAY "HELLO " GREETING.

$ cob2 -o tcSameName tcSameName2.cbl
IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
     11  IGYPS3007-S   "GREETING" was not defined as a procedure-name.
                       The statement was discarded.
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       1                                              1
End of compilation 1,  program TEST-CASE,  highest severity: Severe.
Return code 12
```

According to Standard COBOL 2002, a user-defined words within a source element can be used as only one type of user-defined word.

Change your COBOL source to follow Standard COBOL 2002.

For Example 1:

```
$ cat sol-tcSameName.cbl
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. GREETING.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050
000060 PROCEDURE DIVISION.
000070
000080 MY-PARAGRAPH.
000090     PERFORM HELLO.
000100     PERFORM END-PARAGRAPH.
000110
000120 HELLO.
000130     DISPLAY "WELCOME TO IBM COBOL for Linux 1.2.0...".
000140
000150 END-PARAGRAPH.
000160     DISPLAY "HAVE A NICE DAY!".
```

For Example 2:

```
$ cat sol-tcSameName2.cbl
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. TEST-CASE.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050
000060 WORKING-STORAGE SECTION.
000070
000080 01 GREETING PIC X(20) VALUE "FROM IBM.".
000090 PROCEDURE DIVISION.
000100 MY-PROGRAM.
000110     PERFORM MY-GREETING.
000120     STOP RUN.
000130
000140 MY-GREETING.
000150     DISPLAY "HELLO " GREETING.
```

REPORT SECTION and SCREEN SECTION

IBM COBOL for Linux on x86 does not support `REPORT SECTION` or `SCREEN SECTION`.

When you use IBM COBOL for Linux on x86 to compile source code that contains `REPORT SECTION` or `SCREEN SECTION`, you will receive the error messages in the following example:

```
$ cat tcReportScreenSection.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. TEST-CASE.
     000030 ENVIRONMENT DIVISION.
     000040 DATA DIVISION.
     000050
     000060 REPORT SECTION.
     000070 01 TYPE IS PAGE HEADING.
     000080 01 TYPE IS PAGE FOOTING.
     000090
     000100 SCREEN SECTION.

$ cob2 -o tcReportScreenSection tcReportScreenSection.cbl IBM COBOL for Linux 1.2.0 compile
started
0LineID  Message code  Message text
      6  IGYDS0148-S   "REPORT" is a reserved word related to language not supported by this
                       compiler.  The statement was discarded.
     10  IGYDS0009-E   "SCREEN" should not begin in area "A".  It was processed as if found
                       in area "B".
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       2                                   1         1
End of compilation 1,  program TEST-CASE,  highest severity: Severe.
Return code 12
```

For details about `REPORT SECTION` and `SCREEN SECTION`, see Standard COBOL 2002, section 13.7, Report section, and section 13.8, Screen section.

Embedded null characters in variables

When you use IBM COBOL for Linux on x86 to compile source code that contains a variable with embedded null characters, null characters will be removed.

See the output of `MY-DATA` in the following example. The null characters are removed.

```
$ cat tcBlankReplaceNull.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. "TEST-CASE".
     000030 ENVIRONMENT DIVISION.
     000040 CONFIGURATION SECTION.
     000050 DATA DIVISION.
     000060 WORKING-STORAGE SECTION.
     000080   77 MY-DATA PIC X(5) VALUE X"4F4E00004E".
     000120 PROCEDURE DIVISION.
     000125     DISPLAY MY-DATA "<-".
     000130     STOP RUN.
$ cob2 tcBlankReplaceNull.cbl IBM COBOL for Linux 1.2.0 compile started
End of compilation 1,  program TEST-CASE,  no statements flagged.
$ a.out
ONN<-
```

Format of floating point literals

IBM COBOL for Linux on x86 uses this format for floating point literals: [+|-] <mantissa>E [+|-] <exponent>.

When you use IBM COBOL for Linux on x86 to compile source programs containing floating point literals that do not use an E-notation; for example, if 1500.0 is used instead of 1.5E3, you will receive the error messages in the following example:

```
$ cat tcFloatingFormat.cbl
     000100 IDENTIFICATION DIVISION.
     000200 PROGRAM-ID. TEST-CASE.
     000300 ENVIRONMENT DIVISION.
     000400 DATA DIVISION.
     000500
     000600 WORKING-STORAGE SECTION.
     000700 01 FLOAT1 COMP-1 VALUE 1357.9.
     000800 01 FLOAT2 COMP-1 VALUE 2468.0.
     000900
     001000 PROCEDURE DIVISION.
     001100     DISPLAY "FLOAT 1 IS " FLOAT1.
     001200     DISPLAY "FLOAT 2 IS " FLOAT2.
     001300     STOP RUN.

$ cob2 -o tcFloatingFormat tcFloatingFormat.cbl IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      7  IGYGR1080-S   A "VALUE" clause literal was not compatible with the
                       data category of the subject data item.  The "VALUE"
                       clause was discarded.
                       Same message on line:   8
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       2                                             2
End of compilation 1,  program TEST-CASE,  highest severity: Severe.
Return code 12
```

You can also use `scu` to convert such format of floating point literals.

RECORD SEQUENTIAL file organization

IBM COBOL for Linux on x86 does not support `RECORD SEQUENTIAL` file organization.

If your source contains the `RECORD SEQUENTIAL` file organization, you will receive the error messages in the following example.

Change `RECORD SEQUENTIAL` to `LINE SEQUENTIAL`, and then compile again. You can also use `scu` to replace `RECORD SEQUENTIAL` with `LINE SEQUENTIAL`.

```
$cat tcRecordSequential.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. TEST-CASE.
     000030 ENVIRONMENT DIVISION.
     000031
     000032 INPUT-OUTPUT SECTION.
     000033 FILE-CONTROL.
     000034     SELECT CONTACTS
     000035        ASSIGN TO "MYFILE.DAT"
     000036        ORGANIZATION IS RECORD SEQUENTIAL.
     000037
     000040 DATA DIVISION.
     000050 FILE SECTION.
     000051 FD CONTACTS
     000052     RECORD CONTAINS 20 CHARACTERS.
     000053 01 CONTACT-RECORD.
     000054    05 FNAME  PIC X(10).
     000055    05 LNAME  PIC X(10).
     000073
     000090 PROCEDURE DIVISION.
     000100     PERFORM OPEN-FILE.
     000110     PERFORM INPUT-DATA.
     000120     PERFORM SAVE-DATA.
     000130     PERFORM CLOSE-FILE.
     000140     PERFORM END-PROGRAM.
     000150
     000160 INPUT-DATA.
     000170     DISPLAY "ENTER FIRST NAME.".
     000180     ACCEPT FNAME.
     000190     DISPLAY "ENTER LAST NAME.".
     000200     ACCEPT LNAME.
     000210
     000220 OPEN-FILE.
     000230     OPEN OUTPUT CONTACTS.
     000240
     000250 SAVE-DATA.
     000260     WRITE CONTACT-RECORD.
     000270
     000280 CLOSE-FILE.
     000290     CLOSE CONTACTS.
     000300
     000310 END-PROGRAM.
     000320     STOP RUN.
$ cob2 -o tcRecordSequential tcRecordSequential.cbl IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      9  IGYLN2929-S   Incorrect ORGANIZATION type detected at RECORD.  Clause ignored.
      9  IGYLN1357-S   Expected a data-name; SEQUENTIAL found. Statement or clause ignored.
      13  IGYGR1216-I   A "RECORDING MODE" of "F" was assumed for file "CONTACTS".
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       3           1                                  2
End of compilation 1,  program TEST-CASE,  highest severity: Severe.
Return code 12
```

Data representation

The representation of data can differ between non-IBM compilers and COBOL for Linux on x86.

Binary data

IBM COBOL compilers use the native representation of the platform when handling binary (BINARY, COMP, COMP-4 and COMP-5) data items. On Linux x86, binary data items will be stored and manipulated in little-endian format (least significant digit at the lowest address).

When migrating COBOL applications from non-IBM compilers to COBOL for Linux on x86, you might get unexpected results if your program accesses data that is stored in big-endian format as the compiler and runtime will treat the data as little-endian format by default.

You can use the `BINARY(BE)` option to inform the COBOL for Linux on x86 compiler to handle BINARY, COMP and COMP-4 data items in big-endian format consistent with non-IBM compilers that represent binary items as big-endian. However, this will have some performance overhead as the compiler needs to convert the data to and from its native format, little-endian. COMP-5 data items are not affected by the `BINARY(BE)` or `BINARY(LE)` option as COMP-5 indicates a native binary data item that uses the native representation of the platform. If you use a combination of COMP-5 and other BINARY/COMP/COMP-4 data types in your program, take care when using the `BINARY(BE)` option. If a particular data item needs to remain in little-endian (LE) representation when `BINARY(BE)` has been specified, use the `NATIVE` clause on the USAGE statement.

Note:

- Micro Focus Visual Studio and COBOL-IT store BINARY/COMP/COMP-4 data in big endian format, irrespective of the platform, and COMP-5 data in the native endianness of the platform, so when using COBOL for Linux on x86, you might need to use the `BINARY(BE)` option to work with data in a way that is compatible with those compilers.
- If you are using IBM MQ, API parameters are expected to be in big endian format, so you will need to use the BINARY(BE) and FLOAT(BE) option when working with MQ on Linux.
- IBM Db2, and Oracle Pro*COBOL add their own data areas in the generated COBOL program to communicate with their client libraries. These client libraries expect data in native little-endian format on Linux, even if they support big-endian host data. When working with Db2 or Pro*COBOL, you should use the default native binary format (BINARY(NATIVE), or BINARY(LE)).
- Since IBM MQ expects a different binary format than IBM Db2 and Oracle Pro*COBOL, it is not recommended to have MQ and SQL calls in the same compilation unit or batch compilation.

National data

IBM COBOL compilers use the native UTF-16 representation of the platform when handling national data. On Linux x86, national data items will be stored and manipulated in UTF-16 little-endian format.

When migrating COBOL applications from non-IBM compilers to COBOL for Linux on x86, you might get unexpected results if your program accesses data that is stored in big-endian format as the compiler and runtime will treat the data as little-endian format by default.

You can use the `UTF16(BE)` option to inform the COBOL for Linux on x86 compiler to handle national data items in big-endian format consistent with non-IBM compilers that represent national data items as big-endian. However, this will have some performance overhead as the compiler needs to convert the data to and from its native format, little-endian. If a particular data item needs to remain in little-endian (LE) representation when `UTF16(BE)` has been specified, use the `NATIVE` clause on the USAGE statement.

Related tasks
"Setting the locale" in *Programming Guide*
"Fixing differences caused by data representations" in *Programming Guide*

Related references
"CHAR" in *Programming Guide*
"SOSI" in *Programming Guide*

IDENTIFICATION DIVISION changes

`IDENTIFICATION DIVISION` must be the first division in each COBOL source program.

When you use IBM COBOL for Linux on x86 to compile source code that contains an `IDENTIFICATION DIVISION` not as the first division in a COBOL source program, you will receive the error messages in the following example:

```
$ cat tcIdentificationDivision.cbl
000010 ENVIRONMENT DIVISION.
000020 DATA DIVISION.
000030
000040 WORKING-STORAGE SECTION.
000050 01 GREETING PIC X(50) VALUE "HELLO FROM IBM...".
000060 01 GOODBYE PIC X(50) VALUE "HAVE A NICE DAY...".
000070
000080 IDENTIFICATION DIVISION.
000090 PROGRAM-ID. TEST-CASE.
000100
000110 PROCEDURE DIVISION.
000120     DISPLAY GREETING.
000130     DISPLAY GOODBYE.
000140     STOP RUN.

$ cob2 -o tcIdentificationDivision tcIdentificationDivision.cbl
IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      1  IGYLN0034-E   Program-name expected; Default program-name COBOLPGM00 assumed.
     12  IGYPS2121-S   "GREETING" was not defined as a data-name.  The statement was
                       discarded.
                       Same message on line:     12
     13  IGYPS2121-S   "GOODBYE" was not defined as a data-name.  The statement was
                       discarded.
                       Same message on line:     13
     14  IGYSC0136-E   End of source file was encountered or an "END PROGRAM" marker for a
                       containing program was found, but an "END PROGRAM" marker for program
                       "TEST-CASE" was not found.  "END PROGRAM TEST-CASE" was assumed.
     14  IGYSC0136-E   End of source file was encountered or an "END PROGRAM" marker for a
                       containing program was found, but an "END PROGRAM" marker for program
                       "COBOLPGM00" was not found.  "END PROGRAM COBOLPGM00" was assumed.
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       7                                    3         4
End of compilation 1,  program COBOLPGM00,  highest severity: Severe.
Return code 12
```

According to Standard COBOL 2002, a source unit is a set of COBOL statements as specified in this document and consists of an `IDENTIFICATION DIVISION` followed optionally by any of the `ENVIRONMENT DIVISION`, `DATA DIVISION`, or `PROCEDURE DIVISION`, or a combination of those divisions.

Change your COBOL source to follow Standard COBOL 2002:

```
$ cat sol-tcIdentificationDivision.cbl
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. TEST-CASE.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050
000060 WORKING-STORAGE SECTION.
000070 01 GREETING PIC X(50) VALUE "HELLO FROM IBM...".
000080 01 GOODBYE PIC X(50) VALUE "HAVE A NICE DAY...".
000090
000100 PROCEDURE DIVISION.
000110     DISPLAY GREETING.
```

```
000120     DISPLAY GOODBYE.
000130     STOP RUN.
```

DATA DIVISION changes

To run your source programs on IBM COBOL for Linux on x86, make the required DATA DIVISION changes to your source programs.

OCCURS clauses in level 01

According to Standard COBOL 2002, you must follow the syntax rules for the OCCURS clause.

When you use IBM COBOL for Linux on x86 to compile source code that contains an OCCURS clause in level 01, you will receive the error messages in the following example:

```
$ cat tcOccursClause.cbl
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. TEST-CASE.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050
000060 WORKING-STORAGE SECTION.
000061
000062
000063 01 EMPLOYEES OCCURS 100 TIMES.
000064    05 FNAME PIC X(10).
000065    05 LNAME PIC X(10).
000066    05 SALARY PIC 9(5).
000067
000068 77 COUNTER PIC 9.
000070
000071 PROCEDURE DIVISION.
000072     MOVE "DAVID" TO FNAME(1).
000073     MOVE "SMITH" TO LNAME(1).
000074     MOVE "60000" TO SALARY(1).
000075     MOVE "JENNY" TO FNAME(2).
000076     MOVE "HU" TO LNAME(2).
000077     MOVE "55000" TO SALARY(2).
000078
000080 DISPLAY-RESULT.
000081     PERFORM VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 2
000082      DISPLAY "EMPLOYEE: " FNAME IN EMPLOYEES(COUNTER)" "
000083               LNAME IN EMPLOYEES(COUNTER)" "
000084              "Salary: " SALARY IN EMPLOYEES(COUNTER)
000085     END-PERFORM.
000086 STOP-PROGRAM.
000090     STOP RUN.
$ cob2 -o tcOccursClause tcOccursClause.cbl
IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      9  IGYDS1063-E   An "OCCURS" clause was found in the definition of a level-1 item.  The
                       "OCCURS" clause was discarded.
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       1                                    1
End of compilation 1,  program TEST-CASE,  highest severity: Error.
Return code 8
```

According to Standard COBOL 2002, the OCCURS clause cannot be specified in a data description entry that has any of the following items:

- A level-number of 01, 66, 77, or 88
- A variable-occurrence data item subordinate to it

Change your COBOL source to follow Standard COBOL 2002. For example, if you change the code in the previous example by using the definition of EMPLOYEES that is moved from level 01 to level 03, it compiles without messages.

```
$ cat sol-tcOccursClause.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. TEST-CASE.
     000030 ENVIRONMENT DIVISION.
     000040 DATA DIVISION.
     000050
     000060 WORKING-STORAGE SECTION.
     000061
     000062 01.
     000063 03 EMPLOYEES OCCURS 100 TIMES.
     000065    05 FNAME PIC X(10).
     000066    05 LNAME PIC X(10).
     000067    05 SALARY PIC 9(5).
     000068
     000069 77 COUNTER PIC 9.000070
     000071 PROCEDURE DIVISION.
     000072     MOVE "DAVID" TO FNAME(1).
     000073     MOVE "SMITH" TO LNAME(1).
     000074     MOVE "60000" TO SALARY(1).
     000075     MOVE "JENNY" TO FNAME(2).
     000076     MOVE "HU"    TO LNAME(2).
     000077     MOVE "55000" TO SALARY(2).
     000078
     000080 DISPLAY-RESULT.
     000082      PERFORM VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 2
     000084       DISPLAY "EMPLOYEE: " FNAME IN EMPLOYEE(COUNTER) " "
     000085                LNAME IN EMPLOYEE(COUNTER) " "
     000085               "Salary: " SALARY IN EMPLOYEE(COUNTER)
     000086      END-PERFORM.
     000087 STOP-PROGRAM.
     000120      STOP RUN.
```

COMP-X

IBM COBOL for Linux on x86 does not support `COMP-X`.

When you use IBM COBOL for Linux on x86 to compile source code that contains `COMP-X`, you will receive the error message in the following example:

```
$ cat tcCompX.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. TEST-CASE.
     000030 ENVIRONMENT DIVISION.
     000040 DATA DIVISION.
     000050
     000060 WORKING-STORAGE SECTION.
     000070 01 STU-ID PIC S9(4) COMP-X VALUE ZERO.

$ cob2 -o tcCompX tcCompX.cbl
IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      7  IGYDS1089-S   "COMP-X" was invalid.  Scanning was resumed at the
                       next area "A" item, level-number, or the start of
                       the next clause
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       1                                              1
End of compilation 1,  program TEST-CASE,  highest severity: Severe.
Return code 12
```

Change `COMP-X` to `COMP-5`, and make corresponding changes to the `PICTURE` clause. You must keep the same allocated storage for the data item.

The following table shows the `COMP-5` storage allocation:

Table 1. COMP-5 storage allocation

Picture	Storage representation
S9(1) through S9(4)	Binary halfword (2 bytes)
S9(5) through S9(9)	Binary fullword (4 bytes)
S9(10) through S9(18)	Binary doubleword (8 bytes)
9(1) through 9(4)	Binary halfword (2 bytes)
9(5) through 9(9)	Binary fullword (4 bytes)
9(10) through 9(18)	Binary doubleword (8 bytes)

Runtime errors because of uninitialized variables being converted to a display type

You will receive runtime error messages if the sender in a statement is moved to the receiver but the sender does not contain valid data for the type of the receiver, possibly because the variable being moved was not initialized.

When you run a program that contains an uninitialized variable, you will receive the error messages in the following example:

```
$cat tcConvert2DisplayType.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. "TEST-CASE".
     000030 ENVIRONMENT DIVISION.
     000040 CONFIGURATION SECTION.
     000050 DATA DIVISION.
     000060 WORKING-STORAGE SECTION.
     000070 01 EMP-INFO.
     000080    05  EMP-ID PIC S9(5) sign is trailing separate.
     000090 01 EMP-ID2 PIC S9(4).
     000100 PROCEDURE DIVISION.
     000120     MOVE EMP-ID OF EMP-INFO TO EMP-ID2.
     000130     STOP RUN.
```

```
$ cob2 -o tcConvert2DisplayType tcConvert2DisplayType.cbl
IBM COBOL for Linux 1.2.0 compile started
End of compilation 1,  program TEST-CASE,  no statements flagged.
> tcConvert2DisplayType
  Traceback:
/opt/ibm/cobol/rte/usr/lib/libcob2_64r.so(+0xa6302)[0x7f46046d9302]
/opt/ibm/cobol/rte/usr/lib/libcob2_64r.so(writeERRmsg+0x72e)[0x7f46046b0d2e]
/opt/ibm/cobol/rte/usr/lib/libcob2_64r.so(+0x7e762)[0x7f46046b1762]
/opt/ibm/cobol/rte/usr/lib/libcob2_64r.so(_iwzcBCD_CONV_ZndTS_To_ZndTO+0x356)[0x7f460466f446]
tcConvert2DisplayType(TEST-CASE+0xce)[0x560b45e13eca]
    --- End of call chain ---
IWZ040S  An invalid separate sign was detected.
IWZ901S  Program exits due to severe or critical error.

Aborted (core dumped)
```

Initialize `EMP-ID` of `EMP-INFO` to 0:

```
$ cat sol-tcConvert2DisplayType.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. "TEST-CASE".
     000030 ENVIRONMENT DIVISION.
     000040 CONFIGURATION SECTION.
     000050 DATA DIVISION.
     000060 WORKING-STORAGE SECTION.
     000070 01 EMP-INFO.
     000080    05  EMP-ID PIC S9(5) SIGN IS TRAILING SEPARATE
                VALUE ZERO.
     000090 01 EMP-ID2 PIC S9(4).
     000100 PROCEDURE DIVISION.
     000120     MOVE EMP-ID OF EMP-INFO TO EMP-ID2.
     000130     STOP RUN.
```

In addition to initializing `EMP-ID` to 0, you can compile by using `-qwsclear`, which clears a program's `WORKING-STORAGE` to binary zeros when the program is initialized. The storage is cleared before any VALUE clauses are applied. However, this approach is not preferred because of performance considerations.

Redefined data items and OCCURS clauses

IBM COBOL for Linux on x86 does not support redefining a data item that has an OCCURS clause.

When you use IBM COBOL for Linux on x86 to compile source code where a data item with the OCCURS clause is redefined, you will receive the error messages in the following example:

```
$ cat tcRedefineOccurs.cbl
     01 EMPLOYEE-INFO.
```

```
         03 EMPLOYEE OCCURS 50 TIMES.
               05 NAME PIC X(30).
               05 AGE PIC 9(3).
         03 NEW-EMPLOYEE REDEFINES EMPLOYEE.
            04 G2 OCCURS 50.
               05 FNAME PIC X(10).
               05 LNAME PIC X(20).
               05 AGE PIC 9(3).

$ cob2 -o tcRedefineOccurs tcRedefineOccurs.cbl
IBM COBOL for Linux 1.2.0 compile started
0LineID Message code Message text
     10 IGYLN1222-S EMPLOYEE contains a OCCURS clause. Clause ignored.
Messages Total Informational Warning Error Severe Terminating
Printed: 1 1
End of compilation 1, program PGM1, highest severity: Severe.
Return code 12
```

According to Standard COBOL 2002, the data item being redefined cannot contain an OCCURS clause.

Change your COBOL source to follow Standard COBOL 2002. For example, if you change the code in the previous example by adding a group item and renumbering, it compiles without messages.

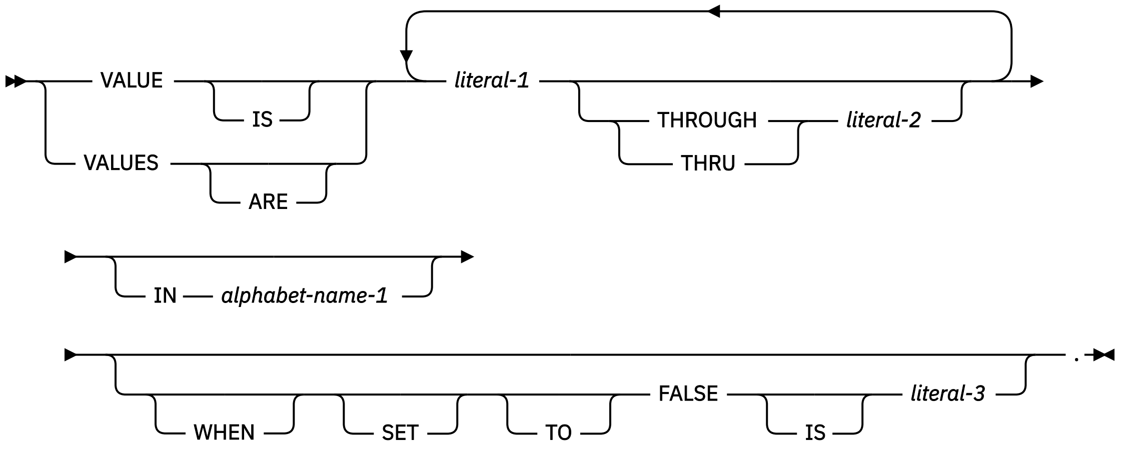
```
$ cat tcRedefineOccurs.cbl
     01 EMPLOYEE-INFO.
        02 G1.
           03 EMPLOYEE OCCURS 50 TIMES.
              05 NAME PIC X(30).
              05 AGE PIC 9(3).
        02 G2 REDEFINES G1.
           03 EMPLOYEE OCCURS 50 TIMES.
              05 FNAME PIC X(10).
              05 LNAME PIC X(20).
              05 AGE PIC 9(3).
```

WHEN condition used with VALUE clause

COBOL for Linux on x86 does not support the WHEN condition in format 2 of the VALUE clause.

Below is the syntax not supported:

Format 2: condition-name value



When using COBOL for Linux on x86 to compile source code that contains a data item with the above WHEN condition of the VALUE clause, you will receive the following error message:

```
$cat example.cbl
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    EXAMPLE.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 IsWeekday PIC A(10) VALUE "MON-FRI" WHEN SET TO FALSE IS "SAT/SUN".
       PROCEDURE DIVISION.
           STOP RUN.

IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      6  IGYDS1089-S   "WHEN" was invalid.  Scanning was resumed at the next area "A" item,
                       level-number, or the start of the next clause.
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       1                                            1
End of compilation 1,  program EXAMPLE,  highest severity: Severe.
Return code 12
```

The conditional statement in the VALUE clause needs to be removed to compile successfully. To achieve similar behavior as above, you can use the level 88 items. An example is as follows:

```
$cat modified.cbl
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    MODIFIED.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 dayType PIC A(10).
         88 weekday VALUE "MON-FRI".
         88 weekend VALUE "SAT/SUN".
       PROCEDURE DIVISION.
           SET weekday TO TRUE.
           IF weekday
             DISPLAY "Day type is a weekday!"
           END-IF.

           SET weekend TO TRUE.
           IF weekend
             DISPLAY "Day type is a weekend!"
           END-IF.
           STOP RUN.
```

Expressions used with VALUE clause

COBOL for Linux on x86 does not support the use of expressions with the VALUE clause. The VALUE clause only supports literals. If an expression is used with the VALUE clause, the statement will be discarded and the following error message will be emitted.

```
$cat example.cbl
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    EXAMPLE.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01   exampleItem   pic x(11) value "hello" & " world".
       PROCEDURE DIVISION.
           DISPLAY exampleItem.
           STOP RUN.

IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
      7  IGYDS1089-S   "&" was invalid.  Scanning was resumed at the next area "A" item,
                       level-number, or the start of the next clause.
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       1                                            1
End of compilation 1,  program EXAMPLE,  highest severity: Severe.
Return code 12
```

To resolve this issue, you need to remove the expression from the VALUE clause as the VALUE clause only supports literals.

PROCEDURE DIVISION changes

To run your source programs on IBM COBOL for Linux on x86, make the required `PROCEDURE DIVISION` changes to your source programs.

Moving national data items to alphanumeric data items

You cannot move a national data item to an alphanumeric data item.

When you use IBM COBOL for Linux on x86 to compile source code that contains a `MOVE` statement from a national data item to an alphanumeric data item, you will receive the error message in the following example:

```
$ cat tcNational2Alphanumeric.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. ND2AD.
     000030 ENVIRONMENT DIVISION.
     000040 CONFIGURATION SECTION.
     000050
     000060 DATA DIVISION.
     000070 WORKING-STORAGE SECTION.
     000080
     000090 01 DataNational      PIC N(100).
     000100
     000110 01 DataAlphanumeric    PIC X(100).
     000120
     000130 PROCEDURE DIVISION.
     000140       MOVE DataNational TO DataAlphanumeric.
     000150 STOP-MY-PROGRAM.
     000160       STOP RUN.


$ cob2 -o tcNational2Alphanumeric tcNational2Alphanumeric.cbl IBM COBOL for Linux 1.2.0
compile started 0LineID  Message code  Message text      14  IGYPA3005-S   "DATANATIONAL
(NATIONAL ITEM)" and "DATAALPHANUMERIC (ALPHANUMERIC)"                          did not
follow the "MOVE" statement compatibility rules.  The
statement was discarded.                                               -Messages
Total    Informational    Warning    Error    Severe    Terminating 0Printed:
1                                              1             End of compilation 1,  program
ND2AD,  highest severity: Severe. Return code 12
```

According to Standard COBOL 2002, you cannot move a national data item to an alphanumeric data item.

Change your COBOL source to follow Standard COBOL 2002.

Undefined symbol errors for C functions being called

Use the PGMNAME compiler option to control how the compiler handles program-names.

When you use IBM COBOL for Linux on x86 to compile source code that contains a call to a C function in mixed or lowercase characters, you will receive the error messages in the following example:

```
$ cat cTest.c
     void CFunction(){
     printf("HELLO FROM C...\n");
     }

$ cat tcCallCFunction.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. TEST-CASE.
     000030 ENVIRONMENT DIVISION.
     000040 CONFIGURATION SECTION.
     000050 DATA DIVISION.
     000060 WORKING-STORAGE SECTION.
     000070
     000080 PROCEDURE DIVISION.
     000090     CALL "CFunction"
     000100     DISPLAY "HELLO FROM COBOL...".
     000110     STOP RUN.

$ gcc -m64 -fPIC -c cTest.c  $ cob2 -o tcCallCFunction tcCallCFunction.cbl cTest.o IBM
COBOL for Linux 1.2.0 compile started End of compilation 1,  program TEST-CASE,  no
statements flagged. tcCallCFunction.o: In function `TEST-CASE': /home/user1/tcCallCFunction.cbl:
```

```
(.text+0xb5): undefined reference to `CFUNCTION' tcCallCFunction.o:(.rodata+0x50): undefined
reference to `CFUNCTION' collect2: error: ld returned 1 exit status
```

IBM COBOL for Linux on x86 folds program-names to uppercase. If you have COBOL source that contains a call to a C function in mixed or lowercase characters, this function is folded to uppercase characters. The linker will not find the program, and an error message is displayed to indicate an unresolved symbol.

You can use the PGMNAME compiler option to control how the compiler handles program-names. The default is `PGMNAME(UPPER)`, but you can use `PGMNAME(MIXED)` to process the program-name as is, without truncation, translation, or folding to uppercase. When you specify `PGMNAME(MIXED)`, use the literal format of the program-name; that is, make the program-name a literal string such as "`programname`", or you will see the following message:

```
IGYDS1046-E   A user-defined word was found as a "PROGRAM-ID" name under
the "PGMNAME(LONGMIXED)" compiler option.
```

For an example about a COBOL program that calls C functions, see *Example: COBOL program calling C functions* in the COBOL for Linux on x86 Programming Guide.

```
$ cat sol-tcCallCFunction.cbl
000010 CBL PGMNAME(LONGMIXED)
000020 IDENTIFICATION DIVISION.
000030 PROGRAM-ID. "TEST-CASE".
000040 ENVIRONMENT DIVISION.
000050 CONFIGURATION SECTION.
000060 DATA DIVISION.
000070 WORKING-STORAGE SECTION.
000080
000090 PROCEDURE DIVISION.
000100     CALL "CFunction"
000110     DISPLAY "HELLO FROM COBOL...".
000120     STOP RUN.
$ cob2 -o sol-tcCallCFunction sol-tcCallCFunction.cbl cTest.o
IBM COBOL for Linux 1.2.0 compile started
End of compilation 1,  program TEST-CASE,  no statements flagged.
$ sol-tcCallCFunction
HELLO FROM C...
HELLO FROM COBOL...
```

COBOL programs containing an ACCEPT statement

In IBM COBOL for Linux on x86, the ACCEPT statement assigns an input line to the data item.

- If the input line is shorter than the data item, the data item is padded with spaces of the appropriate representation.
- If the input line is longer than the data item:
  - When a program reads from a screen, the remaining characters are discarded.
  - When a program reads from a file, the remaining characters are retained as the next input line for the file.

If the input data is longer than the receiving area, then IBM COBOL for Linux on x86 pads the area with spaces of the appropriate representation for the receiving area.

According to Standard COBOL 2002, the implementer must define the size of a data transfer for each hardware device.

BINARY formatted data in an ACCEPT statement

According to Standard COBOL 2002, the ACCEPT statement causes the transfer of data from the hardware device. This data replaces the content of the data item referenced by *identifier-1*. Any conversion of data that is required between the hardware device and the data item referenced by *identifier-1* is defined by the implementer. The IBM COBOL for Linux on x86 implementation allows only displayable data (display, national, numeric, alphanumeric) which precludes binary formatted data.

When you use IBM COBOL for Linux on x86 to compile source code that contains BINARY formatted data in an ACCEPT statement, you will receive the error message in the following example:

```
$ cat tcAcceptBinary.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. TEST-CASE.
     000030 ENVIRONMENT DIVISION.
     000040 DATA DIVISION.
     000050
     000060 WORKING-STORAGE SECTION.
     000070 01 EMPLOYEE-ID PIC S9(4) COMP-5 VALUE ZERO.
     000080
     000090 PROCEDURE DIVISION.
     000100 MY-PROGRAM.
     000110     DISPLAY "PLEASE INSERT EMPLOYEE'S ID NUMBER..".
     000120     ACCEPT EMPLOYEE-ID.
     000130
     000140     STOP RUN.

$ cob2 -o tcAcceptBinary tcAcceptBinary.cbl IBM COBOL for Linux 1.2.0 compile
started 0LineID  Message code  Message text      12  IGYLN0642-E   Statement
accepted, but ACCEPT identifier may not achieve expected
conversion.                                                          -Messages    Total
Informational    Warning    Error    Severe    Terminating 0Printed:
1                                    1                       End of compilation 1,  program
TEST-CASE,  highest severity: Error. Return code 8
```

Change the data type from COMP-5 to DISPLAY, and make further changes to the input source as well.

```
$ cat sol-tcAcceptBinary.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. TEST-CASE.
     000030 ENVIRONMENT DIVISION.
     000040 DATA DIVISION.
     000050
     000060 WORKING-STORAGE SECTION.
     000070 01 EMP-ID PIC 9(4) VALUE ZEROES.
     000080 01 EMPLOYEE-ID PIC S9(4) COMP-5 VALUE ZERO.
     000090
     000100 PROCEDURE DIVISION.
     000110 MY-PROGRAM.
     000120     DISPLAY "PLEASE INSERT EMPLOYEE'S ID NUMBER..".
     000130     ACCEPT EMP-ID.
     000140     MOVE EMP-ID TO EMPLOYEE-ID.
     000150     STOP RUN.
```

NULL parameters in CALL statements

IBM COBOL for Linux on x86 does not support NULL parameters in a CALL statement.

When you use IBM COBOL for Linux on x86 to compile source code that contains NULL parameters in a CALL statement, you will receive the error message in the following example:

```
$ cat tcCallNull.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. TEST-CASE.
     000030 ENVIRONMENT DIVISION.
     000040 DATA DIVISION.
     000050
     000060 WORKING-STORAGE SECTION.
     000070
     000080 PROCEDURE DIVISION.
     000090     CALL "TEST-CASE2" USING NULL.
     000100     STOP RUN.

$ cob2 -o tcCallNull tcCallNull.cbl IBM COBOL for Linux 1.2.0 compile
started 0LineID  Message code  Message text       9  IGYPS2000-S   Expected a
data-name, but found "NULL".  The "CALL" statement was
discarded.                                                           -Messages    Total
Informational    Warning    Error    Severe    Terminating 0Printed:
1                                              1             End of compilation 1,  program
TEST-CASE,  highest severity: Severe. Return code 12
```

According to Standard COBOL 2002, syntax rules do not identify NULL as a valid parameter for the CALL statement.

```
$ cat sol-tcCallNull.cbl
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. TEST-CASE.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050
000060 WORKING-STORAGE SECTION.
000070 01 NULLPTR USAGE POINTER VALUE NULL.
000080
000110 PROCEDURE DIVISION.
000140     CALL "TEST-CASE2" USING NULLPTR.
000150     STOP RUN.
```

Unterminated nested statements

COBOL for Linux on x86 does not support an unterminated conditional statement that is contained within another statement with the exception of nested conditional statements within the IF statement. Nested statements must be imperative and follow the rules stated for imperative statements.

When you use COBOL for Linux on x86, if the procedure division contains an unterminated conditional statement that is nested, COBOL for Linux on x86 will emit the following error message.

```
$cat example.cbl
       IDENTIFICATION DIVISION.
       PROGRAM-ID.
           EXAMPLE
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 item PIC 9(10) VALUE 10.
       01 STU-ID PIC 9(10) VALUE 5.
       PROCEDURE DIVISION.
           EVALUATE item
             WHEN 10
               IF STU-ID = 5
                 DISPLAY "5"
           END-EVALUATE
           STOP RUN.

IBM COBOL for Linux 1.2.0 compile started
0LineID  Message code  Message text
     12  IGYPS2112-E   The "IF" statement did not have a matching scope terminator.  A scope
                       terminator was assumed on line 14.  The execution results may not be
                       correct.
-Messages    Total    Informational    Warning    Error    Severe    Terminating
0Printed:       1                                    1
End of compilation 1,  program EXAMPLE,  highest severity: Error.
Return code 8
```

The corresponding scope terminator needs to be added to the IF statement to become an imperative statement, and therefore would allow the IF statement to be nested in another scope. So, to resolve this issue, you can add the scope terminator to the source. Below is an example:

```
$cat example.cbl
       IDENTIFICATION DIVISION.
       PROGRAM-ID.
           EXAMPLE
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 item PIC 9(10) VALUE 10.
       01 STU-ID PIC 9(10) VALUE 5.
       PROCEDURE DIVISION.
           EVALUATE item
             WHEN 10
               IF STU-ID = 5
                 DISPLAY "5"
               END-IF
           END-EVALUATE
           STOP RUN.
```

Output for positive numbers

IBM COBOL for Linux on x86 output does not contain a plus sign (+) for positive numbers.

For example, when you use IBM COBOL for Linux on x86 to compile source code that contains a positive number, the output does not display the plus sign (+):

```
$ cat tcPlusSign.cbl
     000010 IDENTIFICATION DIVISION.
     000020 PROGRAM-ID. TEST-CASE.
     000030 ENVIRONMENT DIVISION.
     000040
     000050 INPUT-OUTPUT SECTION.
     000060
     000070 DATA DIVISION.
     000080 WORKING-STORAGE SECTION.
     000090 01 POSITIVE-NUM PIC S9(4) COMP-5 VALUE ZERO.
     000100 01 NEGATIVE-NUM PIC S9(4) COMP-5 VALUE ZERO.
     000110
     000120 PROCEDURE DIVISION.
     000130     COMPUTE POSITIVE-NUM = 10 - 3.
     000140     COMPUTE NEGATIVE-NUM = 3 - 10.
     000150     DISPLAY "10 - 3 = " POSITIVE-NUM.
     000160     DISPLAY "3 - 10 = " NEGATIVE-NUM.
     000170     STOP RUN.

$ cob2 -o tcPlusSign tcPlusSign.cbl IBM COBOL for Linux 1.2.0 compile started End of
compilation 1,  program TEST-CASE,  no statements flagged.

$ tcPlusSign2
10 - 3 = +0007
3 - 10 = -0007
```

Note: This rule does not apply to numeric-edited items. If you use an editing sign control symbol in a variable declared with `USAGE DISPLAY` or `NATIONAL`, such as the plus sign (+) in the following example:

```
000090 01 POSITIVE-NUM PIC +9(4).
000100 01 NEGATIVE-NUM PIC +9(4).
```

The output is as follows:

```
10 - 3 = +0007
3 - 10 = -0007
```

Source conversion utility (scu)

The source conversion utility, `scu`, is a stand-alone Linux program that assists in the conversion of COBOL source programs from non-IBM or free-format source formats to a format that can be compiled by COBOL for Linux on x86.

`Scu` performs the following transformations:

- Converts white space characters to true spaces, including tab expansion, with optional controls
- Normalizes line-end characters
- Repositions items to start in the proper areas, such as the indicator area, Area A, or Area B, as required
- Ensures special alignment for CBL (PROCESS) statements
- Optionally blanks sequence numbers in columns 1 through 6, and removes serial numbers in columns 73 through 80
- Converts anomalous fixed-source-format input by default, and optionally converts free-format input

It also makes the following syntax and semantic fixes:

- Adds missing spaces around quoted literals
- Adjusts the OCCURS clause to level 02 if it is at level 01
- Converts non-floating point literals to floating constant notations
- Fixes extra and misplaced periods

- Converts SET statements to MOVE when required by data types
- Converts "<>" to "NOT ="
- Replaces VALUES with VALUE
- Moves level 01 to Area A
- Replaces RECORD SEQUENTIAL with LINE SEQUENTIAL

Restrictions:

- Scu converts SET statements to MOVE only for the following data types:
  - `COMP`, `COMP-4`, or `BINARY`
  - `COMP-3` or `PACKED-DECIMAL`
  - `COMP-5`
  - `DISPLAY`
  - `NATIONAL` (non-literal)
- Scu might incorrectly convert the SET statement to MOVE if the statement spans two or more lines.
- Scu replaces VALUES with VALUE only when VALUES is followed by a literal on the same line.
- Scu converts non-floating point literals to floating constant notations only when the literals and VALUE (or VALUES) are on the same line.
- Scu ignores the REPLACE statement and the REPLACING phrase of the COPY statement.

Tips:

- For more information about Area A and Area B, see "Reference format" in *Language Reference*.
- Usually `scu` does initial transformation changes, and then fixes syntax and semantic errors (the `-N` option and copybooks are exceptions). However, if `scu` detects severe problems during transformation, such as non-COBOL standard special line, `scu` will generate error messages and skip the syntax and semantic phase. The `-N` option and copybook exceptions:
  - When `-N` is specified, `scu` does only the initial transformation changes.
  - Syntax and semantic errors in copybooks are fixed when these errors are fixed for the main source file with the `-G` option specified. For more information about processing copybooks with `scu`, see "Source conversion utility (scu) options" in *Language Reference*.

The output file contains compatible COBOL code that includes all fixes that `scu` can provide. If you do not specify the `-o` option for an output file name, the default output file is *`filename`*`.scu.cbl`, where *`filename`* is the original file name without the suffix. For example, the output file name for the source `abc.def.cob` is `abc.def.scu.cbl`.

`scu` issues return codes that indicate a success or failure of program conversion. See the return codes and corresponding explanations in the following table:

Table 2. Scu return codes

Return code	**Explanation**
0	To indicate that `scu` runs successfully.
1 - 4	Reserved for future use to indicate success.
5	To indicate a general failure.
6	To indicate a failure to open a copy file.

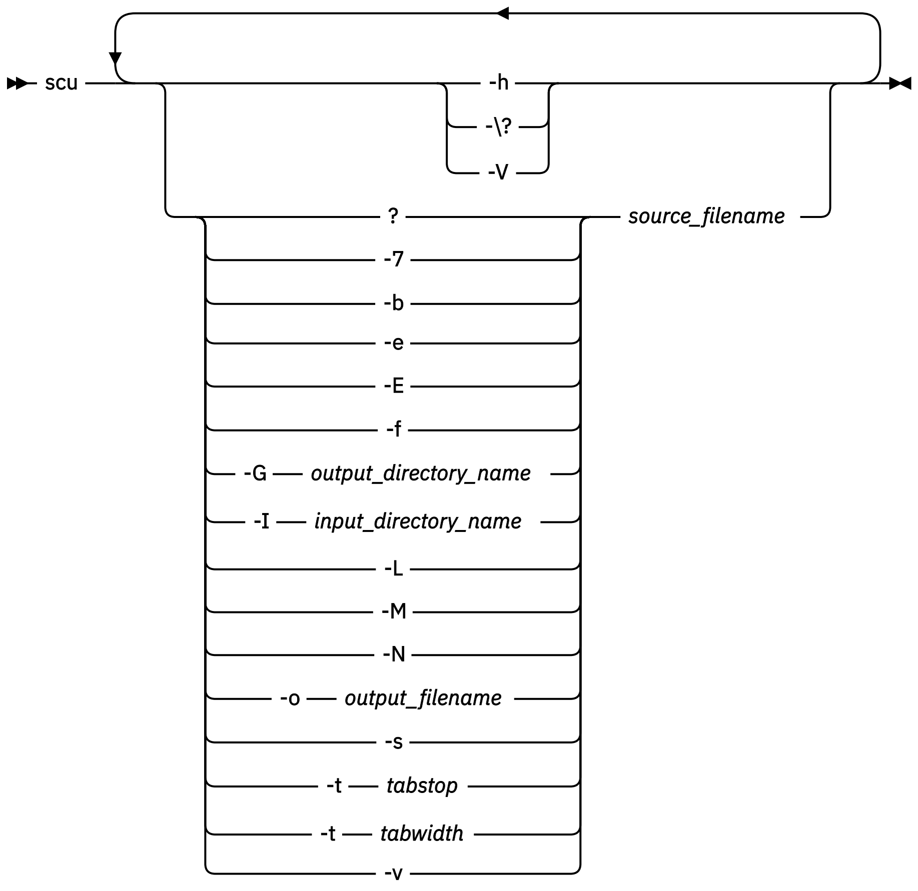
Attention: Scu converts source programs to a format that can be compiled with COBOL for Linux, but `scu` might not identify or fix all incompatibilities that exist in the code. IBM does not guarantee that the changes made by `scu` are error-free, or that the changes compile and run as you expect.

You must verify the results that are produced by `scu` and make sure that the changes meet your expectations. You might also need to further modify the code that `scu` attempted to correct.

Source conversion utility (scu) options

Multiple options are available in the source conversion utility (`scu`) for source program conversion.

The `scu` command has the following syntax:



Option descriptions:

-7

If a line starts with a special character, the `-7` option detects and moves the character to column 7 (the indicator area of fixed or extended format). The special character here can be an asterisk '*', slash '/', dollar sign '$', character 'D' followed by space and 'd' followed by space. The characters that follow the moved special character are handled depending on their positions:

- If the following characters are in column 2 - 7, the entire line is moved to the right until the special character is in column 7.
- If the following characters are in Area A or B, they remain where they are, unless there are characters in column 73 or beyond. When there are characters in column 73 or beyond, the characters might be moved left to the start of Area B.

Notes:

- For a dollar sign '$', `scu` issues an error that states manual intervention is required for this line.
- The special character '-' as the first character of a line is moved to column 7 only when you specify both the `-f` and `-7` options or when the `-f` option is specified and '-' is in column 1.
- For special characters that are not in column 7 or are not moved to column 7, `scu` handles these characters as regular (non-special) ones.

-b

Removes trailing blanks.

-e

Indicates whether an input file is in extended source format of a 252-character line. This option allows `scu` to distinguish the extended format from the default fixed format and covert the source code correctly.

-E

Tells `scu` that the output (the converted source) is not limited to the default 72 columns (the fixed format). `scu` can extend the lines to the maximum length of 252 columns if necessary.

-f

Identifies the input source as being in free format. By default the fixed (compatible) source format allows executable COBOL source code in Area A (column 8 - 11), and Area B (column 12 - 72), with indicators in column 7. Optionally, if you specify `-e`, Area B is extended to column 252 for the input file. The `-f` option causes `scu` to move COBOL source code from column 1 - 6 to the indicator column, Area A, or Area B depending on the content of the source code to be moved.

The `-f` option handles free-format source lines that start with special characters in one of the following ways:

- If the special character is in column 1 (the indicator area of free format), the character is moved to column 7 (the indicator area of fixed format).
- If the special character is in column 2-11, it is moved to column 12 (the start column of Area B).

Notes:

- Special characters in a free-format source file can be '*', '/', '$', '-','D' followed by space and 'd' followed by space.
- By default the `-f` option is not specified, and a special character at the start of a free-format line is moved to column 12 (start column of Area B) only when it is in column 8-11 (Area A). When the special character is in other columns, it remains where it is. A character that is not a special character in column 7 is blanked out.
- When the `-f` option is used in combination with the `-7` option, any special character in column 1-6 is moved to column 7. This usage is similar to the `-7` option alone, but includes '-' as a special character.

-G*<copybook output directory name>*

Fixes COPY files and places them in the specified directory. For a COPY file that is qualified with a directory name, the directory name is kept as the subdirectory of the specified COPY file directory. If the `-G` option is not specified, only the main source file is fixed.

Note: Do not insert spaces between `-G` and *<copybook output directory name>*.

-h

Provides `scu` basic help with information about the available functions of `scu`. You can also specify `-\?` to display the same help information as `-h` does. For more detailed help, see the `scu` man page by running the command `man scu`.

-I*<copybook input directory name>*

Adds the specified path to the directories to be searched for copybooks if a library-name or SYSLIB is not specified.

Notes:

- This option is the uppercase letter I, not the lowercase letter l.
- Only a single path is allowed for each `-I` option. To add multiple paths, use multiple `-I` options.
- Do not insert spaces between `-I` and *<copybook input directory name>*.
- COPY files that are retrieved from an `-I` directory, fixed by `scu` and stored in the`-G` directory can then be picked up by specifying `-I` and the same `-G` directory. In this way, `scu` uses a fixed version of the copy file on subsequent runs against the same or different main source files.

-L

Indents level numbers other than 01 and 77 to Area B when the level numbers are in Area A.

-M

Issues a `scu` fix code (for example, SCU0001) at the end of each fixed line and provides a short description and summary at the bottom of the output file. When standard compatible COBOL is input and the option `-M` is specified, a `scu` fix code is added to the unused noncompilable columns (starting at column 82) to indicate that the line is changed. A summary is also provided to display fix information that is associated with each fix code. The fix codes and summary are provided for syntax and semantic changes, not for the initial transformation changes. For example, when you specify `-f` to convert a file of free format to fixed 80 column or extended format, the line changes are made but you do not receive a `scu` fix code.

Use the fix code numbering to identify the level of needed attention for the message:

Table 3. Scu message severity levels

Fix code range	Severity	Description
SCU0001 - SCU1999	Informational	Scu expects that no further changes are needed for the fix.
SCU2000 - SCU3999	Warning	Scu expects that further changes might be needed. For instance, the fix of changing OCCURS from level 01 to level 02 might require further changes that are related to the fix.
SCU8000 - SCU8999	Error	Scu expects that further changes are required to complete the fix.
SCX0001 - SCX8999	Unfixed error	The problem is identified but `scu` is unable to fix the problem. The SCX*nnnn* error code corresponds to the matching SCU*nnnn* fix code.
SCX9000 - SCX9999	Ignored error	The problem is identified, but `scu` does not attempt to fix it.

Note: Scu might not identify or fix all incompatibilities that exist in the code.

The following list gives examples of `scu` fix codes and the corresponding fixed errors:

```
SCU0001 fix for IGYDS0001-W: Add missing space(s).
SCU0004 fix for IGYPS0019-W: Extra and misplaced periods in COBOL source. Scu removes the
extra periods.
SCU1002 fix for IGYGR1080-S: Non-floating point literal is assigned to floating point data
item. Scu converts it to floating point constant notation.
SCU1005 fix for IGYPS2024-S: SET used in place of MOVE. Scu converts SET stmt to MOVE stmt.
SCU1006 fix for IGYPS2094-S: "<>" converted to "NOT = ".
SCU1008 fix for IGYDS0017-E: "01" not in Area A. Scu moves 01 to Area A.
SCU3001 fix for IGYDS1063-E: OCCURS clause in level 01. Scu changes it to level 02 and adds
a dummy 01.
SCU8001 fix for IGYDS0093-S: RECORD SEQUENTIAL not supported. Scu replaces RECORD
SEQUENTIAL with LINE SEQUENTIAL.
SCX9001 specified for an 02 level data item following an 01 level OCCURS that has been
changed
to an 02 level OCCURS with fix code SCU3001
```

-N

Enables `scu` to do only the initial transformation changes without syntax and semantic changes, and forces the output to be written to the standard output.

-o *<output filename>*

Specifies the output file name for the source file. The output file can be qualified with an existing directory. For example, the command `scu -o/dirname1/abc.modified.cbl abc.cbl` saves the output file `abc.modified.cbl` in the `/dirname1` directory. By default the output file is saved in your current directory. If the `-o` option is not specified, the output file for the source file would be `abc.scu.cbl`.

-s
Removes the leading and trailing sequence numbers by blanking columns 1-6 and truncating the source line at column 73.

-t *<tabwidth>*
Passes to `scu` the tab width that is used in the source code to ensure that the converted data is in correct columns. Tab characters that are encountered before a tab position are replaced by spaces sufficient to move the ensuing character to the tab position. The default tab width is 8.

-t *<tabstop>,...*
Passes to `scu` the tab stops for conversion. Specify two or more tab stops separated by commas. Tab characters encountered past the last tab position are replaced by a single space character.

-v
Enables verbose output so that error and fix information is sent to STDERR during the source transformation, syntax and semantic checking, and fixing.

-V
Displays the version information of `scu`.

Copybooks and `scu`:

It is a good practice to have all copybooks go through transformation changes before `scu` attempts to fix syntax and semantic errors, because currently `scu` does not automatically perform transformation changes on copybooks. You can first run `scu` for your copybooks by specifying the `-N` option with any other transformation options, such as `-7`, `-b`, `-e`, `-E`, `-f`, `-L`, `-s`, and `-t`. Then, run `scu` for the main source files and specify `-I` with the copybook directory that contains the transformed copybooks for syntax and semantic error processing.

Chapter 4. Migrating from Enterprise COBOL for z/OS to COBOL for Linux on x86

This information describes some areas you might need to consider if migrating programs from Enterprise COBOL for z/OS to COBOL for Linux on x86.

Compiler options

COBOL for Linux on x86 does not support the following Enterprise COBOL compiler options.

ADATA, ADV, AFP, ARCH, AWO, BLOCK0, BUFSIZE, CODEPAGE, COPYLOC, COPYRIGHT, DATA, DBCS, DECK, DISPSIGN, DLL, DUMP, EXPORTALL, FASTSRT, HGPR, INITCHECK, INITIAL, INLINE, INTDATE, LANGUAGE, LP, MAXPCF, NAME, NUMCHECK, NUMPROC, OBJECT, OFFSET, OPTFILE, OUTDD, PARMCHECK, QUALIFY, RENT, RMODE, RULES, SERVICE, SQLCCSID, SQLIMS, STGOPT, SUPPRESS, THREAD, VLR, VSAMOPENFS, WORD, XMLPARSE, ZONECHECK, and ZONEDATA

While many of the options above have no equivalent on COBOL for Linux on x86 , the following options do have equivalents:

- Use ADDR instead of LP
- Use -dll, -dso or -shared flag options instead of DLL
- Use -c in combination with -o flag options instead of OBJECT

Related tasks
"Getting IBM Enterprise COBOL for z/OS applications to compile" in the *Programming Guide*

Related references
"cob2 options" in *Programming Guide*

Data representation

The representation of data can differ between Enterprise COBOL and COBOL for Linux on x86.

Binary data

IBM COBOL compilers use the native representation of the platform when handling binary (BINARY, COMP, COMP-4 and COMP-5) data items.

On Linux on x86, binary data items will be stored and manipulated in little-endian format, that is, the least significant digit is at the lowest address. On z/OS, binary data items will be stored and manipulated in big-endian format, that is, the most significant digit is at the lowest address.

When migrating COBOL applications from Enterprise COBOL for z/OS to COBOL for Linux on x86, you might get unexpected results if your program accesses data that is stored in big-endian format as the compiler and runtime will treat the data as little-endian format by default.

You can use the `BINARY(BE)` option to inform the COBOL for Linux on x86 compiler to handle BINARY, COMP and COMP-4 data items in big-endian format consistent with Enterprise COBOL for z/OS. However, this will have some performance overhead as the compiler needs to convert the data to and from its native format, little-endian. COMP-5 data items are not affected by the `BINARY(BE)` or `BINARY(LE)` option as COMP-5 indicates a native binary data item that uses the native representation of the platform. If you use a combination of COMP-5 and other BINARY/COMP/COMP-4 data types in your program, take care when using the `BINARY(BE)` option. If a particular data item needs to remain in little-endian (LE) representation when `BINARY(BE)` has been specified, use the `NATIVE` clause on the USAGE statement.

Notes:

- If you are using IBM MQ, API parameters are expected to be in big endian format, so you will need to use the BINARY(BE) and FLOAT(BE) option when working with MQ on Linux.
- IBM Db2 and Oracle Pro*COBOL add their own data areas in the generated COBOL program to communicate with their client libraries. These client libraries expect data in native little-endian format on Linux, even if they support big-endian host data. When working with Db2 or Pro*COBOL, you should use the default native binary format, that is BINARY(NATIVE) or BINARY(LE).
- Since IBM MQ expects a different binary format than IBM Db2 and Oracle Pro*COBOL, it is not recommended to have MQ and SQL calls in the same compilation unit or batch compilation.

Zoned decimal data

Sign representation for zoned decimal data is based on ASCII or EBCDIC depending on the setting of the `CHAR` compiler option (`NATIVE` or `EBCDIC`) and whether the `USAGE` clause is specified with the `NATIVE` phrase. COBOL for Linux on x86 processes the sign representation of zoned decimal data consistently with the processing that occurs on z/OS when the compiler option `NUMPROC(NOPFD)` is in effect.

Packed-decimal data

Sign representation for unsigned packed-decimal numbers is different between COBOL for Linux on x86 and Enterprise COBOL. COBOL for Linux on x86 always uses a sign nibble of `x'C'` for unsigned packed-decimal numbers. Enterprise COBOL uses a sign nibble of `x'F'` for unsigned packed-decimal numbers. If you are going to share data files that contain packed-decimal numbers between Linux and z/OS, it is recommended that you use signed packed-decimal numbers instead of unsigned packed-decimal numbers.

Internal floating-point data (COMP-1, COMP-2)

You can use the `FLOAT(BE)` compiler option to indicate that internal floating-point data items are in the IBM Z® data representation (hexadecimal) as opposed to the native (IEEE) format.

National data

IBM COBOL compilers use the native endianness of the platform when handling national data.

On Linux on x86, national data items will be stored and manipulated in UTF-16 little-endian format. On z/OS, national data items will be stored and manipulated in UTF-16 big-endian format.

When migrating COBOL applications from Enterprise COBOL for z/OS to COBOL for Linux on x86, you might get unexpected results if your program accesses data that is stored in big-endian format because the compiler and runtime will treat the data as little-endian format by default.

You can use the `UTF16(BE)` option to inform the COBOL for Linux on x86 compiler to handle national data items in big-endian format consistent with Enterprise COBOL for z/OS. However, this will have some performance overhead as the compiler needs to convert the data to and from its native format, little-endian. If a particular data item needs to remain in little-endian (LE) representation when `UTF16(BE)` has been specified, use the NATIVE clause on the USAGE statement.

EBCDIC and ASCII data

You can specify the EBCDIC collating sequence for alphanumeric data items using the following language elements:

- `ALPHABET` clause
- `PROGRAM COLLATING SEQUENCE` clause
- `COLLATING SEQUENCE` phrase of the `SORT` or `MERGE` statement

You can specify the `CHAR(EBCDIC)` compiler option to indicate that `DISPLAY` data items are in the IBM Z data representation (EBCDIC).

Code-page determination for data conversion

For alphabetic, alphanumeric, DBCS, and national data items, the source code page used for implicit conversion of native characters is determined from the locale in effect at run time.

For alphanumeric, DBCS, and national literals, the source code page used for implicit conversion of characters is determined from the locale in effect at compile time.

DBCS character strings

Under COBOL for Linux on x86, ASCII DBCS character strings are not delimited with the shift-in and shift-out characters except possibly with the dummy shift-in and shift-out characters as discussed below.

Use the SOSI compiler option to indicate that Linux workstation shift-out (X'1E') and shift-in (X'1F') control characters delimit DBCS character strings in the source program, including user-defined words, DBCS literals, alphanumeric literals, national literals, and comments. Host shift-out and shift-in control characters (X'0E' and X'0F', respectively) are usually converted to workstation shift-out and shift-in control characters when COBOL for Linux on x86 source code is downloaded, depending on the download method that you use.

Using control characters X'00' through X'1F' within an alphanumeric literal can yield unpredictable results.

Related tasks
"Setting the locale" in *Programming Guide*
"Fixing differences caused by data representations" in *Programming Guide*

Related references
"BINARY" in *Programming Guide*
"CHAR" in *Programming Guide*
"FLOAT" in *Programming Guide*
"SOSI" in *Programming Guide*
"UTF16" in *Programming Guide*

Compiler and runtime environment variables

COBOL for Linux on x86 recognizes several compiler and runtime environment variables that are not used in Enterprise COBOL, as listed below.

- COBCPYEXT
- COBLSTDIR
- COBOPT
- DB2DBDFT
- DBCS_CODEPAGE
- LANG
- LC_ALL
- LC_COLLATE
- LC_CTYPE
- LC_MESSAGES
- LC_TIME
- LD_LIBRARY_PATH
- *library-name*
- NLSPATH
- SYSLIB
- *text-name*

- TMPDIR
- TZ

Related information:

"Setting environment variables" in *Programming Guide*

File specification

There are some differences between the way COBOL for Linux on x86 handles files and the way Enterprise COBOL handles files.

The differences between COBOL for Linux on x86 and Enterprise COBOL in file handling are in the following areas:

- Single-volume files
- Source-file suffixes
- Generation data groups (GDGs)
- File concatenation

Single-volume files: COBOL for Linux on x86 treats all files as single-volume files. All other file specifications are treated as comments. This difference affects the following items: `REEL`, `UNIT`, `MULTIPLE FILE TAPE` clause, and `CLOSE. . .UNIT/REEL`.

Source-file suffixes: In COBOL for Linux on x86, when you compile using one of the `cob2` commands, COBOL source files that either have suffix .cbl or .cob are passed to the compiler. In Enterprise COBOL, when you compile in the z/OS UNIX file system, only files that have suffix .cbl are passed to the compiler.

Generation data groups (GDGs): GDG support is almost identical to GDG support in Enterprise COBOL. However, there are differences in COBOL for Linux on x86:

- Generation data sets (GDSs), or *generation files* as they are referred to in this information, are supported for all file organizations and access modes in all the supported file systems.
- GDG support is not integrated into the file systems. A stand-alone utility, `gdgmgr`, is provided for creating and deleting GDGs, managing and querying GDG entries, performing limit processing, and reconciling the GDG catalog against the existing files.
- The resolution of generation file names occurs when the files are opened rather than at job initialization.
- Limit processing is done when a new generation is added to a group, rather than at job termination.
- The generational range within any given epoch is from 1 to 9999, inclusive, instead of being limited to 1000. Therefore generations 0001 and 9999 can exist in the same epoch.
- A group can contain 1000 generations instead of 255.
- Versioning is not supported. The automatically generated version is always `v00`.

File concatenation: In COBOL for Linux on x86, you concatenate multiple files by separating the file identifiers with a colon (:). A COBOL file that is concatenated must have sequential or line-sequential organization, must be accessed sequentially, and can be opened only for input.

Related concepts
"File systems" in *Programming Guide*
"Generation data groups" in *Programming Guide*

Related tasks
"Concatenating files" in *Programming Guide*
"Compiling from the command line" in *Programming Guide*

Related references
"Limit processing of generation data groups" in *Programming Guide*

Interlanguage communication (ILC)

ILC is available with C/C++ programs.

These are the differences in ILC behavior on Linux on x86 compared to using ILC on z/OS with Language Environment®:

- There are differences in termination behavior when a COBOL `STOP RUN` or a C exit() is used.
- There is no coordinated condition handling with COBOL for Linux on x86. Avoid using a C longjmp() that crosses COBOL programs.
- With Enterprise COBOL, the first program that is invoked within the process and that is enabled for Language Environment is considered to be the "main" program. With COBOL for Linux on x86, the first COBOL program invoked within the process is considered to be the main program by COBOL. This difference affects language semantics that are sensitive to the definition of the run unit (the execution unit that starts with a main program). For example, a `STOP RUN` results in the return of control to the invoker of the main program, which in a mixed-language environment might be different as stated above.

Related concepts
"Calling between COBOL and C/C++ programs" in *Programming Guide*
"Preinitializing the COBOL runtime environment" in *Programming Guide*

Input and output

COBOL for Linux on x86 supports input and output for sequential, relative, and indexed files with the Db2, MONGO, SdU, SFS, and STL file systems, and also supports input and output for sequential files with the QSAM file system and RSD file system.

Line-sequential input and output is supported by the native byte stream file support of the operating system.

Sizes and values of the returned file status information can vary depending on which file system is used.

COBOL for Linux on x86 does not provide direct support for tape drives or diskette drives.

Related concepts
"File systems" in the *Programming Guide*
"Line-sequential file organization" in the *Programming Guide*

Related tasks
"Using file status keys" in the *Programming Guide*
"Using file system status codes" in the *Programming Guide*

Runtime options

COBOL for Linux on x86 does not recognize the following Enterprise COBOL runtime options, and treats them as not valid: `AIXBLD`, `ALL31`, `CBLPSHPOP`, `CBLQDA`, `COUNTRY`, `HEAP`, `MSGFILE`, `NATLANG`, `SIMVRD`, and `STACK`.

With Enterprise COBOL, you can use the `STORAGE` runtime option to initialize COBOL `WORKING-STORAGE`. With COBOL for Linux on x86, use the `WSCLEAR` compiler option.

Related references
"Runtime environment variables" in *Programming Guide*
"Compiler and runtime common environment variables" in *Programming Guide*
"WSCLEAR" in *Programming Guide*

Source code line size

In COBOL for Linux on x86, COBOL source lines can have varying lengths. A source line ends when a newline control character is encountered or when the maximum line length has been reached.

In Enterprise COBOL, each source line has the same length.

Related references
"SRCFORMAT" in the *Programming Guide*

Language elements

The following table lists language elements that are different between Enterprise COBOL and COBOL for Linux on x86 compilers, and where possible offers advice about how to handle such differences in COBOL for Linux on x86 programs.

Many COBOL clauses and phrases that are valid in Enterprise COBOL are syntax checked but have no effect on the execution of COBOL for Linux on x86 programs. These clauses and phrases should have minimal effect on existing applications that you download. COBOL for Linux on x86 recognizes most Enterprise COBOL language syntax even if that syntax has no functional effect.

Table 4. Language differences between Enterprise COBOL for z/OS and COBOL for Linux on x86

Language element	**COBOL for Linux on x86 implementation or restriction**
ACCEPT statement	If your Enterprise COBOL program expects ddnames as the targets of ACCEPT statements, define these targets by using equivalent environment variables with values set to appropriate file-names. In COBOL for Linux on x86, *environment-name* and the associated environment-variable value, if set, determine file identification.
APPLY WRITE-ONLY clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86
ASSIGN clause	COBOL for Linux on x86 uses a different syntax and mapping to the system file-name based on *assignment-name*. ASSIGN. . .USING *data-name* is not supported in Enterprise COBOL.
BLOCK CONTAINS clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86
CALL statement	A file-name as a CALL argument is not supported in COBOL for Linux on x86.
CLOSE statement	The following phrases are syntax checked, but have no effect on the execution of the program in COBOL for Linux on x86: FOR REMOVAL, WITH NO REWIND, and UNIT/REEL. Avoid use of these phrases in programs that are intended to be portable.
CODE-SET clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86
DATA RECORDS clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86
DISPLAY statement	If your Enterprise COBOL program expects ddnames as the targets of DISPLAY statements, define these targets by using equivalent environment variables with values set to appropriate file-names. In COBOL for Linux on x86, *environment-name* and the associated environment-variable value, if set, determine file identification.
DYNAMIC LENGTH clause	Dynamic-length elementary items are not currently supported in COBOL for Linux on x86.

Table 4. Language differences between Enterprise COBOL for z/OS and COBOL for Linux on x86 (continued)

Language element	**COBOL for Linux on x86 implementation or restriction**
File status *data-name-1*	Some values and meanings for file status 9*x* are different in Enterprise COBOL than in COBOL for Linux on x86.
File status *data-name-8*	The format and values are different depending on the platform and the file system.
INDEX data items	In Enterprise COBOL, INDEX data items are implicitly defined as 4 bytes. In COBOL for Linux on x86 programs compiled with ADDR(32), their size is 4 bytes; with ADDR(64), their size is 8 bytes.
INVOKE statement	COBOL for Linux on x86 does not support writing object-oriented programs, creating object instances of a COBOL or Java™ class or invoking a method defined in a COBOL or Java class.
JSON GENERATE and JSON PARSE statements	JSON is not currently supported in COBOL for Linux on x86.
LABEL RECORDS clause	The phrases LABEL RECORD IS *data-name*, USE. . .AFTER. . .LABEL PROCEDURE, and GO TO MORE-LABELS are syntax checked, but have no effect on the execution of the program in COBOL for Linux on x86. A warning is issued if you use any of these phrases. The user-label declaratives are not called at run time. You cannot port programs that depend on the user-label processing that z/OS QSAM supports.
MULTIPLE FILE TAPE	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86. On the Linux workstation, all files are treated as single-volume files.
OBJECT REFERENCE data items	OBJECT REFERENCE data items are not supported in COBOL for Linux on x86.
OPEN statement	The following phrases are syntax checked, but have no effect on the execution of the program in COBOL for Linux on x86: REVERSED and WITH NO REWIND.
PASSWORD clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86
POINTER, PROCEDURE-POINTER, and FUNCTION-POINTER data items	In Enterprise COBOL, POINTER and FUNCTION-POINTER data items are implicitly defined as 4 bytes as is the special register ADDRESS OF; PROCEDURE-POINTER data items are implicitly defined as 8 bytes. In COBOL for Linux on x86 programs compiled with ADDR(32), the size of each of these items is 4 bytes; with ADDR(64), their size is 8 bytes.
READ. . .PREVIOUS	In COBOL for Linux on x86 only, allows you to read the previous record for relative or indexed files with DYNAMIC access mode
RECORD CONTAINS clause	The RECORD CONTAINS *n* CHARACTERS clause is accepted with one exception: RECORD CONTAINS 0 CHARACTERS is syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86.
RECORDING MODE clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86 for relative, indexed, and line-sequential files. RECORDING MODE U is syntax checked, but has no effect on the execution of the program for sequential files.
RERUN clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86
RESERVE clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86

Table 4. Language differences between Enterprise COBOL for z/OS and COBOL for Linux on x86 (continued)

Language element	**COBOL for Linux on x86 implementation or restriction**
SAME AREA clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86
SAME SORT clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86
SHIFT-IN, SHIFT-OUT special registers	The COBOL for Linux on x86 compiler puts out an E-level message if it encounters these registers unless the CHAR(EBCDIC) compiler option is in effect.
SORT-CONTROL special register	The implicit definition and contents of this special register differ between host and workstation COBOL.
SORT-CORE-SIZE special register	The contents of this special register differ between host and workstation COBOL.
SORT-FILE-SIZE special register	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86. Values in this special register are not used.
SORT-MESSAGE special register	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86
SORT-MODE-SIZE special register	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86. Values in this special register are not used.
SORT MERGE AREA clause	Syntax checked, but has no effect on the execution of the program in COBOL for Linux on x86
START...	In COBOL for Linux on x86, the following relational operators are allowed: IS LESS THAN, IS <, IS NOT GREATER THAN, IS NOT >, IS LESS THAN OR EQUAL TO, IS <=.
STOP RUN	Not supported in COBOL for Linux on x86 multithreaded programs; can be replaced in a multithreaded program with a call to the C exit() function
UTF-8 phrase of the USAGE clause and the 'U' PICTURE symbol	The UTF-8 data class and UTF-8 data category are not currently supported in COBOL for Linux on x86.
WRITE statement	In COBOL for Linux on x86, if you specify WRITE. . .ADVANCING with environment names C01 through C12 or S01 through S05, one line is advanced.
XML PARSE statement	In Enterprise COBOL programs compiled using the host-only option XMLPARSE(XMLSS), additional syntax (the ENCODING phrase and RETURNING NATIONAL phrase) and special registers for namespace processing are available that are not available with COBOL for Linux on x86.
Names known to the platform environment	The following names are identified differently: program-name, text-name, library-name, assignment-name, file-name in the SORT-CONTROL special register, basis-name, DISPLAY or ACCEPT target identification, and system-dependent names.

Complementary products

COBOL Report Writer is not available on Linux on x86.

The COBOL Report Writer Precompiler is used on z/OS to compile applications that contain Report Writer statements, or to permanently convert Report Writer statements to valid Enterprise COBOL statements. More information about COBOL Report Writer is available at the Enterprise COBOL for z/OS Documentation.

If you are migrating COBOL programs from z/OS to Linux on x86 and those programs use the COBOL Report Writer Precompiler, first use the Precompiler to permanently convert Report Writer statements to Enterprise COBOL statements, and then migrate the post-processed COBOL programs to Linux on x86. Future changes to those programs that require updates in Report related sections might need to be re-processed on z/OS and then moved to Linux on x86.

Getting IBM Enterprise COBOL for z/OS applications to compile

If you move Enterprise COBOL programs from an IBM Z system to a Linux on x86 system and compile them using COBOL for Linux on x86, you need to choose the right compiler options and be aware of language features that differ from IBM Enterprise COBOL for z/OS. You can also use the `COPY` statement to help port programs.

Choosing the right compiler options: For additional information about Enterprise COBOL compiler options that affect portability, see the related reference about compiler options.

Allowing for language features of Enterprise COBOL: Several language features that are valid in Enterprise COBOL programs can create errors or unpredictable results when compiled with COBOL for Linux on x86. For details, see the related reference about language elements.

Using the `COPY` statement to help port programs: In many cases, you can avoid potential portability problems by using the `COPY` statement to isolate platform-specific code. For example, you can include platform-specific code in a compilation for a given platform and exclude it from compilation for a different platform. You can also use the `COPY REPLACING` phrase to globally change nonportable source code elements, such as file-names.

Related tasks
"Setting environment variables" in the *Programming Guide*

Related references
Compiler options
"Language elements" on page 36
"COPY statement" in the *Language Reference*

Getting IBM Enterprise COBOL for z/OS applications to run: overview

After you download an Enterprise COBOL program and successfully compile it using COBOL for Linux on x86, the next step is to run the program. In many cases, you can get the same results as on IBM z/OS without greatly modifying the source.

To assess whether to modify the source, you need to know how to fix the elements and behavior of the COBOL language that vary due to the underlying hardware or software architecture.

Related tasks
"Fixing differences caused by data representations" on page 39
"Fixing environment differences that affect portability" on page 42
"Fixing differences caused by language elements" on page 42

Fixing differences caused by data representations

To ensure the same behavior for your programs, you should understand the differences in certain ways of representing data, and take appropriate action.

Character data might be represented differently, depending on the USAGE clause that describes data items and the locale that is in effect at run time. COBOL stores signed packed-decimal in the same manner on both Linux on x86 and IBM z/OS. However, binary, external-decimal, floating-point, and unsigned packed-decimal data are by default represented differently.

Most programs behave the same on IBM z/OS and Linux on x86 regardless of the data representation.

Related tasks
"Handling differences in ASCII SBCS and EBCDIC SBCS characters" on page 40
"Handling differences in IEEE and hexadecimal data" on page 41
"Handling differences in ASCII multibyte and EBCDIC DBCS strings" on page 41

Related references
"Data representation" on page 31

Handling differences in ASCII SBCS and EBCDIC SBCS characters

To avoid problems with the different data representation between ASCII and EBCDIC characters, use the `CHAR(EBCDIC)` compiler option.

COBOL for Linux on x86 uses the ASCII character set, and Enterprise COBOL for z/OS uses the EBCDIC character set. Therefore, most characters have a different hexadecimal value, as shown in the following table.

Table 5. ***ASCII characters contrasted with EBCDIC***

Character	Hexadecimal value if ASCII	Hexadecimal value if EBCDIC
'0' through '9'	X'30' through X'39'	X'F0' through X'F9'
'a'	X'61'	X'81'
'A'	X'41'	X'C1'
blank	X'20'	X'40'

Also, code that depends on the EBCDIC hexadecimal values of character data probably fails when the character data has ASCII values, as shown in the following table.

Table 6. ***ASCII comparisons contrasted with EBCDIC***

Comparison	Evaluation if ASCII	Evaluation if EBCDIC
'a' < 'A'	False	True
'A' < '1'	False	True
x >= '0'	If true, does not indicate whether *x* is a digit	If true, *x* is probably a digit
x = X'40'	Does not test whether *x* is a blank	Tests whether *x* is a blank

Because of these differences, the results of sorting character strings are different between EBCDIC and ASCII. For many programs, these differences have no effect, but you should be aware of potential logic errors if your program depends on the exact sequence in which some character strings are sorted. If your program depends on the EBCDIC collating sequence and you are porting it to the workstation, you can obtain the EBCDIC collating sequence by using `PROGRAM COLLATING SEQUENCE IS EBCDIC` or the `COLLSEQ(EBCDIC)` compiler option.

Related references
"CHAR" in the *Programming Guide*
"COLLSEQ" in the *Programming Guide*

Handling differences in IEEE and hexadecimal data

To avoid most problems with the different representation between IEEE and hexadecimal floating-point data, use the `FLOAT(BE)` compiler option.

COBOL for Linux on x86 represents floating-point data using the IEEE format. Enterprise COBOL for z/OS uses the IBM Z hexadecimal format. The following table summarizes the differences between normalized floating-point IEEE and normalized hexadecimal for `USAGE COMP-1` data and `USAGE COMP-2` data.

Table 7. ***IEEE contrasted with hexadecimal***

Specification	IEEE for COMP-1 data	Hexadecimal for COMP-1 data	IEEE for COMP-2 data	Hexadecimal for COMP-2 data
Range	1.17E-38* to 3.37E+38*	5.4E-79* to 7.2E+75*	2.23E-308* to 1.67E+308*	5.4E-79* to 7.2E+75*
Exponent representation	8 bits	7 bits	11 bits	7 bits
Mantissa representation	23 bits	24 bits	53 bits	56 bits
Digits of accuracy	6 digits	6 digits	15 digits	16 digits

* Indicates that the value can be positive or negative.

For most programs, these differences should create no problems. However, use caution when porting if your program depends on hexadecimal representation of data.

Performance consideration: In general, IBM Z floating-point representation makes a program run more slowly because the software must simulate the semantics of IBM Z hardware instructions. This is a consideration especially if the `FLOAT(BE)` compiler option is in effect and a program has a large number of floating-point calculations.

"Examples: numeric data and internal representation" in the *Programming Guide*

Related references
"FLOAT" in the *Programming Guide*

Handling differences in ASCII multibyte and EBCDIC DBCS strings

To obtain Enterprise COBOL behavior for alphanumeric data items that contain DBCS characters, use the `CHAR(EBCDIC)` and `SOSI` compiler options. To avoid problems with the different data representation between ASCII DBCS and EBCDIC DBCS characters, use the `CHAR(EBCDIC)` compiler option.

In alphanumeric data items, Enterprise COBOL double-byte character strings (containing EBCDIC DBCS characters) are enclosed in shift codes, and COBOL for Linux on x86 multibyte character strings (containing ASCII DBCS, UTF-8, or EUC characters) are not enclosed in shift codes. The hexadecimal values used to represent the same characters are also different.

In DBCS data items, Enterprise COBOL double-byte character strings are not enclosed in shift codes, but the hexadecimal values used to represent characters are different from the hexadecimal values used to represent the same characters in COBOL for Linux on x86 multibyte strings.

For most programs, these differences should not make porting difficult. However, if your program depends on the hexadecimal value of a multibyte string, or expects that an alphanumeric character string contains a mixture of single-byte characters and multibyte characters, use caution in your coding practices.

Related references
"CHAR" in the *Programming Guide*
"SOSI" in the *Programming Guide*

Fixing environment differences that affect portability

Differences in file-names and control codes between Linux on x86 and IBM z/OS platforms can affect the portability of your programs.

File naming conventions on Linux on x86 are very different from those on IBM z/OS. This difference can affect portability if you use file-names in your COBOL source programs. The following file-name, for example, is valid on Linux on x86 but not on IBM z/OS (except in the z/OS UNIX file system):

```
/users/joesmith/programs/cobol/myfile.cbl
```

Case sensitivity: Unlike z/OS, Linux is case sensitive. Names used in source programs (such as uppercase file-names) should be named appropriately in Linux file directories.

Some characters that have no particular meaning on z/OS are interpreted as control characters by Linux. This difference can lead to incorrect processing of ASCII text files. Files should not contain any of the following characters:

- X'0A' (LF: line feed)
- X'0D' (CR: carriage return)
- X'1A' (EOF: end-of-file)

If you use device-dependent (platform-specific) control codes in your programs or files, these control codes can cause problems when you try to port the programs or files to platforms that do not support the control codes. As with all other platform-specific code, it is best to isolate such code as much as possible so that you can replace it easily when you move the application to another platform.

Fixing differences caused by language elements

In general, you can expect portable COBOL programs to behave the same way on Linux as they do on z/OS. However, be aware of the differences in file-status values used in I/O processing.

If your program responds to file-status data items, be concerned with two issues, depending on whether the program is written to respond to the first or the second file-status data item:

- If your program responds to the first file-status data item (*data-name-1*), be aware that values returned in the 9*n* range depend on the platform. If your program relies on the interpretation of a particular 9*n* value (for example, 97), do not expect the value to have the same meaning on Linux that it has on z/OS. Instead, revise your program so that it responds to any 9*n* value as a generic I/O failure.
- If your program responds to the second file-status data item (*data-name-8*), be aware that the values returned depend on both the platform and file system. For example, the STL file system returns values with a different record structure on Linux than the VSAM file system does on z/OS. If your program relies on the interpretation of the second file-status data item, the program is probably not portable.

Related tasks
"Using file status keys" in the *Programming Guide*
"Using file system status codes" in the *Programming Guide*

Related references
"FILE STATUS clause" in the *Programming Guide*
"File status key" in the *Programming Guide*

Writing code to run with IBM Enterprise COBOL for z/OS

You can use COBOL for Linux on x86 to develop new applications, and take advantage of the productivity gains and increased flexibility of using your Linux on x86 system. However, when you develop COBOL programs, you need to avoid using features that are not supported by IBM Enterprise COBOL for z/OS.

Language features: COBOL for Linux on x86 supports several language features that are not supported by Enterprise COBOL. As you write code on Linux on x86 that is intended to run on z/OS, avoid using these features:

- Code-page names as arguments to the DISPLAY-OF and NATIONAL-OF intrinsic functions
- READ statement using the PREVIOUS phrase
- START statement using <, <=, or NOT > in the KEY phrase
- >>CALLINTERFACE compiler directive

Compiler options: Several compiler options are not available on Enterprise COBOL. Do not use any of the following compiler options in your source code if you intend to port the code to z/OS:

- BINARY(NATIVE)
- CALLINT (treated as a comment)
- CHAR(NATIVE)
- FLOAT(NATIVE)

File names: Be aware of the difference in file-naming conventions between Linux and host file systems. Avoid hard-coding the names of files in your source programs. Instead, use mnemonic names that you define on each platform, and map them in turn to mainframe ddnames or environment variables. You can then compile your program to accommodate the changes in file-names without having to change the source code.

Specifically, consider how you refer to files in the following language elements:

- ACCEPT or DISPLAY target names
- ASSIGN clause
- COPY statement (*text-name* or *library-name*)

File suffixes: In COBOL for Linux on x86, when you compile using one of the cob2 commands, COBOL source files that have suffix .cbl or .cob are passed to the compiler. In mainframe COBOL, when you compile in the z/OS UNIX file system, however, only files that have suffix .cbl are passed to the compiler.

Nested programs: Multithreaded programs on the mainframe must be recursive. Therefore, avoid coding nested programs if you intend to port your programs to the mainframe and enable them to run in a multithreaded environment.

Chapter 5. Migrating from COBOL for AIX to COBOL for Linux on x86

This information describes some areas you might need to consider if migrating programs from COBOL for AIX to COBOL for Linux on x86.

Compiler options

COBOL for Linux on x86 does not support the following COBOL for AIX compiler options.

ADATA
The ADATA option is not currently supported. Use the default **NOADATA** for now.

ARCH
This option is used by COBOL for AIX to target different Power® machine architectures. It is not supported on Linux because COBOL for Linux on x86 makes use of the most efficient x86 instruction set currently available.

ENTRYINT
This option is treated as a comment on AIX, and not supported on Linux.

DUMP
Use of this option will generate a warning message but will not terminate compilation.

LIB
Use of this option will generate a warning message but will not terminate compilation.

MAXMEM
Use of this option will generate a warning message but will not terminate compilation.

SIZE
Use of this option will generate a warning message but will not terminate compilation.

COBOL for Linux on x86 does not support the following COBOL for AIX flag options:

-cmain

On Linux, this option is accepted and ignored.

With COBOL for AIX, the **-cmain** option has an effect only if you also specify **-host**. When **-host** is used, the compiler needs a pseudo main program to convert the arguments from EBCDIC to ASCII, and then call the COBOL program. If you have a C or PL/I object file that contains a main routine, the **-cmain** option informs cob2 not to link with the pseudo main, and makes the C or PL/I object file that contains a main routine the main entry point in the executable file.

With COBOL for Linux on x86, cob2 always links with the pseudo main, and forces it to be the entry point of the executable. You can still have your own C program called `main`, and it will be called from the pseudo main. If you do provide your own main(), it must understand that it is being passed a single z/OS style parameter list with the string null terminated.

```
struct plist {
    uint16_t len;
    uint8_t  str[1025];
};
```

-p and -pg
These options are profiling options for use with tprof on AIX. On Linux, use Valgrind instead. No additional instrumentation or changes to the COBOL program are needed to make use of Valgrind.

Related references
"cob2 options" in *Programming Guide*

Data representation

The representation of data can differ between COBOL for AIX and COBOL for Linux on x86.

Binary data

IBM COBOL compilers use the native representation of the platform when handling binary (BINARY, COMP, COMP-4 and COMP-5) data items.

On Linux x86, binary data items will be stored and manipulated in little-endian format (least significant digit at the lowest address). On AIX, binary data items will be stored and manipulated in big-endian format (most significant digit at the lowest address).

When migrating COBOL applications from COBOL for AIX to COBOL for Linux on x86, you might get unexpected results if your program accesses data that is stored in big-endian format as the compiler and runtime will treat the data as little-endian format by default..

You can use the `BINARY(BE)` option to inform the COBOL for Linux on x86 compiler to handle BINARY, COMP and COMP-4 data items in big-endian format consistent with COBOL for AIX, however this will have some performance overhead as the compiler needs to convert the data to and from its native format, little-endian. COMP-5 data items are not affected by the BINARY(BE) or BINARY(LE) option as COMP-5 indicates a native binary data item that uses the native representation of the platform. If you use a combination of COMP-5 and other BINARY/COMP/COMP-4 data types in your program, take care when using the BINARY(BE) option. If a particular data item needs to remain in little-endian (LE) representation when BINARY(BE) has been specified, use the NATIVE clause on the USAGE statement.

Note:

- If you are using IBM MQ, API parameters are expected to be in big endian format, so you will need to use the BINARY(BE) and FLOAT(BE) option when working with MQ on Linux.
- IBM Db2, and Oracle Pro*COBOL add their own data areas in the generated COBOL program to communicate with their client libraries. These client libraries expect data in native little-endian format on Linux, even if they support big-endian host data. When working with Db2 or Pro*COBOL, you should use the default native binary format (BINARY(NATIVE), or BINARY(LE)).
- Since IBM MQ expects a different binary format than IBM Db2 and Oracle Pro*COBOL, it is not recommended to have MQ and SQL calls in the same compilation unit or batch compilation.

National data

IBM COBOL compilers use the native UTF-16 representation of the platform when handling National data.

On Linux x86, National data items will be stored and manipulated in UTF-16 little-endian format. On AIX, National data items will be stored and manipulated in UTF-16 big-endian format.

When migrating COBOL applications from COBOL for AIX to COBOL for Linux on x86, you might get unexpected results if your program accesses data that is stored in big-endian format as the compiler and runtime will treat the data as little-endian format by default..

You can use the UTF16(BE) option to inform the COBOL for Linux on x86 compiler to handle National data items in big-endian format consistent with COBOL for AIX, however this will have some performance overhead as the compiler needs to convert the data to and from its native format, little-endian. If a particular data item needs to remain in little-endian (LE) representation when UTF16(BE) has been specified, use the NATIVE clause on the USAGE statement.

Related tasks
"Setting the locale" in *Programming Guide*
"Fixing differences caused by data representations" in *Programming Guide*

Related references
"CHAR" in *Programming Guide*
"SOSI" in *Programming Guide*

Compiler and runtime environment variables

COBOL for Linux on x86 does not recognize the following compiler and runtime environment variables that are available in COBOL for AIX.

- CLASSPATH
- COBJVMINITOPTIONS
- COBRTDUMP
- COBMSGS
- LIBPATH
- LOCPATH
- TMP

The ENCINA SFS related environment variables available with COBOL for AIX are renamed to CICS SFS environment variables in COBOL for Linux on x86.

COBOL for Linux on x86 recognizes several compiler and runtime environment variables that are not used in COBOL for AIX, as listed below.

- COBCORE
- COBOUTDIR
- DBCS_CODEPAGE
- LD_LIBRARY_PATH
- TMPDIR
- VFS_MONGODB_DATABASE
- VFS_MONGODB_URI

Related information:

"Setting environment variables" in *Programming Guide*

Language elements

The following table lists language elements that are different between COBOL for AIX and COBOL for Linux on x86 compilers, and where possible offers advice about how to handle such differences in COBOL for Linux on x86 programs.

Table 8. ***Language differences between COBOL for AIX and COBOL for Linux on x86***

Language element	**COBOL for Linux on x86 implementation or restriction**
INVOKE statement	COBOL for Linux on x86 does not support writing object-oriented programs, creating object instances of a COBOL or Java™ class, or invoking a method defined in a COBOL or Java class.
OBJECT REFERENCE data items	OBJECT REFERENCE data items are not supported in COBOL for Linux on x86.
SDU file system	COBOL for Linux on x86 does not support the SDU file system. If you specify SDU, it will be treated as if STL was specified. When you specify VSAM as the file system, it will default to STL. See "Identifying files" in *Programming Guide*.

File Specification

There are some differences between the way COBOL for Linux on x86 handles files and the way COBOL for AIX handles files.

Opening files for writing

Opening a file for writing is a non-blocking call on AIX, but is a blocking call on Linux. On AIX, if multiple processes are accessing the same file, process B will not wait for process A to be done with the file before process B opens the file for a write. It is up to you to ensure that only one process writes to the file at a time. On Linux, if one process has opened a file for a write, the second process will wait until the first process is finished to avoid any file corruption. You can use the VFS_LOCK=0 environment variable to change the file open for writing operation on Linux back to a non-blocking call. You should use this environment variable with caution as there is the potential for file corruption.

Notices

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Trademarks

Product Number: 5737-L11

SC28-3454-00