Redpaper

Daniel Donnelly

Updating Rules for COBOL Generated Code to Use CICS Channels and Containers

IBM® WebSphere® ILOG Rules for COBOL generates a COBOL sub-program from the rules that have been externalized and managed in the ILOG Rules Repository. The externalization of business rules provides benefit to your business, as well as improves the efficiency of maintaining your COBOL applications. Corporations today continue to see value in running their core business applications on large-capacity mainframes. An organization's success depends largely upon its ability to respond quickly to today's complex, ever changing markets and regulatory climate. Businesses today need a Business Rule Management Systems (BRMS) solution that supports business changes and cycles far more effectively than traditional methods and takes direct advantage of business expertise.

A true BRMS provides the technology to manage business logic separately from application code with minimum disruption to the application when the business logic changes. This IBM Redpaper™ describes how you can take COBOL code generated by WebSphere ILOG Rules for COBOL and modify it to use CICS® channels and containers. By wrapping the Rules for COBOL program, we demonstrate how CICS channels and containers can be used to remove the need to statically link a Rules for COBOL module with its calling programs. If a Rules for COBOL module is used by multiple calling modules you can reduce the

amount of relinking that must be performed every time that a Rules for COBOL module changes. Therefore, instead of having to link the changed COBOL module into every single application that calls it, a single link step is performed to link the Rules for COBOL module into your Load Library. After the program associated with the Rules for COBOL module is reloaded, the changed code is instantly available to all applications that call it. An additional advantage of this method of structuring a Rules for COBOL generated program is that it does not have to run in the same CICS region as its calling program. Therefore, greater flexibility is allowed in how an application is architected. For example, multiple regions running the Rules for COBOL program can be used in conjunction with CICSPlex® System Manager Workload Management to route requests to maximize availability, optimize response times, and optimize the use of the capacity of the system on which the rules are running. CICSPlex System Manager Workload Management monitors the status and health of the regions and automatically routes requests to the most appropriate region. For example, if a region goes Short-on-Storage or reaches its MAXTASK limit, CICSPlex System Manager Workload Management automatically stops routing rules requests to that region.

This document briefly describes WebSphere ILOG Rules for COBOL, as well as CICS channels and containers. It then goes on to describe how to create a wrapper program for a Rules for COBOL generated program, and how to modify an associated calling program to allow the use of CICS channels and containers.

WebSphere ILOG Rules for COBOL

WebSphere ILOG Rules for COBOL is an extension to WebSphere ILOG JRules. It allows you to create and manage business rules that are based on COBOL data structures. Business rules can be authored using business terminology and then deployed across multiple platforms. By using Rules for COBOL, business rules can be deployed as a COBOL module for use by mainframe applications running in CICS, IMS™, or batch environments. It is used in conjunction with a number of other components to provide a complete BRMS solution. Two of the most important components with which it interacts are WebSphere ILOG Rule Studio and WebSphere ILOG Rule Team Server. Rule Studio is an Eclipse-based environment that is used to develop and deploy business rules. In Rule Studio, rule developers define the business vocabulary to be used, write rules based on that vocabulary, and deploy the rules. Rule Team Server is a Web-based interface that allows business users and others to collaborate in creating, modifying, and deploying business rules.

To create business rules using Rules for COBOL, a Business Object Model (BOM) is created based on a COBOL copybook that describes the data

structures involved in your business decisions. Rule Studio or Rule Team Server can then be used to create, edit, and manage business rules based on the BOM. Rule Studio can be used to generate COBOL source code that enforces the business rules. This COBOL source code can then be transferred and compiled to the mainframe for use with your mainframe applications.

CICS channels and containers

Channels and containers, like COMMAREAs before them, are used to pass data between programs in CICS. Channels and containers were first made available in CICS TS Version 3.1. Prior to this version, program-to-program communication only occurred through COMMAREAs. *COMMAREAs* are pieces of storage that get passed between programs. A calling program passes data to a called program by adding the data to a storage area and passing this data on an EXEC CICS LINK call to it. The called program can then modify this data and add its response to the same piece of storage. This storage then is made available to the calling program when the called program has finished execution.

COMMAREAs can still be used to pass information between programs today; however, they are limited to 32 KB. Channels and containers were introduced to allow programs to pass more than 32 KB of data to each other. With channels and containers, you can pass data of any size between programs as long as enough overall storage is available to the CICS region to process the request.

A *channel* is a group of containers. A channel can have any number of containers associated with it. A *container* is a piece of storage that we want to pass from one program to another program.

Why use CICS channels and containers

Currently with Rules for COBOL, the generated COBOL program is statically linked into the programs that call it. Therefore, when the generated program changes, it must be relinked with all its calling programs to allow the change to take effect. See Figure 1 on page 4.

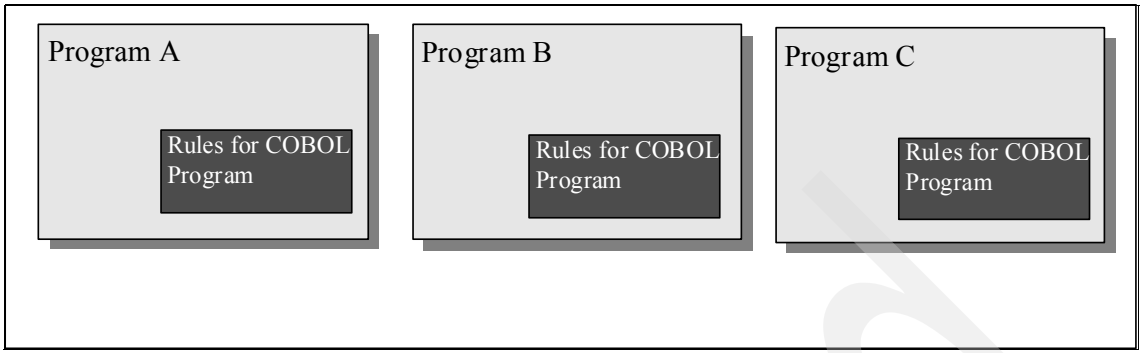

Figure 1 Rules for COBOL generated program statically linked with its calling programs

By using CICS channels and containers, we can perform an EXEC CICS LINK to the generated program. We can compile and link the generated program completely independently of its calling programs as long as there have been no changes to the copybook that the programs use. By using channels and containers, we can avoid having to relink the generated code with each calling program. See Figure 2.

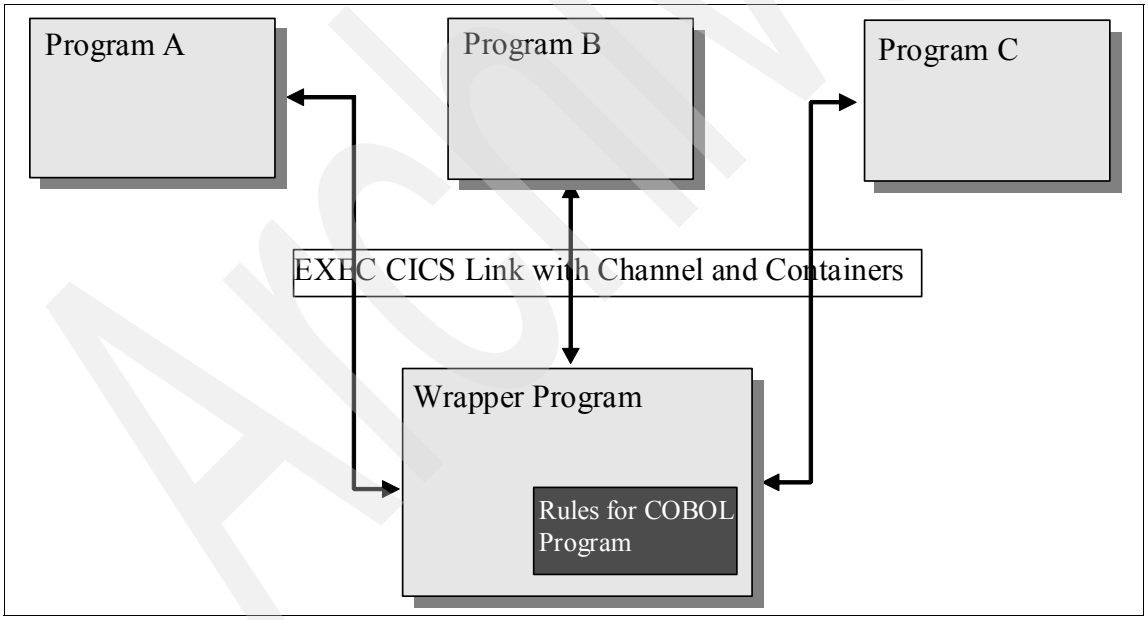

Figure 2 Rules for COBOL generated program linked with a wrapper program

By performing an EXEC CICS LINK to a Rules for COBOL generated program, you can have the calling program in a language other than COBOL. You can have a COBOL application, a C application, and a PL/I application all calling the same Rules for COBOL generated modules to make business decisions.

An additional advantage of using channels and containers with Rules for COBOL is that the generated Rules for COBOL program no longer has to reside in the same CICS region as the calling program. You might want to create a *Rules Owning Region*, which is a CICS region that is dedicated to running your Rules for COBOL programs. The advantage of using a Rules Owning Region is that it allows you to modify and manage all your rules programs on a single region. Multiple cloned Rules Owning Regions can also be used in conjunction with CICSPlex System Manager Workload Management to achieve the best throughput in your applications.

The scenario

This example of using CICS channels and containers is based on the code that is generated by the Rules for COBOL tutorial, which is provided in the Rules for COBOL documentation at this Web site:

http://www.ilog.com/dev/brms/media/ilog_rules_for_cobol_user_guide.pdf

In this example, a loan company has a COBOL application that validates loan applications. The company wanted to share the rules that govern loan validation among multiple applications (for example, between their existing COBOL application and a new Java™-based application). They also want to be able to manage these rules from a central location. The first step that they take is to create a copybook called `miniloan.cpy` that describes the data structures involved in the business rules of the application. Example 1 shows this copybook.

Example 1 Copybook

```
01  Borrower.
   05 name PIC X(20).
   05 creditScore PIC 9(10).
   05 yearlyIncome PIC 9(10).
   05 location PIC X(3).
   05 age   PIC 9(3).
      88 teenager VALUE 0  THRU 17.
      88 adult VALUE 18 THRU 60.
      88 retired VALUE 61 THRU 150.
01  Loan.
   05 amount PIC 9(10).
   05 yearlyInterestRatePIC 99.
```

```
   05 yearlyRepayment PIC 9(10).
   05 effectDate PIC x(8).
   05 approved PIC X.
   05 messageCount PIC 9(2).
   05 messages PIC X(60) OCCURS 0 TO 99
            TIMES DEPENDING ON messageCount.
```

The loan company then uses this copybook in conjunction with Rules for COBOL to produce a set of rules that decides if a particular borrower qualifies for a particular loan. These rules are then used to generate a COBOL program called `miniloan` that enforces these rules. Example 2 shows a section of the code that is generated by Rules for COBOL.

Example 2 Data division

```
DATA DIVISION.
  WORKING-STORAGE SECTION.
      COPY LOCLMINI.
  01 LOCALAREA.
      02 MAPPEDMETHOD1-MSG PIC X(40).
      02 STRTEMP1 PIC X(33) VALUE
      "The age exceeds the maximum line.".
      02 STRTEMP2 PIC X(30) VALUE "The loan cannot exceed 1000000".
      02 STRTEMP3 PIC X(28) VALUE "Too big Debt-To-Income ratio".
      02 STRTEMP4 PIC X(22) VALUE "Credit score below 200".
      02 STRTEMP5 PIC X(48) VALUE
      "The income after tax is lower than basic request".
      02 METHOD3-INCOMEAFTERTAX PIC 9(10).
      02 METHOD3-TABLE OCCURS 100 TIMES.
          03 METHOD3-SUB-ITEM PIC 9(10).
      02 METHOD3-IX PIC 99 VALUE 0.
  LINKAGE SECTION.
     COPY MINILOAN.

  PROCEDURE DIVISION USING BORROWER LOAN.
  * Task: mainflow
  TASK1-MAINFLOW.
      PERFORM TASK2-MAINFLOW-VALIDATION
      PERFORM TASK3-MAINFLOW-ELIGIBILITY
      GOBACK.
….
….
….
```

Any programs that want to call this COBOL program must have it statically linked in. Example 3 shows part of the program that performs this sort of call to the generated COBOL module.

Example 3 Procedure division

```
….

Procedure Division.
    Root.
        move "John" to name of Borrower
        move 100 to creditScore of Borrower
        move 10000 to yearlyIncome of Borrower

        move 10000 to amount of Loan
        move 5 to yearlyInterestRate of Loan

        move 1000 to yearlyRepayment of Loan
        move "N" to approved of Loan
        move 0 to messageCount of Loan

        call "miniloan" using Loan Borrower

        if approved of Loan = "Y"
            display "Loan approved"
        else
            display "Loan not approved"
            move 1 to tableIndex
            perform until tableIndex > messageCount of Loan
                display messages of Loan (tableIndex)
                compute tableIndex = tableIndex + 1
            end-perform
        end-if
        goback.
```

You can compile, link, and run this code inside a CICS system; however, the generated COBOL code must be statically linked with every program that calls it. We now show how the previous code snippets can be modified to allow the generated COBOL code to be handled like a separate, linkable, CICS program.

Naming the channels and containers

Each channel is identified by a name that is up to 16 characters long. In this example, we have chosen the project name (MINILOAN) with the characters -CHANNEL attached to the end to be the name of the channel. Regarding the containers, you must first decide how many containers will exist and what data will be passed in each container. We can pass all the data in a single container on the channel; however, in our example, this approach does not work. In our example, there are two top-level data items: Borrower and Loan. If we use a single container, we must put these items into a contiguous piece of storage before putting them into the container. On the other side or the other end, a program using this container will have to determine where one data item stops and the second data item begins. In our example, it makes sense to have a container for each top-level data item: one container for the Borrower data item and one container for the Loan data item.

Each container is identified by a name that is up to 16 characters long. In this example, we use the names of the top-level group items (LOAN and BORROWER) with the characters -CONTAIN attached to the end to be the name of our two containers.

It is good practice to declare the names of your channels and containers as constants in a copybook that is shared by both the calling program and the called program. For the purposes of this example, we create a new copybook called chnlcont.cpy. Example 4 shows our updated copybook.

Example 4 Updated copybook

```
01  MINILOAN-CHANNEL         PIC X(16) VALUE 'MINILOAN-CHANNEL'
01  LOAN-CONTAINER              PIC X(16) VALUE 'LOAN-CONTAIN'
01  BORROWER-CONTAINER  PIC X(16) VALUE 'BORROWER-CONTAIN'
```

Creating a wrapper program

To avoid directly changing the generated Rules for COBOL code, we create a wrapper program. By minimizing changes to the generated code, we can avoid introducing unintentional errors into the generated code.

The wrapper program is responsible for performing the following tasks:

- Retrieving information from the containers that are passed to it from a calling program
- Putting the information received into an appropriate structure to pass to the generated Rules for COBOL program
- Putting any data modified by the generated program into a container to return it to the calling program

In this example, we name the wrapper program `CICSWRAP`. The program will include the `miniloan` copybook, as well as the `chnlcont` copybook. The program's first action is to get the data from the containers given to it by the calling program. The EXEC CICS GET CONTAINER statement achieves this action. We need two of these statements at the start of our wrapper program: one statement to get data from the `LOAN-CONTAIN` container and one statement to get data from the `BORROWER-CONTAIN` container. Example 5 shows the start of the wrapper program.

Example 5 Start of the wrapper program

```
Identification Division.
Program-ID. CICSWRAP.

Data Division.
Working-Storage Section.
    Copy "miniloan".
    Copy "chnlcont".

Procedure Division.
    Root.

        EXEC CICS GET CONTAINER(LOAN-CONTAINER)
                                  CHANNEL(MINILOAN-CHANNEL)
                                  INTO(Loan) END-EXEC

        EXEC CICS GET CONTAINER(BORROWER-CONTAINER)
                                  CHANNEL(MINILOAN-CHANNEL)
                                  INTO(Borrower) END-EXEC

.......
.......
```

The next step in creating the wrapper program is to make a COBOL call to the Rules for COBOL generated code. After we have made this call, we need to put any changed data back into a container so that it can be passed back to our calling program. By examining our generated code, we can see that there are statements in TASK2-MAINFLOW-VALIDATION and TASK3-MAINFLOW-ELIGIBILITY that might cause changes to be made to the Loan data. We must put the Loan data structure back into the Loan container to allow it to be passed back to the calling program. Data is put into a container by using the EXEC CICS PUT CONTAINER command, so we add this command after the call to the generated module. The final change that we need to make to the generated code is the addition of a RETURN statement that informs CICS that control needs to be passed back to the calling program. Example 6 shows the changes to the wrapper program. The changes are highlighted in bold.

Example 6 Changes to the wrapper program

```
Identification Division.
Program-ID. CICSWRAP.

Data Division.
Working-Storage Section.
    Copy "miniloan".
    Copy "chnlcont".

Procedure Division.
    Root.

        EXEC CICS GET CONTAINER(LOAN-CONTAINER)
                                  CHANNEL(MINILOAN-CHANNEL)
                                  INTO(Loan) END-EXEC

        EXEC CICS GET CONTAINER(BORROWER-CONTAINER)
                                  CHANNEL(MINILOAN-CHANNEL)
                                  INTO(Borrower) END-EXEC

        call "minicbl" using Loan Borrower

        EXEC CICS PUT CONTAINER(LOAN-CONTAINER)
                                    CHANNEL(MINILOAN-CHANNEL)
                                    FROM(Loan) FLENGTH(Length of Loan)
                                    END-EXEC

        EXEC CICS RETURN END-EXEC
```

Creating a channel and the containers

We need to change the calling program. The calling module is responsible for creating the channel and containers and calling the wrapper program.

Data is added to a container in a channel by using the EXEC CICS PUT CONTAINER command. This CICS API command first checks to see if a named channel or a named container already exists. If a channel or a container does not exist, the EXEC CICS PUT CONTAINER command creates the channel or container. After the channel and container have been found or created, the command adds any data that is supplied into the container. Because we need to create a container for our Borrower group item and a container for our Loan group item, we need to perform two EXEC CICS PUT CONTAINER calls. In Example 7, the first of these calls creates and adds data to the `BORROW-CONTAIN` container, and the second call creates and adds data to the `LOAN-CONTAIN` container. The changes are highlighted in bold.

Example 7 EXEC CICS PUT CONTAINER calls

```
Procedure Division.
    Root.
        move "John" to name of Borrower
        move 100 to creditScore of Borrower
        move 10000 to yearlyIncome of Borrower

        move 10000 to amount of Loan
        move 5 to yearlyInterestRate of Loan

        move 1000 to yearlyRepayment of Loan
        move "N" to approved of Loan
        move 0 to messageCount of Loan

        EXEC CICS PUT CONTAINER(LOAN-CONTAINER)
                                    CHANNEL(MINILOAN-CHANNEL)
                                    FROM(Loan) FLENGTH(Length of Loan)
                                    END-EXEC

        EXEC CICS PUT CONTAINER(BORROWER-CONTAINER)
                                    CHANNEL(MINILOAN-CHANNEL)
                                    FROM(Borrower) FLENGTH(Length of Borrower)
                                    END-EXEC
call "miniloan" using Loan Borrower

        if approved of Loan = "Y"
            display "Loan approved"
```

```
        else
            display "Loan not approved"
            move 1 to tableIndex
            perform until tableIndex > messageCount of Loan
                display messages of Loan (tableIndex)
                compute tableIndex = tableIndex + 1
            end-perform
        end-if
        goback.
```

Calling the generated program with the channel

Now that we have put the Loan and Borrower data into containers on the channel, we now need to modify the calling program so that it will pass the containers to the COBOL wrapper program that we created earlier. To do this, we replace the COBOL `call` statement with an EXEC CICS LINK PROGRAM statement. On this EXEC CICS statement, we specify the name of the channel that we want to pass to the wrapper program (refer to Example 8 where the changes are highlighted in bold).

Example 8 EXEC CICS LINK PROGRAM

```
Procedure Division.
    Root.
        move "John" to name of Borrower
        move 100 to creditScore of Borrower
        move 10000 to yearlyIncome of Borrower

        move 10000 to amount of Loan
        move 5 to yearlyInterestRate of Loan

        move 1000 to yearlyRepayment of Loan
        move "N" to approved of Loan
        move 0 to messageCount of Loan

        EXEC CICS PUT CONTAINER(LOAN-CONTAINER)
                                    CHANNEL(MINILOAN-CHANNEL)
                                    FROM(Loan) FLENGTH(Length of Loan)
                                    END-EXEC

        EXEC CICS PUT CONTAINER(BORROWER-CONTAINER)
                                    CHANNEL(MINILOAN-CHANNEL)
                                    FROM(Borrower) FLENGTH(Length of Borrower)
```

```
                                        END-EXEC
EXEC CICS LINK PROGRAM("CICSWRAP")
                                          CHANNEL(MINILOAN-CHANNEL)
                                          END-EXEC
if approved of Loan = "Y"
            display "Loan approved"
        else
            display "Loan not approved"
            move 1 to tableIndex
            perform until tableIndex > messageCount of Loan
                display messages of Loan (tableIndex)
                compute tableIndex = tableIndex + 1
            end-perform
        end-if
        goback.
```

Receiving the modified loan container

There is one final set of changes that needs to be made to the calling program. The program generated by Rules for COBOL adds messages into the *message* data item on the Loan group item. This information is passed back via the wrapper program to the calling program in `LOAN-CONTAINER`. The data in this container will be larger than the data originally passed to the Rules for COBOL program via the wrapper program. We will not be able to fit this larger amount of data into Loan variable in our calling program. If we to try, we get an error on the EXEC CICS GET CONTAINER command that we use to extract the data. In order to successfully get the larger amount of data into our calling program, we must use COBOL pointers, which are a standard CICS programming technique that used in these situations.

To use a COBOL pointer, we first declare the pointer in the Working Storage Section of the COBOL code. Alongside it, we also declare a variable to hold the length of the data inside the container. Next, we have to declare a copy of the Loan structure in the Linkage Section of the COBOL program. This declaration is required, because COBOL will only let us address variables that are declared in the Linkage Section. Because our original version of the Loan structure is declared inside the copybook in Working Storage, we cannot address it using a pointer. Example 9 shows these declarations, and the changes are highlighted in bold.

Example 9 Linkage section of COBOL

```
Working-Storage Section.
......
......
01    rl-pointer              pointer.
01    rl-length                pic 9(8) binary.

Linkage-Section.
01    received-Loan.
   05   rl-amount              pic 9(10).
   05   rl-yearlyInterestRate     pic 99.
   05   rl-yearlyRepayment       pic 9(10).
   05   rl-effectDate                  pic X(8).
   05   rl-approved                   pic X.
   05   rl-messageCount             pic 9(2).
   05   rl-messages                    pic X(60) occurs 99 times depending on
                     rl-messageCount.
Procedure Division.
    Root.
        ......
        ......
        ......
EXEC CICS LINK PROGRAM("CICSWRAP")
                                        CHANNEL(MINILOAN-CHANNEL)
                                        END-EXEC

        if approved of Loan = "Y"
            display "Loan approved"
        else
            display "Loan not approved"
            move 1 to tableIndex
            perform until tableIndex > messageCount of Loan
               display messages of Loan (tableIndex)
               compute tableIndex = tableIndex + 1
            end-perform
        end-if
        goback.
```

Next, we need to add an EXEC CICS GET CONTAINER command to set the rl-pointer to the storage inside the container. After this command completes, we must set the address of the received-Loan group item in the Linkage Section to the value of the rl-pointer to get access to the storage. After this step, we perform minor renaming to use the received-Loan group item instead of the Loan group

item in working storage. Refer to Example 10 on page 15 where the changes are highlighted in bold.

Example 10 EXEC CICS GET CONTAINER

```
......
......
Procedure Division.
    Root.

        ......
        ......
        ......
        EXEC CICS PUT CONTAINER(LOAN-CONTAINER)
                                    CHANNEL(MINILOAN-CHANNEL)
                                    FROM(Loan) FLENGTH(Length of Loan)
                                    END-EXEC

        EXEC CICS PUT CONTAINER(BORROWER-CONTAINER)
                                    CHANNEL(MINILOAN-CHANNEL)
                                    FROM(Borrower) FLENGTH(Length of Borrower)
                                    END-EXEC

        EXEC CICS LINK PROGRAM("CICSWRAP")
                                      CHANNEL(MINILOAN-CHANNEL)
                                      END-EXEC

        EXEC CICS GET CONTAINER(LOAN-CONTAINER)
                                    CHANNEL(MINILOAN-CHANNEL)
                                    SET(rl-pointer) FLENGTH(rl-length)
                                    END-EXEC
SET ADDRESS OF received-Loan TO rl-pointer

        if rl-approved of received-Loan = "Y"
           display "Loan approved"
        else
            display "Loan not approved"
            move 1 to tableIndex
           perform until tableIndex > rl-messageCount of received-Loan
display rl-messages of received-Loan (tableIndex)
                compute tableIndex = tableIndex + 1
            end-perform
        end-if
        goback.
```

Now, we have completed the necessary changes to the calling program.

Compiling the code

We have made all of the changes that are required to use channels and containers with our Rules for COBOL generated program; however, note that the previous examples do not include any error handling for abnormal responses from the EXEC CICS commands.

The final step in this example is to translate, compile, and link our calling program, wrapper program, and generated module. This step differs slightly from a normal COBOL compile, because all the EXEC CICS statements need to be translated so that the code can successfully compile.

You can obtain more information about translating and compiling COBOL for CICS (including sample JCL) in the *CICS Transaction Server for z/OS Application Programming Guide* at this Web site:

http://publib.boulder.ibm.com/infocenter/cicsts/v3r2/topic/com.ibm.cics.ts.applicationprogramming.doc/topics/overview.html

Conclusion

By using CICS channels and containers with Rules for COBOL generated code, you can reduce the amount of recompiling and relinking that occurs when business rules change. By using channels and containers, you can have a Rules for COBOL program that is located in a separate CICS region than the region in which the calling program resides. Therefore, you can use new architectures to improve availability and throughput.

By making a few simple changes, you can use the programs that are generated by Rules for COBOL in a much more flexible way inside CICS.

SupportPac

A forthcoming CICS SupportPac will provide a utility that will automatically generate a CICS Wrapper program for a given Rules for COBOL program and copybook. The SupportPac details are CA0A – CICS Channels and Containers Support Utility for WebSphere ILOG Rules for COBOL. The SupportPac is expected to be available later in 2009.

Author

Daniel Donnelly is an Advisory Software Engineer at IBM Hursley in the UK. He has worked for IBM for the past eight years. Dan is currently part of the WebSphere ILOG Synergies Team, with specific responsibility for identifying synergies between WebSphere ILOG products and z/OS. Prior to joining the WebSphere ILOG team, Dan worked for seven years in the CICS Development organization where, for much of his time, he led the development of the CICSPlex SM Web User Interface and the CICS Management Client Interface. He is a Chartered Engineer and a Chartered IT Professional.

Notices

This information was developed for products and services offered in the U.S.A.

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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

This document REDP-4589-00 was created or updated on September 10, 2009.

Send us your comments in one of the following ways:
- Use the online **Contact us** review IBM Redbooks publications form found at:
  ibm.com/redbooks
- Send your comments in an e-mail to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099, 2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks