

LegacyJ PERCobol FAQ -- Java CALLing COBOL

QUESTION:

Please tell us how to pass values from java to the COBOL program and get values back. An example of getting some parameters, loading up an array of value objects and returning this array back to java would be great.

ANSWER:

Be sure to first examine the Java Calling COBOL sample. Create a new 'LegacyJ PERCobol' project, call it java2cobol. In it, you'll find a .java file and a .cbl file.

A Java program creates 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. (It's all lower-case because COBOL is case insensitive.) The instance is created using the normal 'new' operation with an empty constructor.

That 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. From the Java side, this just means it 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. From the Java side, know 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.

This sample has the Cobol and Java in the same project, but that need not be the case. To be in different projects, as it looks like you'd like to do:

- 1) 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.
- 2) In the COBOL project, set its Java Build Path setting to export the *percobol.jar* runtime. This is done by checking the

PERCOBOL_CORE/percobol/percobol.jar entry in the Order and Export property tab. That will allow a Java program in the other dependent project to successfully execute the PERCObol programs.

When deploying, 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.

QUESTION:

If I have the following definitions in my working storage:

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

How can I get the value from string1 into string2? I can go the opposite way (*move string2 to string1*), but nothing I've tried seems to want to let me *move string1 into string2*.

ANSWER:

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

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

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

QUESTION:

Need help with using a boolean in my Cobol program. I have no problem defining it, I assume this is right:

```
01 my-flag object reference "java.lang.Boolean".
```

However, I'm not sure how to assign a value to it. I need to use it in a call to another Java object that was coded purely in Java (not that that matters

ANSWER:

There is another way to define a boolean, more useful for COBOL programs. 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, 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.)