Object REXX in OS/2

BY RICK BYRLEY

Since this is my first OS/2 column, please bear with me as I get my feet wet! I've been working with OS/2 for several years for SofTouch Systems, a major ISV of OS/2 software, and hope I can give you some insight into the world of OS/2.

The last few months my predecessor, Michael Norton, has discussed the Java Developer Kit provided with OS/2. I want to continue to explore the object-oriented world, since OS/2 is very much an object-oriented operating system. OS/2 provides the Object REXX language, which is an object-oriented version of the popular procedures language (now called Classic REXX). Although Java is certainly a hot programming language, for someone new to object-oriented programming, Object REXX may provide a better introduction. REXX is a very simple language to learn in comparison to C++ and Java, so it is much easier to focus on object-oriented programming techniques rather than getting hung up on syntax.

Unfortunately, it is easy to miss Object REXX; by default, Classic REXX is the REXX interpreter. Thus, Object REXX is not automatically enabled when you install OS/2, and the documentation is buried in the README file in the boot drive root directory. Moreover, the Object REXX INF file has no icon on the Desktop or elsewhere. (You can preview the online documentation before enabling Object REXX by entering the command VIEW OREXX.INF from a command line.)

[Object] REXX is a very simple language to learn in comparison to C++ and Java, so it is much easier to focus on object-oriented programming techniques rather than getting hung up on syntax.

ENABLING OBJECT REXX

As noted previously, Object REXX is not automatically enabled when you install OS/2. To enable Object REXX, use the SWITCHRX command. The SWITCHRX command detects the current version and switches to the other version. The system must be rebooted to complete the switch. Note that the REXX reference also changes when the SWITCHRX command is used; thus, when you switch to Object REXX, the Object REXX reference will appear when you select the REXX reference from the Warp Center menu. If you are used to accessing help from the command line, VIEW REXX.INF will always display the correct reference for the active interpreter. SWITCHRX renames the REXX.INF file to either OREXX.INF (the Object REXX reference) or CREXX.INF (the Classic REXX reference), then renames the reference associated with the new interpreter to REXX.INF.

Note: If you intend to use the Workplace Shell (WPS) features of REXX, you must issue the WPSINST command after switching to Object REXX. This command registers two new classes to provide WPS support.

DIFFERENCES BETWEEN OBJECT REXX AND CLASSIC REXX

The Object REXX interpreter is pickier about syntax than the Classic REXX interpreter. Although I've never encountered any problems personally, the documentation notes that some programs that run fine under Classic REXX will generate errors when run under Object REXX. The example provided in the documentation shows an extra END statement at the end of a program, but I found that this generated errors both under Object REXX and Classic REXX. Object REXX prescans the code, which means that errors that previously would have been encountered only when the code was executed are now caught before the program begins execution. This can be a problem if your programs have a lot of "dead code," that is, code that is present in the source but never actually executed.

Object REXX provides a facility for syntax checking. REXXC.EXE will report syntax errors without actually running the program. Simply enter REXXC source.cmd at the command line to use this facility. Some error codes have been changed between Classic REXX and Object REXX . This is a minor annoyance that is outweighed by the fact that Object REXX also provides extended error codes and information. For example, instead of simply reporting an extra END at the end of a program, Object REXX will include an additional line of error information:

```
REX0010E: Error 10 running
    G:\orexxtst.cmd line 6:
    Unexpected or unmatched END

REX0212E: Error 10.1:  END has no
    corresponding DO or SELECT
```

There are a few other differences documented in the "Migration" section of the Object REXX reference, but most of the time you will only get into trouble when you forget you are working with an

Figure 1: Simple "Hello World" Program That Illustrates the Basic Framework of an Object REXX Program

```
/*****************************************************************************
* PROGRAM: HELLOREXX.CMD
* AUTHOR:  Rick Byrley
* PURPOSE: Hello World program for Object REXX
*****************************************************************************/


/*****************************************************************************
* Mainline Logic
*****************************************************************************/

hw = .HelloWorld~new                    /* create new instance of object   */
hw~sayHello                             /* call method to say hello        */
exit                                    /* exit program                    */

/*****************************************************************************
* HelloWorld Class
*****************************************************************************/

::CLASS HelloWorld                      /* Class directive                 */

::METHOD sayHello                       /* Method directive                */
   say "Hello Object REXX!!!"           /* say Hello                       */
```

object-oriented language. For example, LINEIN is both a method and a function (to provide support for Classic REXX programs); using a mixture of both methods and functions can cause unpredictable results.

GETTING STARTED

Like other object-oriented languages, Object REXX uses classes to define objects and adheres to the principles of data abstraction, polymorphism, and inheritance. We'll be looking at these concepts and how they apply to Object REXX in forthcoming columns; for now I'm sure you're eager to get started. The sample code shown in Figure 1 is a simple "Hello World" program that illustrates the basic framework of an Object REXX program. Note that like C++ and Java (applications), the program utilizes a mainline routine that calls object methods to perform actions, although in Object REXX it is not necessary to label the routine "main." This routine creates instances and accesses object methods to perform its logic. In our sample program, the mainline routine creates an instance of the HelloWorld object, then calls that object's "sayHello" method to display a message on the console. Object instances are created in Object REXX by calling an object's "new" method, which is inherited from the Object superclass. The sample program creates an instance of the HelloWorld object and assigns it to the "hw" variable in the line of code:

```
hw = .HelloWorld~new
```

It is essential to create an instance of the program because, although methods look very much like functions, class methods are not identical to functions and cannot be called directly, which confuses many who are new to object-oriented programming. (This rule may seem contradictory, since the "new" method of the HelloWorld class is called above. For now, just note that the "new" method is an exception to the rule). The tilde (~) is the "message send" symbol that sends a message from the mainline code to the hw object to perform the action. The period (.) before HelloWorld indicates that HelloWorld is an environmental symbol (i.e., it references a class and is not a variable symbol).

The "hw" instance of the HelloWorld class is then called to perform its "sayHello" method:

```
hw~sayHello
```

The mainline program then exits. The remaining code defines the HelloWorld class and its methods. Classes and methods are defined in Object REXX by the ::CLASS and ::METHOD directives. Note that a class or method continues until the next ::CLASS or ::METHOD directive or the end of the file is encountered; there is no delimiter such as a terminating brace in C++. Thus, our sample program has one class with one method.

SUMMARY

That should be enough to get you started. Next month's column will examine in detail object-oriented concepts and structure. ts

Rick Byrley is senior workstation division technician for SofTouch Systems, Oklahoma City, OK., which provides both mainframe and PC software solutions. His primary focus is object-oriented programming. He can be reached at rbyrley@softouch.com.