Creating Applications Using Java and Micro Focus COBOL

Part 2
The COP Framework
Abstract

This is the second in a series of papers that examine how Micro Focus tools enable you to effectively use Java and COBOL together to develop new applications. The first paper focused on the basics of calling COBOL from Java and how data type conversion could be done to handle the differences in how Java and COBOL store data. This paper focuses on the problems of developing robust, multi-threaded applications, in particular, web-based applications and introduces a new technology from Micro Focus that solves these problems.
## Contents

Introduction ............................................................................................................................. 1

Going Beyond a Simple Application ...................................................................................... 2

The Problems ..................................................................................................................... 2

   In-Process or Out-of-Process ..................................................................................... 2

   The Problem of Multiple Threads ........................................................................ 2

   Use of JNI in J2EE Applications is Undesirable .............................................. 3

The Solution - The COP Framework ............................................................................... 3

   So what is the COP Framework? ...................................................................... 4

Example Code ............................................................................................................. 5

Developing an Application Using the COP Framework .............................................. 6

   Use the COP Package ....................................................................................... 7

   Establishing a Connection to the COP Server ................................................... 8

   Setting Up Parameter Lists ............................................................................... 8

   Calling the COBOL Program ............................................................................ 11

   Retrieving the Returned Parameter Values ...................................................... 11

   Destroying the Connection to the COP Server ................................................ 12

   Using this Class from a JSP-based Application ............................................. 12

   Running the COP Server ............................................................................... 13

Conclusion .................................................................................................................... 14
**Introduction**

This is the second in a series of papers that examine how Java and COBOL can be successfully used together to develop scalable, reliable applications for use in the enterprise. The number of programmers developing applications in Java has grown enormously over the last few years and that growth is expected to continue, with a projected number of 2.5 million programmers using Java by 2005. One of the primary reasons for this is that Java is not just a programming language, but also an entire architecture. Java 2 Enterprise Edition (J2EE) is becoming the mainstream enterprise application architecture. All of the leading J2EE Application Server vendors, including IBM with WebSphere, BEA with WebLogic and Oracle with Oracle9i Application Server are delivering solutions that can meet the largest customer’s speed and scalability requirements and these Application Servers are increasingly being adopted by corporations to implement the applications that run their business.

At the same time, much of the code that is running the business today is written in COBOL and, rather than attempting to rewrite all of this code, there is increasing demand to reuse as much as possible in new applications. In addition, the programmers who know the internal workings of business today are likely to be COBOL programmers, so providing a way to utilize their experience is highly desirable. A logical choice is to adopt a solution that allows Java and COBOL technologies to work together to produce highly scalable, reliable enterprise applications.

The first paper in this series\(^1\) looked at the problems inherent in Java interfacing with other languages and the solutions offered by Micro Focus with Micro Focus Net Express® and Server Express™. This paper examines the additional problems introduced by the introduction of J2EE Application Servers and multi-threaded applications and introduces a new technology from Micro Focus that provides a solution to these problems.

When reading this paper, you will find it helpful if you have some familiarity with the Java programming language. It is also recommended, but not required, that you read the first paper in this series, since the example code builds on techniques covered in that paper.

---

\(^1\) “Creating Applications using Java and Micro Focus COBOL – Part 1”, http://www.microfocus.com/whitepapers
Going Beyond a Simple Application

The first paper discussed how a simple Java-based graphical application could be written that calls a COBOL program to store and retrieve data from indexed-sequential files. This application demonstrated the core techniques involved in packaging the COBOL program and conversion of data-types.

However, the development of more complex applications introduces new problems to be solved. This section looks at those problems and examines what is needed to solve them.

The Problems

In-Process or Out-of-Process

If you develop an application using the techniques described in paper 1, the COBOL program will run in-process and the communication with the COBOL program is via Java Native Interface (JNI) calls. This means that the COBOL code runs in the same process as the Java virtual machine that calls it. Although in-process calls execute quickly, if the COBOL program crashes for any reason, it may stop the Java application that called it as well. Obviously, in the case of a J2EE Application Server, which may be running many applications and serving a large number of users, this is not a desirable prospect.

The solution to this is to run the COBOL program in a different process on the computer. This is called out-of-process execution. While calls to the COBOL program may be slightly slower because of the need to cross process boundaries, if the COBOL program crashes, only the process it is running in will be terminated and the calling process will be unaffected. So, for reliable systems, out-of-process execution of the COBOL code is highly desirable. In this scenario, a separate server process would load and call the COBOL code. The Java applications would communicate with this server using standard Java cross-process communication mechanisms.

The Problem of Multiple Threads

More complex Java applications usually involve the use of multiple threads. Java was designed from the ground up with multi-threading in mind and the language makes the creation and control of multiple threads easy to do. Therefore, it is common for more complex Java applications to make extensive use of threads. This is particularly true in web-based applications running under a J2EE Application Server, which will use multiple threads to serve multiple users. In addition, to conserve resources, most Application Servers reuse threads. So, it is possible for the same instance of a COBOL program to be called from different threads.
The COBOL run-time system allocates working storage and other data areas on a per-thread basis. This means that when the COBOL program is first called in a particular thread, a new copy of working storage is allocated and associated with that thread. This works fine if the COBOL program is always called from the same thread. However, if it is called from different threads, different copies of working storage will be used in subsequent calls to the program. This is not an issue if the COBOL program is “stateless”, that is, it does not need to store any information between calls to the program. Such a program might be one that is called to just perform one particular calculation based on data passed to the program. However, if the program provides a number of functions that rely on data stored in working storage between calls, then this is a problem. Such programs are often known as “stateful” programs.

Running the program out-of-process from a separate server program provides a means of solving this problem. The server can control the access to the COBOL program, ensuring that a particular instance of the COBOL program is always called from the same thread.

Use of JNI in J2EE Applications is Undesirable

The use of the JNI in an application does not conform to the J2EE specification. Therefore, the use of JNI in an application that is going to run under a J2EE Application Server is undesirable. In addition, the extra environment information that is needed (for example, the location of the native code modules) can be difficult to configure. So the use of a “pure Java” solution to invoking the COBOL program is desirable for many enterprises.

The Solution - The COP Framework

Hopefully, after reading the above section, it has become obvious that a solution that allows the COBOL program to be run out-of-process is desirable in many cases and necessary if you want to develop an application using J2EE technologies. The J2EE specification provides for a mechanism to achieve this. This is the J2EE Connector Architecture. The J2EE Connector Architecture provides a Java solution to the problem of connectivity between Application Servers and the many different non-Java systems that an enterprise may want to utilize from their new Java-based systems. In order to use the J2EE Connector Architecture, a resource adapter is needed. The resource adapter plugs in to the application server, providing connectivity between the Application Server and the application that is being called and handling connection, transaction and security management.

Micro Focus is developing a resource adapter that will enable COBOL programs to be used with J2EE Application Servers. This resource adapter will be delivered alongside Net Express 4.0 and Server Express 4.0. However, an interim solution has been developed to enable the COBOL program to be run out-of-process today. This is the COBOL Out-Of-Process Framework (COP Framework). Although the COP Framework is described as an interim solution, it will be supported going forward. The reasons for this are:
• The J2EE Connector Architecture is a recent addition to the J2EE specification and only the most recent versions of J2EE Application Servers support it. Therefore, use of the J2EE Connector Architecture may not be a viable option for users of older versions of Application Servers.

• Some customers may not require the complete J2EE environment, but are developing multi-threaded Java applications that need to call COBOL programs. For these users, the COP Framework provides a lightweight solution that does not require the overhead of a complete J2EE solution.

So What is the COP Framework?

The COP Framework consists of two parts:

• The COP Server. This is a separate application that loads and runs COBOL programs in a separate process to the Application Server.

• The COP Client. The client provides a 100% pure Java API (Application Programming Interface). This is used by the Java applications to connect to the COP Server and make calls to the COBOL programs that are being run by the COP Server.

The COP Server is started independently of the Application Server or main Java application. It runs continuously and loads and calls COBOL programs as requested by the Java application via calls to the COP Client API. Figure 1 shows a general overview of the components involved in an application running under the COP Framework.

The COP Framework is available as an installable add-on for Net Express Service Pack 1 that can be downloaded from the Micro Focus website. The COP Framework includes the COP Server, the files needed for the COP Client APIs and full documentation on the COP Client APIs.
Example Code

Included with this paper is a complete example application developed using the COP Framework. It is an extension of the example program developed for the first paper. It is a very simple classified advertisement system for a newspaper called The Corgi Times that allows the user to view advertisements in the system and place new advertisements. For paper 1, the user interface for this application was written with a graphical user interface using the Java Swing APIs. For this paper, the application has been changed to be a web-based application that implemented on top of an Application Server (the installed version uses the Tomcat Application Server developed under the Apache Open Source project). It uses Java Server Pages (JSP) to provide the user interface. The back-end processing code that retrieves and places advertisements is unchanged from the first paper. It is written in COBOL and is in the form of a simple, called COBOL program with a number of entry points. All data is stored in COBOL indexed-sequential files.
Developing an Application Using the COP Framework

To make any call to a COBOL program using the COP Framework client APIs, the program must do the following:

- Establish a connection to the COP Server
- Set-up a parameter list for the input parameters (not needed if the program does not take any parameters)
- Set-up a parameter list for the output parameters (not needed if the program does not take any parameters)
- Send a request to the COP Server to call the COBOL program
- Retrieve the values of any output parameters or return code
- Destroy the connection to the COP Server

In between establishing a connection to the COP Server and destroying the connection, you can make as many calls to the COBOL program as you wish. The process is the same for each call – set-up input and output parameter lists, call the program and then retrieve any output parameters.

The following is a section from the wrapper class for the COBOL program that is included with the Corgi Times application mentioned previously. All access to the COBOL program is made via this class. The code will be covered in detail in the following sections.

```java
package com.microfocus.demo.classifieds;
import com.microfocus.demo.coboltypes.*;
import com.microfocus.cobol.cop.*;
import com.microfocus.cobol.cop.lang.*;
import java.net.InetAddress;
import java.util.Vector;
import java.util.Iterator;
public class Classifieds
{
    private Vector Categories;
    private CategoryRecord[] CategoryArray;

    public Classifieds() throws Exception
    {
        Categories = new Vector();
        // Initialize the array to be used when calling
        // GetCategories
        CategoryArray = new CategoryRecord[20];
        for (int i = 0; i < 20; i++)
            CategoryArray[i] = new CategoryRecord();
    }

    // Get the categories of ads that are stored. Returns an
    // iterator to the list of category records
    public Iterator getCategories() throws Exception
```
int categoryCount;
Integer returnStatus = new Integer(0);
COBOLArray COBOLCategories =
    new COBOLArray(CategoryArray);
pic9 COBOLCategoryCount = new pic9(2);

ClientRequest request =
    new ClientRequest(InetAddress.getLocalHost(),
    9200);
request.connect();
// Create the parameter list and add the parameters
// to it
ParameterList inputParms = new ParameterList();
inputParms.add(returnStatus);
inputParms.add(COBOLCategories);
inputParms.add(COBOLCategoryCount);
ParameterList outputParms = new ParameterList();
request.cobcall("GetCategories",
    inputParms,
    outputParms);
// Get the returned values
returnStatus = (Integer)outputParms.getArgument(0);
COBOLCategories =
    (COBOLArray)outputParms.getArgument(1);
COBOLCategoryCount =
    (pic9)outputParms.getArgument(2);
request.dispose();
if (returnStatus.intValue() != 0)
    categoryCount = 0;
else
    categoryCount = COBOLCategoryCount.intValue();
}
// Setup the vector
Categories.clear();
for (int i=0; i < categoryCount; i++)
    Categories.add(CategoryArray[i]);
// and return an iterator to the categories
return Categories.iterator();
}

---

**Note:** The example program contains additional exception handling code that has been removed here for clarity.

---

**Use the COP Package**

The first point to notice in this class is the lines:

```java
import com.microfocus.cobol.cop.*;
import com.microfocus.cobol.cop.lang.*;
```

These make all of the classes required for the COP Client API provided by Micro Focus available to your Java code.
These classes are supplied in the file COPClient.jar, which is provided with the COP Framework. When you compile an application that uses the COP Framework, you need to make this jar file available to the Java compiler by specifying it on the classpath for the compiler. COPClient.jar also needs to be distributed with your application. If you are creating an application using Java Server Pages or Java servlets to run under an Application Server, then the COPClient.jar file can be included in the .war package for your application.

Establishing a Connection to the COP Server

The COP Client communicates with the COP Server via a socket. The COP Server listens on a particular socket for clients that want to connect to it. To call the server, the COP Client needs to know the DNS name or IP address of the machine it is calling and the port number. Usually, the COP Server will be on the same machine as the calling Java application, so you can simply use “localhost” as the name of the machine. The port number used can be changed by specifying options when the COP Server is started, but by default it is port 9200.

Once you know these two pieces of information, you can connect to the COP Server by creating a new instance of the ClientRequest class and invoking the connect method. For example:

```java
ClientRequest request = new ClientRequest(InetAddress.getLocalHost(), 9200);
request.connect();
```

This will connect to a COP server running on the same machine as the COP client, on port 9200.

You can also specify the port and machine information using separate calls if you wish. For example, the following code achieves the same result as the code above:

```java
ClientRequest request = new ClientRequest();
request.setMachine(InetAddress.getLocalHost());
request.setPort(9200);
request.connect();
```

Setting Up Parameter Lists

Once you have a connection to the COP server, if your COBOL program requires parameters to be sent to it, you need to set-up the lists of parameters for the call to the COBOL program. Two parameter lists are needed, one for input parameters and one for output parameters. The input parameter list is used to pass the values of the parameters to the COBOL program and the output parameter list is used to return values from the COBOL program. The input parameter list is created by creating a new instance of the class ParameterList and then adding the parameters to it. For example:
Integer returnStatus = new Integer(0);
COBOLArray COBOLCategories = new COBOLArray(CategoryArray);
Pic9 COBOLCategoryCount = new Pic9(2);

ParameterList inputParms = new ParameterList();
inputParms.add(returnStatus);
inputParms.add(COBOLCategories);
inputParms.add(COBOLCategoryCount);

Note that the add method returns a reference to the ParameterList, so the code for setting up the parameter list can be abbreviated to:

ParameterList inputParms = new ParameterList()
    .add(returnStatus)
    .add(COBOLCategories)
    .add(COBOLCategoryCount);

The parameters are added to the list in the order they will be passed to the COBOL program. A parameter can either be a Java basic type (such as int, long, etc) or it can be an object. For complex types, where conversion is needed between the Java type and the type expected in the COBOL program, you can use a class that implements the interface DataType (for a full description of the DataType interface, refer to the first paper in this series).

For example,

    short a = 12;
    inputParms.add(a);

will pass a short value as a parameter.

Using the add() method as shown above will add the parameter to the parameter list as a BY REFERENCE parameter. If you want to specify a different usage type such as BY VALUE or BY CONTENT, an alternative form of the add() method is provided that enables you to specify the usage type. For example:

    short a = 12;
    inputParms.add(a, ClientRequest.BY_VALUE);
    Pic9 b = new Pic9(2);
    inputParms.add(b, ClientRequest.BY_CONTENT);

will pass parameter a by value and parameter b by content. The default usage type is BY REFERENCE, so using:

    inputParms.add(a, ClientRequest.BY_REFERENCE);

is functionally the same as using:

    inputParms.add(a);
Note: There are some important changes between the classes that are used for data type conversion if you are using the COP Framework and the classes used in the first paper. The DataType interface to use with the COP Framework is in a different package. You should now use

    com.microfocus.cobol.cop.lang.DataType

instead of:

    mfcobol.lang.DataType

To do this, in most instances, you will simply need to change the import statement in your source code from:

    import mfcobol.lang.*;

to:

    import com.microfocus.cobol.cop.lang.*;

In addition, because the COP Framework may need to make copies of the objects, the classes must either have a default constructor or implement the interface Cloneable. You can see examples of this in the classes provided with the example.
To create the output parameter list, all you need to do is create an empty instance of the ParameterList class, for example:

```java
ParameterList outputParms = new ParameterList();
```

### Calling the COBOL Program

Now, we can make the call to the COBOL program. This is done using the cobcall method in the COP client API. If your COBOL program does not require any parameters, you simply specify the name of the program. For example,

```java
request.cobcall("cobolprogram");
```

If your COBOL program requires parameters, you provide the input and output parameter lists, for example:

```java
request.cobcall("GetCategories",
               inputParms,
               outputParms);
```

If your program sets a return code, it will be passed back as an integer by cobcall. For example:

```java
int return-code = request.cobcall("cobolprogram");
```

### Retrieving the Returned Parameter Values

Once the call has been made to the COBOL program, the returned values of the parameters will be in the output parameter list. The return value for each parameter can be obtained using the method getArgument and specifying the position of the parameter you want in the list, starting from 0.

For example:

```java
returnStatus = (Integer)outputParms.getArgument(0);
```
returns the value of the first parameter and

```java
COBOLCategoryCount =
          (pic9)outputParms.getArgument(2);
```
returns the value of the third parameter.

Note that getArgument() returns a value of type Object, so you must use a cast to convert it to the correct type. If the corresponding input parameter was one of the Java basic types, it will be returned as an instance of the appropriate Java wrapper type. For example, an input parameter of type int, will be returned as an object of type Integer. The value of the Java basic type can be retrieved using the appropriate methods in the wrapper class. For example:
Short shortA = (Short)outputParms.getArgument(0);
short a = shortA.shortValue();

**Destroying the Connection to the COP Server**

Finally, when you have no more calls to make to the COBOL program, you must destroy the connection to the COP Server. This is done using the dispose() method. For example:

```java
request.dispose();
```

Once you have made a call to dispose, you will need to reconnect to the COP Server in order to use it again.

**Using this Class from a JSP-based Application**

Now that you have created the wrapper class for your COBOL program, it can be used just like any other Java class in your application. For example, the following is an example of a simple Java Server Page that uses the method getCategories to retrieve the categories of advertisements.

```html
<html>
<body>
<%@ page import="java.util.*,java.text.*, com.microfocus.demo.classifieds.*" %>
<%@ page errorPage="errorpage.jsp" %>
<jsp:useBean id="Classifieds"
    class="com.microfocus.demo.classifieds.Classifieds"/>
<table border=0 cellspacing=0 cellpadding=0
    width="100%">
    <tr>
        <td width="100%" height=67>
            <p align="left">
                <font size="5">Categories</font></p>
    </td>
</tr>
<% Iterator categoryIterator = Classifieds.getCategories();
    while (categoryIterator.hasNext()) {
        CategoryRecord category = (CategoryRecord)categoryIterator.next();
        String categoryName = category.getCategoryName();
        int categoryAdCount = category.getCategoryAdCount();
        if (categoryAdCount > 0) {
            %>
            <a href="viewads.jsp?category=<%= categoryName %>"><%= categoryName %>(<%= categoryAdCount %>)</a>
            %>
        } else {
        }
```
If you are unfamiliar with using Java Server Pages, it is a mechanism where Java code can be embedded in HTML pages to perform scripting operations. When the page is requested from the server, it is compiled into a Java servlet and executed, generating the HTML code that is passed back to the browser that requested it. This page simply calls the getCategory method in the Classifieds class and then iterates through the results, building an HTML table that lists each category and the number of advertisements in that category. If there are advertisements (categoryAdCount > 0), then the category name is turned into a hyperlink to a page that will list the advertisements for that category.

Note: It is generally accepted good practice not to place Java code inside your Java Server Pages unless absolutely necessary. However, it has been done here to keep the example simple.

Running the COP Server

Finally, before your application can be executed, the COP Server must be started on your computer in a separate process. For information on how to run this on your particular operating system, refer to the COP Framework documentation. Any environment information that is required by your COBOL program (for example, the name of files defined as external) can be set in this process before the COP Server is started.
Conclusion

This paper has introduced the problems involved in reusing COBOL programs in applications that use J2EE Application Servers. The COP Framework from Micro Focus provides a solution to the problems and is available today. Longer term, Micro Focus will introduce support for the J2EE Connector Architecture, but applications developed using the COP Framework will continue to run and, over time, can be migrated to the new architecture.

-Written by Wayne Rippin, Consultant

About the author: Wayne Rippin is a self-employed consultant. Previously, he worked for Micro Focus for 16 years, first as a systems programmer and later as a product manager. His most recent role there was director of product management, leading a team of product managers responsible for Net Express, Mainframe Express and UNIX compiler products.
Micro Focus is the fastest, most effective way to migrate, extend, develop and deploy enterprise applications. As the industry standard for COBOL, Micro Focus enables organizations to maximize ROI while reducing development costs for accelerated business success. Founded in 1976, Micro Focus is a global company with principal offices in the United Kingdom, United States and Japan. For more information, visit www.microfocus.com.

Micro Focus Worldwide

<table>
<thead>
<tr>
<th>Country</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>1800 632 626</td>
</tr>
<tr>
<td>Austria</td>
<td>0800 293 535</td>
</tr>
<tr>
<td>Belgium</td>
<td>0800 11 282</td>
</tr>
<tr>
<td>Canada</td>
<td></td>
</tr>
<tr>
<td>Ontario</td>
<td>905 824 7397</td>
</tr>
<tr>
<td>Quebec</td>
<td>800 280 9487</td>
</tr>
<tr>
<td>France</td>
<td>0800 916 564</td>
</tr>
<tr>
<td>Germany</td>
<td>0800 182 5443</td>
</tr>
<tr>
<td>Ireland</td>
<td>+353 1 469 3121</td>
</tr>
<tr>
<td>Italy</td>
<td>800 784 420</td>
</tr>
<tr>
<td>Japan</td>
<td>+81 3 5793 8550</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>800 23743</td>
</tr>
<tr>
<td>Netherlands</td>
<td>0800 022 8562</td>
</tr>
<tr>
<td>Norway</td>
<td>+47 22 91 07 20</td>
</tr>
<tr>
<td>Sweden</td>
<td>+46 8 545 13 390</td>
</tr>
<tr>
<td>Switzerland</td>
<td>0800 564 247</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>0800 328 4967</td>
</tr>
<tr>
<td>United States</td>
<td>877 772 4450</td>
</tr>
<tr>
<td>Other Countries</td>
<td>+44 1635 32646</td>
</tr>
</tbody>
</table>

© 2003 Micro Focus. All Rights Reserved.
Micro Focus and Net Express are registered trademarks, and Server Express is a trademark of Micro Focus. Other trademarks are the property of their respective owners.

WPJACB0103-US