5

Program Flow Using Free Format

In nearly every program you write, you need to control the flow of instructions. Free-format RPG IV offers experienced RPG programmers a new style for controlling program flow. In this chapter, we look at the numerous operations available for controlling flow in free format through conditional and selection logic, "do" functionality, and loop interrupts.

The If Group

Four free-format operations—`If`, `Else`, `Elseif`, and `Endif`—help you construct programs that execute conditional logic in a structured way. Together, these operations form a powerful arsenal for managing the flow of instructions in your program.

If

Since IBM modified the `If` operation for the extended Factor 2 format, this operation has become a favored method for coding conditional logic. Rather than write the following:

```
     A           IFGT         B
     C           ANDLT        D
```

it makes more sense to code

```
     If A > B and C < D;
```

To use the If operation in free format, just take the extended Factor 2 expression and assume you have the freedom to put it anywhere in positions 8–80. For every If, you must also code a corresponding Endif. The generic End operation (used in fixed format) is not available.

The If operation performs exactly like its extended Factor 2, fixed-format counterpart. The compiler checks factors in the comparison expression according to the operators given and the dictates of precedence (order of checking). The order of checking is as follows:

1. Complete checking is performed to true or false within parentheses.
2. When no parentheses are present, ANDs are checked before ORs.

If the premise expression of the If statement resolves to true, the operations that follow the statement are performed. The program continues with any number of operations until it reaches an Else, Elseif, or Endif statement.

If the premise expression resolves to false, program control passes to the Elseif premise, if present, or to an Else statement, if coded. If the If block includes neither an Elseif nor an Else, program control passes to the next operation after the Endif.

Listing 5-1 shows an example of If precedence.

```
/free
 // Consider the following If expression:

 If  (Acctbal > 1000 or Status = 'A')
                  and Date_1 > Date_2;

 // Due to the higher precedence of parentheses,
 // the logic group inside the parentheses will be
 // resolved first. If either comparison is true,
 // the group is true. If the group is true, the next
 // part of the expression is checked. If this
 // expression is also true, the If resolves to true.

 // Now consider the same expression without the
 // parentheses:

 If  Acctbal > 1000 or Status = 'A'
                 and Date_1 > Date_2;

 // This time, with no parentheses, the "and" has the
 // highest precedence, so the Status check is now
 // paired with the date comparison. If Acctbal is
 // greater than 1000, the entire If is true; if
 // false, both the Status check and the date
 // comparison must be true.
```

Listing 5-1: Precedence in If comparison expressions

Free-format's greatest advantage for `If` (and other program-control operations) is the option to indent subordinate program logic. Indenting gives your program a pseudo-formatted look, entirely the same as in other contemporary languages, and it makes the flow of the program's logic easier to discern. Listing 5-2 illustrates this method.

```
/free
 If Status = 'A' and Amount > 100;
   OK_Record = *On;
   Total_Amount += Amount;
 Else;
   Count_No += 1;
 Endif;
/end-free
```

Listing 5-2: If logic using indenting

Else

Most of the time, you won't need an Else operation, but Else (as well as Elseif) is available to handle the "false" leg of programming logic when you need to do so. If the premise of an If expression is false and you have coded an Else operation, program control passes to the first operation after the Else operation.

Just as with If, any number of operations can follow the Else. Indenting the operations specified after the Else gives future readers of your program a visual cue to the logic being used in the program.

Elseif

The Elseif operation is a relative newcomer to RPG IV. It replaces the combination of Else followed immediately by another If operation. By using If with Elseif, you form a control structure that lets only one group of operations be performed. If the group includes no Else, it is possible that no group will be performed. By coding an Else operation at the end of the group, you create an option for the condition "none of the above."

Endif

As I have noted, each If operation must conclude with an Endif. Free-format RPG IV provides no generic End operation like that in fixed format. The new approach makes your code "tighter" in the sense that only specified "end" operations work, reducing the chance of accidental error or misunderstanding by a future reader.

Listing 5-3 shows examples of If used with Elseif, Else, and Endif.

```
/free
 If Action = 'A';
   Write Record_a;
   Message = 'Record added';
 Elseif Action = 'C';
   Update Record_a;
   Message = 'Record updated';
 Elseif Action = 'D';
   Delete Record_a;
   Message = 'Record deleted';
 Else;
   Message = 'Invalid action code entered.';
 Endif;
/end-free
```

Listing 5-3: Using If, Elseif, Else, and Endif operations

The Do Operations

Free-format RPG IV provides two Do operations: Dow (Do while) and Dou (Do until). Although the fixed form of the Do operation isn't available in free format, the For operation (covered later in this chapter) contains all the functionality of Do.

Do While

The Dow operation uses a comparison expression, similar to the If operation. The expression is evaluated, and when it is found to be true, program control continues on the next line after the Dow operation. Any number of operations may follow the Dow operation.

The Dow operation has a looping control point at an Enddo operation. (Remember, no generic End operation exists in free format.) At the Enddo, program control is immediately returned to the Dow operation. The comparison expression is then checked again; if it is true, program control continues on the next line following the Dow.

As long as the Dow comparison expression resolves to true, the program continues in a loop. When the expression resolves to false, program control jumps to the first operation after the Enddo that is paired with the Dow operation, ending the loop.

With most Dow groups, a loop-control condition is set just prior to the group and near the end of the group, just before the Enddo operation. Listing 5-4 shows an example of a file read loop that uses Dow.

```
      /free
       Read File_A;                // Initial Read
       Dow not %eof(File_A);       // Test here

         // Process record here when Dow test is true

         Read File_A;              // Subsequent Reads
       Enddo;                      // Go back to the Dow
       // Control comes here when the test at Dow is false
```

Listing 5-4: Using Do while (Dow) to read a file until end-of-file

Do Until

Like the If and Dow operations, the Dou operation uses a comparison expression. Dou sets up a future check but otherwise does nothing. Program control flows immediately to the next operation after the Dou. The controlling condition of the loop is usually set soon after the Dou, and an If test is placed afterward to determine whether to continue in the loop. You normally place the Endif for this If just before the Enddo. If the If resolves to true, the program continues after the If, keeps on going after the Endif, and finally comes to the Enddo. At this control point, the Dou condition is tested. If the expression resolves to true, program control resumes at the next operation after the Enddo. If the expression resolves to false, program control "jumps" back (loops back) to the Dou operation.

Listing 5-5 shows an example of a file read loop that uses Dou.

```
/free
 Dou %eof(File_A);          // Set up for test only
   Read File_A;             // Get record here
   If not %eof(File_A);     // Check for end-of-file

     // If not end-of-file, process record here

   Endif;
 Enddo;        // Test here. If false, go back to Dou
 // Control comes here when the test at Enddo is true
/end-free
```

Listing 5-5: Using do until (Dou) to read a file until end-of-file

Dow and Dou Differences

The difference between Dow and Dou—and it's a big one—lies in when the comparison expression is checked, as well as in the program-control action. Program-controlled looping is common, and programmers nearly always adopt one of these operations as their preferred method. In most situations, either approach will provide a satisfactory solution.

If you want to set the controlling condition just once, a do until is the correct form to use. A do while may be the better choice if prior programming statements have already set the controlling condition. Which do loop you choose will depend on many factors, but the biggest factor is probably personal preference.

For

A For operation and its termination control point, Endfor, define a controlled-loop group of operations. The For group uses an initial specified index value, an increment index value (or default), and a termination value. You can specify the For group indexing to either increment or decrement the current index before checking to see whether the result meets the termination condition.

A For group uses an index variable, as in the following example:

```
        For     j  = 1 to 10;
```

You must define the indexing variable in definition specifications as a numeric field large enough to handle the largest index value. This example specifies no increment, so 1 is used as the default. The loop termination value in the example is 10. If the value of index j is 10 or less, program control will continue to the next operation after the For. At the Endfor, control returns to the For operation, where the index is incremented (or decremented) and then compared with the termination value. If the index is greater than the termination value (if incrementing), program control jumps to the operation immediately after the Endfor operation. If the index is not greater than the termination value, control continues at the next operation after the For operation. The index used in the For group can be used within the group and changed if desired.

A For group can also start with an index value higher than the termination value and decrement until the current index is less than the termination value:

```
        For j = 100 downto 1 by 2;
```

In this example, the index j has an initial value of 100 and a termination value of 1. The index in this case is 2. The first time through the For group, the value of j is 100. The next time through, j equals 98, then 96, and so on until j equals 2. When j equals 2, the For group is again performed. Control then returns to the For statement, where j is decremented by 2, yielding a j value of 0 (zero). Because j's value is now less than the termination value (1), control passes to the next operation after the Endfor statement.

Listing 5-6 shows some additional examples of For.

```
      /free

       //    Indexing low to high:
       // The following is a traditional bubble sort of
       // an array. Variable n is the highest element to
       // be included in the sort, and i and j are indexes
       // of two For groups.

       For i = 1 to n;
         For j = 1 to n-1;
           If Array(j+1) < Array(j);
             SaveElem = Array(j);
             Array(j) = Array(j+1);
             Array(j+1) = SaveElem;
           Endif;
         Endfor;
       Endfor;

       //    Indexing high to low:
       // The following routine finds the position of the
       // last s in a phrase. Phrase is defined alpha-40
       // "She sells sea shells by the seashore.   "

       For index = %len(phrase) downto 1;
         If %subst(phrase:index:1) = 's';
           Leave; // Loop interrupt, described below
         Endif;
       Endfor;
       // The variable index will be 32 at this point.

      /End-free
```

Listing 5-6: Using the For operation for controlled looping

Loop Interrupt

The loop operations Dou, Dow, and For normally end when the loop's index termination requirement is met. There are times in programming when you need to escape from a loop—either all the way out of it or just to its next iteration. RPG IV's loop interrupt operations Leave and Iter perform these functions for us. (Remember, free-format RPG IV provides no Goto operation.)

Leave

The Leave operation causes program control to jump to the next operation after the current Dou, Dow, or For group. The effect is equivalent to a Goto, but it occurs in a structured way. You may be thinking that this operation's purpose is primarily error handling. Not so. Let's say you are using a For group to load 10 records from a database into a subfile. However, after five successful record reads, you come to end-of-file. To exit the subfile load routine, you can just set up an "If end-of-file" condition and leave immediately after the Read operation.

The code in Listing 5-6 (above) uses Leave after locating a correct value in a string. Listing 5-7 shows a sample subfile load routine that uses Leave.

```
/free
 // Load a subfile with the next 10 data records
 For I = 1 to 10;
   Read data_file;  // Get next record or eof
   If %eof(data_file);  // If eof
     Leave;         // Jump out of For loop
   Endif;

   // Load subfile from data record here

 Endfor;
 // Leave instruction sends control here

/end-free
```

Listing 5-7: Using Leave in a load routine to exit at end-of-file

Iter

The Iter operation is the other loop interrupt. Specifying Iter causes the program to jump to the current Enddo or Endfor operation (depending on which loop we are in), at which point the normal function of the Enddo or Endfor is performed. As with Leave, the effect is equivalent to a Goto, but, again, the action takes place in a structured way.

A good example of using Iter is what I call "one-at-a-time" error reporting on a data entry panel. If a panel allows entry of 12 different fields, any of which could be entered with invalid data, either we check them one at a time and loop back with one error message or we find all the errors and use a message subfile. To use the first method, just check each field. If the first field is okay, check the second, and so on. If an error occurs at any point, set up for the correct error message, and use Iter to skip all further error checking. Listing 5-8 illustrates this scenario.

```
/free
 Dou Exit;
   Exfmt ScreenRec;   // Panel displayed
   If not exit;       // not exit
     If Field_1 <> [valid value]; // Field 1 check
       Message_F1 = *On;
       Iter;
     Endif;
     If Field_2 <> [valid value]; // Field 2 check
       Message_F2 = *On;
       Iter;
     Endif;

     // Continue with other 10 fields to be checked

   Endif;
 Enddo;
/end-free
```

Listing 5-8: Using Iter to skip to the next iteration

The Select Group

The Select operation, with its corresponding operations When, Other, and Endsl, creates a procedural structure very similar to If, Elseif, Else, and Endif.

Select

You code the Select operation on a line by itself, and it starts the select group. Following the Select, you can specify a When operation with a comparison expression. If the expression resolves to false, control is passed to the next When with its comparison expression. The false jumps continue until either an Other statement or the Endsl is reached.

The Other operation is optional. If all When expressions yield false, no action is taken within the Select group unless you have specified an Other operation. Other means "if none of the above" is true, perform the operations between the Other operation and the Endsl. If any of the When expressions is true, the operations specified between the When that is true and the next When (or Other) are performed, and control then jumps to the Endsl operation. In a Select group with no Other operation, either one set of operations is performed or no operations are performed. If the Select group has an Other operation at the end of the group, at least one set of operations will be performed.

Listing 5-9 shows an example of Select, When, Other, and Endsl.

```
 /free
  Exfmt Screen;
  Select;
    When exit;            // User presses F3 - exit
    When update_req;      // User presses F8 for update
      Exsr Verify_data;   // Verify data first
      If no_error;        // Data okay?
        Exsr Update_Rec;  // Go ahead and update
      Endif;
    When cancel;          // User presses F12 - previous
      Message = 'Function canceled by user.';
    Other;                // Enter key pressed
      Exsr Verify_data;   // Verify data
  Endsl;
  // Control comes here after one of When's or Other.
```

Listing 5-9: Using Select, When, Other, and Endsl operations

Operations Absent in Free Format

Many RPG programmers are acquainted with the fixed-format Cas (Case) operation and its two-letter suffixes EQ, NE, GT, GE, LT, and LE. These operations are not available in free format, but you can easily replace them by using Select and When operations with Exsr (Perform a subroutine) operations to call the subroutines.

As you have learned, free format also lacks a Goto operation. Doing without a Goto isn't such a bad thing. Programming style texts and magazine articles have argued nonstop for decades about the good and bad points of using a Goto operation. Rather than anguish over a loss of Goto freedom, let's look on the bright side: No Goto means no spaghetti code! Many of us have had to sort out programs written by programmers who used Gotos—here, there, and everywhere. It was nothing short of a miracle that the code even worked. Maintaining this kind of code is a programmer's nightmare.

In earlier versions of RPG, the use of Goto was pretty common. Now, Leave, Iter, and even LeaveSr (Leave subroutine) let us perform Goto-like functions in a clear and orderly way. Armed with these structured operations, we don't need a "real" Goto, just an understanding of these loop interrupters. The lack of a Goto