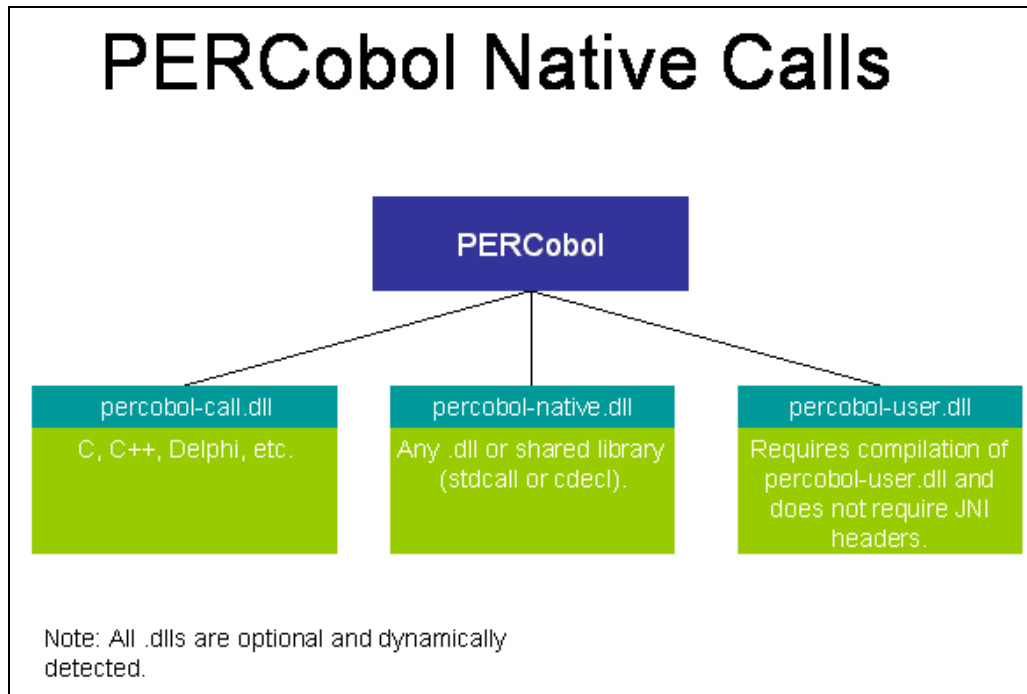


# CALLing To and From LegacyJ

---

PERCobol supports native CALL “to” and “from” numerous program types. LegacyJ supports native C, C++, COBOL, Visual Basic (Windows), Java and other programming languages. The following overview describes how this might be done in a Windows environment.



Generally CALL support can be done in all supported system environments whether that be proprietary mainframe environment or “open” system environment.

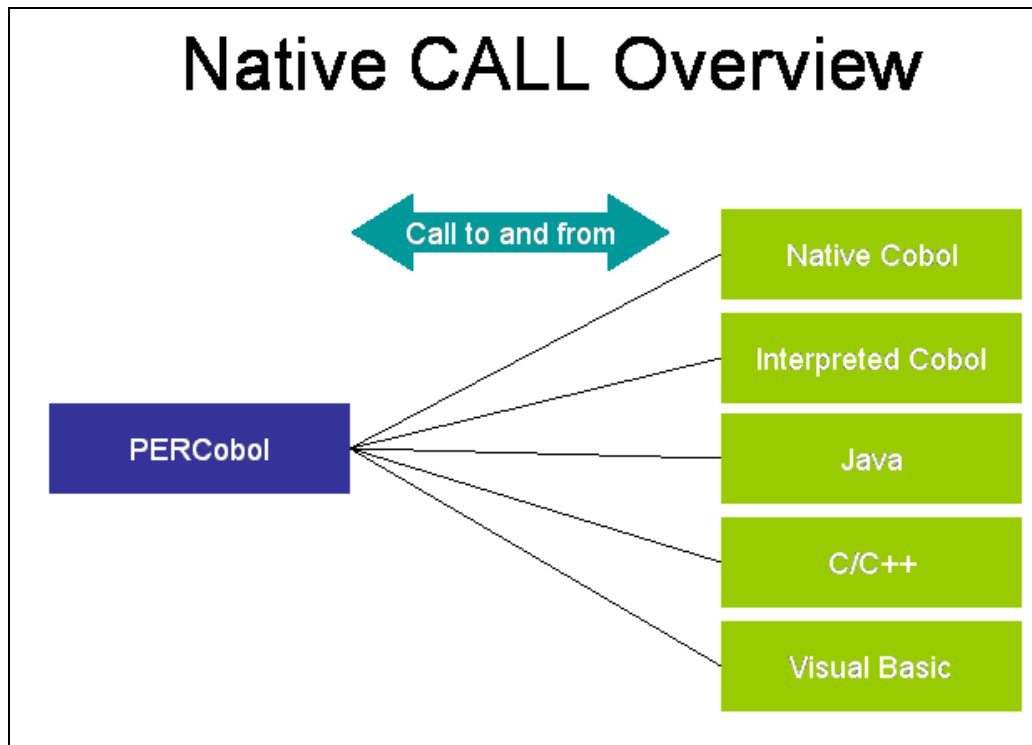
COBOL calls to COBOL, when both are compiled in PERCobol, are not significantly more complex than single COBOL programs. If the one COBOL program is compiled natively using a combination of COBOL compiler technologies.

COBOL calls to native code move the flow of control beyond the scope of the Java Virtual Machine. The native code may be written in a variety of languages, and since this moves beyond the platform independent nature of PERCobol, the capabilities vary per platform.

COBOL calls to Java are made through a variety of means, allowing either the COBOL side or the Java side to exist transparently of the other side.

Java calls to COBOL are enabled through the use of PERCobol. To the Java programmer, COBOL programs compiled with PERCobol are just like other Java classes.

PERCobol programs can be separated into server and client components and run remote from one another. If one or more program units are in Java the program interface is treated in a similar manner to a program unit compiled with PERCobol. A remote client or server in Java is a natural extension of the COBOL support within the PERCobol compiler.



## Java CALLing COBOL

### Passing Values from Java to COBOL

LegacyJ distributes a sample that illustrates how values are passed from Java to COBOL. This sample application is compiled with PERCobol and CALLs to Java to obtain values.

The CALL facility is accomplished by the Java program referring to an instance of the COBOL program. From the Java side, the class name will be obvious; it's the COBOL program's PROGRAM-ID in all lower-case with dashes converted to underscores. (The program id is all lower-case because COBOL is case insensitive.) The Java instance is created using the normal 'new' operation with an empty constructor.

The instance of the program is just a normal Java object. It has a well-defined interface called *com.legacyj.api.Callable* with a single method, named 'call' that takes a boolean[] and an Object[]. The boolean[] and Object[] should be the same length; each element is a parameter. The boolean is true for BY REFERENCE calling (actually copy/restore when calling from Java) or false for BY CONTENT calling, and the Object is the data to be passed. The Object must be a *java.lang.Number* descendent (Integer, BigDecimal, etc.), String or byte[]. The byte[] is treated as raw COBOL memory. It's passed as if via the COBOL MOVE verb. On the Java side, it just means value will be in a form usable from the Cobol side. The integer will be formatted according to whatever USAGE is in effect in the COBOL data automatically. On the Java side, please note that each Cobol data item is a full Java object, typed according to its usage, so it's smart about moving data.)

See the PERCobol IDE's Help Contents, PERCobol Programmer's Guide, Chapter 2 Program Lifecycle; it has a section on Java to COBOL calls and one on COBOL to Java calls. The

COBOL to Java calling section includes JavaDoc for the Callable and Datatype interfaces. The Datatype interface is primarily used when Cobol is CALLing Java; it allows the COBOL to pass datatypes and Java to convert them using toInt(), toString(), toText(), etc.

The sample has the COBOL and Java in the same project, but that need not be the case. If the COBOL and Java applications are in different projects, please note the following:

- A. In the Java project, right-click the Java project, properties, and set the Java Build Path to depend upon the other project. That will make it available to its CLASSPATH.
- B. In the COBOL project, set its Java Build Path setting to export the percobol.jar runtime.

The COBOL project is deployed to its own .jar file (that includes the PERCobol runtimes, licensing and the actual application). The Java side then just must include the deployed .jar in its classpath.

### ***Obtaining values from a String***

The method for obtaining the value from string1 into string2 with a mixed language application.

```
01 string1 pic x(80).  
01 string2 object reference "java.lang.String".
```

Objects are generally manipulated using INVOKE and SET rather than MOVE, for example:

```
invoke string2 using by value string1 giving string2
```

The statement above creates a java.lang.String object using the value of string1. The 'giving' points the object reference 'string2' to reference the new string object.

### ***Boolean in a COBOL Application***

A useful method for defining a Boolean in COBOL program is to use a traditional COBOL SWITCH is treated as a Boolean in invoke. Be aware, since Java has Boolean and boolean, that if the object you're invoking is typed, and it may do static invocation if possible, possibly using the wrong Boolean type.

If you INVOKE an OBJECT REFERENCE that is not given an explicit class name, then it is a dynamic invoke and the Boolean will be converted to lower-case boolean if necessary. (If this were for a CALL, then it doesn't matter; the Java can deal with it anyway.)

## COBOL CALLing COBOL

---

The following example illustrates a COBOL program calling another COBOL program and passing parameters. It illustrates how a native program can CALL a COBOL program compiled with LegacyJ PERCobol.

### The CALLer program

```
* cobolcaller.CBL - MAIN PROGRAM -- THE CALLING PROGRAM
*
* (C) Copyright LegacyJ Corporation 2005. All Rights Reserved.
*
* This file and associated files are copyrighted information
* of LegacyJ Corp. Permission is granted for usage in conjunction
* with the PERCobol product.
*
* Sample of cobol program calling a cobol subprogram and passing parameters both ways.
* Parameters include alphanumeric, numeric, binary, group, and java objects.
* The default passing convention is BY REFERENCE, not BY VALUE. That means that
* the address of the parameter is passed, not the actual literal value.
* So when a string object is passed, its address is passed. Because string objects
* in java are immutable, the strings cannot be changed.
*
* This program is the caller.
* The cobol caller program passes parameters to the cobol callee program.
* The callee then displays these parameters, and then changes the values
* of these parameters. When the caller program gets control again, it
* displays the values of these parameters.

IDENTIFICATION DIVISION.
PROGRAM-ID. cobolcaller.

data division.
working-storage section.
77 param-0 pic x(10) value "hello".
77 param-1 pic 999v99 value 123.45.
77 param-2 pic 999.99 value 678.90.
01 param-3.
   05 item-1 pic x(10) value "in group".
   05 item-2 comp-1 value 3.141.
   05 item-3 pic 999 value 987.
   05 item-4 pic s9(9) comp value 1234.
01 param-4.
   05 item-element pic x(10) occurs 10 times.
01 param-5 object reference "java.lang.String".
01 param-6 signed-int value -765.
01 param-7 object reference "java.lang.StringBuffer".
01 param-g object reference "java.lang.StringBuffer".

77 idx          pic s9(2).
77 ws-param-5   pic x(20).
77 move-param-5 pic x(20).

procedure division.
main-paragraph.
   display "cobolcaller.cbl starts 5:16 pm" upon sysout.
   move "a" to item-element(1)
   move "b" to item-element(2)
   move "c" to item-element(3)
   move "d" to item-element(4)
   move "e" to item-element(5)
   move "f" to item-element(6)
   move "g" to item-element(7)
   move "h" to item-element(8)
   move "i" to item-element(9)
   move "j" to item-element(10)
```

```

* Construct an object to pass as an example
  invoke param-5 using "Java String Object" giving param-5
  display "er: param-5=" param-5 upon sysout.

  invoke param-5 "toString" giving ws-param-5 end-invoke.
  display "er: ws-param-5=" ws-param-5 upon sysout.
  move param-5 to move-param-5.
  display "er: move-param-5=" move-param-5 upon sysout.

  invoke param-7 using "java StringBuffer Object" giving param-7.
  display "er: param-7=" param-7 upon sysout.
  display space upon sysout.

* ***** CALL SUBPROGRAM: *****
  call "cobolcallee" using
    param-0 param-1 param-2 param-3 param-4 by reference param-5 param-6 by
reference param-7
    giving param-g.

  display "CALLER is back" upon sysout.
  display "er: param-0=" param-0 upon sysout.
  display "er: param-1=" param-1 upon sysout.
  display "er: param-2=" param-2 upon sysout.
  display "er: item-1=" item-1 upon sysout.
  display "er: item-2=" item-2 upon sysout.
  display "er: item-3=" item-3 upon sysout.
  display "er: item-4=" item-4 upon sysout.

  perform varying idx from 1 by 1 until idx > 10
    display "er: item-element(" idx ")=" item-element(idx) upon sysout
  end-perform.

  display "er: param-5=" param-5 upon sysout.
  display "er: param-6=" param-6 upon sysout.
  display "er: param-7=" param-7 upon sysout.
  display "er: param-g=" param-g upon sysout.

  display space upon sysout.
  display "er: param-5 is the original string, not the string created by the
subprogram" upon sysout.
  display "er: In java, strings are immutable; i.e., cannot be changed" upon sysout.
  display space upon sysout.
  display "cobolcaller.cbl ends" upon sysout.

```

## *The CALLEE Program*

```

* cobolcallee.CBL - SUBPROGRAM
*
* (C) Copyright LegacyJ Corporation 2005. All Rights Reserved.
*
* This file and associated files are copyrighted information
* of LegacyJ Corp. Permission is granted for usage in conjunction
* with the PERCOBOL product.
*
* Sample of cobol program calling a cobol subprogram.
* This program is the callee, i.e., the subprogram.
* The cobol caller program passes parameters to the cobol callee program.
* The callee then displays these parameters, and then changes the values
* of these parameters. When the caller program gets control again, it
* displays the values of these parameters.

IDENTIFICATION DIVISION.
PROGRAM-ID. cobolcallee.

data division.
working-storage section.
77 idx          pic 99.
77 string-obj object reference "java.lang.String".

linkage section.
77 param-0 pic x(10).

```

```

77 param-1 pic 999v99.
77 param-2 pic 999.99.
01 param-3.
    05 item-1 pic x(10).
    05 item-2 comp-1.
    05 item-3 pic 999.
    05 item-4 pic s9(9) comp.
01 param-4.
    05 item-element pic x(10) occurs 10 times.

01 param-5 object reference.
01 param-6 signed-int.
01 param-7 object reference "java.lang.StringBuffer".
01 param-g object reference "java.lang.StringBuffer".

procedure division using
    param-0 param-1 param-2 param-3 param-4 by reference param-5 param-6 by reference
param-7
    giving param-g.

main-paragraph.
    display space upon sysout.
    display "cobolcallee.cbl starts 4:59 pm" upon sysout.
    display "ee: param-0=" param-0 upon sysout.
    display "ee: param-1=" param-1 upon sysout.
    display "ee: param-2=" param-2 upon sysout.
    display "ee: item-1=" item-1 upon sysout.
    display "ee: item-2=" item-2 upon sysout.
    display "ee: item-3=" item-3 upon sysout.
    display "ee: item-4=" item-4 upon sysout.

perform varying idx from 1 by 1 until idx > 10
    display "ee: item-element(" idx ")=" item-element(idx) upon sysout
end-perform.

display "ee: param-5=" param-5 upon sysout.
display "ee: param-6=" param-6 upon sysout.
display "ee: param-7=" param-7 upon sysout.
display "ee: param-g=" param-g upon sysout.
display space upon sysout.

display "AFTER CHANGES *****" UPON sysout.

move "callee chg" to param-0.
move 678.90 to param-1.
move 876.09 to param-2.
move "out caller" to item-1.
move 2.71727 to item-2.
move 789 to item-3.
move 4321 to item-4.
move "Apple" to item-element(1)
move "item 2 was b" to item-element(2)
move "item-element3 was 'c'" to item-element(3)
invoke string-obj using "callee string" giving param-5.
display "ee: param-5=" param-5 upon sysout.
move -2468123 to param-6.
invoke param-7 "append" using " & more stuff from subpgm" giving param-7.
display "ee: after append - param-7=" param-7 upon sysout.
invoke param-g using "a string from the subprogram" giving param-g.
invoke param-g "append" using " -- plus concatenate another string" giving param-g.
display "ee: param-g=" param-g upon sysout.

display "***** cobolcallee.cbl ends *****" upon sysout.
display space upon sysout.
goback.

```