4

Introducing Free-Format RPG IV

The ability to create free-format calculations in RPG IV became available with the arrival of Version 5 Release 1 of WebSphere® Development Studio, IBM's application development tool suite for the IBM i operating system. Free format's arrival caused little fanfare back then, and most RPG IV developers with whom I spoke at the time weren't very interested in the concept.

Early on, some of my RPG IV programmer friends tried to use free format and asked me for help in their work. The ambitious ones who gave it a try found themselves pleased with the results. However, due to limitations in the first release of free format, RPG calculations still required many lines of fixed-format code, such as for `Klist` and `Kfld` operations. Mixing fixed- and free-format calculations made programs look "clunky" and certainly didn't help convince co-workers to convert to the new style.

In V5R2, IBM provided additional functionality that let RPG IV programmers use key arguments within a free-format `Chain` or `Set`*xx* operation or use the `%Kds` built-in function with a data structure. These enhancements represented a big step forward in "de-clunking" the calculations. Other new functions, such as the += accumulative assignment operator (which adds the result of an expression to the target of the assignment), gave free-format RPG procedures a C- and Java-like appearance. Additional built-in functions at that time also helped to modernize RPG IV's procedures.

Today, free-format calculations in RPG IV can look like any other modern programming language. Newcomers to RPG IV programming who have prior experience in C, Cobol, PL/1, or another free-format language find free-format RPG IV simple to learn and use. Now that free-format calculations have gained a modicum of acceptance, many fixed-format RPG IV "old-timers" are taking a good look at the free-format style.

For those new to free format, getting started is sometimes the toughest part. Frankly, it's not easy to change the way you've done something for a long time, especially programming. If you have come this far and want to try writing a little program in the new format, this book will help you begin. We start our tour of free-format calculations in this chapter, with an overview of the free-format structure and a look at some key operations and features.

Coding Free-Format Calculations

The first step in writing free-format calculations is remembering to place your free-format RPG IV code in position 8 or beyond. Positions 1–5 are available for anything (e.g., change control information). Position 6 must be remain blank, and position 7 is reserved for compiler directives, such as `/SQL`. Free-format calculations end in position 80.

All supported free-format operations, as well as all built-in functions, are available to you. In contrast to its fixed-format older brother, which has 153 operations, free-format RPG IV provides just 62 free-format operations (as of V7.1). However, free-format RPG IV isn't simply a "stripped-down" subset of its fixed-format counterpart. IBM has created many built-in functions to provide functionality equivalent to (or better than) most of the "missing" operation codes. As of V7.1, RPG IV provides a total of 80 built-in functions.

A line of free-format source begins with a free-format operation code, followed by one or more spaces, the Factor 1 operand, one or more additional spaces, and then the Factor 2 operand. Free-format RPG IV has no result field operand. Instead, you perform arithmetic and character-management operations using assignment statements (`Eval` operations without the `Eval`).

An implication of no result field is the inability to define work variables "on the fly" as RPG programmers commonly did years ago. However, thanks to the powerful nature of free-format expressions, we don't need as many work fields today, and declaring them in definition specifications makes program maintenance easier and more productive.

The final requirement for a line of free-format RPG IV source is a terminating semicolon (;), followed by a comment if desired. You enter a comment by keying two slashes (//) followed by the text of the comment. You can also place comments on a line by themselves.

Listing 4-1 shows a sample block of free-format code, including a few free-format operations and some comments.

```
Miles_per_gallon = Miles / Gallons;
Eval(h) Pay = Hourly_rate * Hours;
      //  An entire line comment
Name = %trim(First_n) + ' ' + %trim(Last_n);
Error_cust_no = *On;    // Short comment on a calculation line
```

Listing 4-1: Sample free-format block

Naming Variables

Free-format RPG IV's rules for naming variables are no different from fixed format's, but when employing longer names (more than 14 characters), you must use either the extended Factor 2 format or free format in your calculations. Variable names must begin with a character and can be in any case. The character can be any one of the 26 regular alphabet characters or the special character #, $, @, or _. Numbers (0–9) are optional and can be used after the first character. Variable names cannot contain blanks, but you can use the underscore character (_) as a word separator to form a multiword name (e.g., Miles_per_gallon).

Until RPG IV came along in late 1994, RPG variable names were limited to six characters. This limitation included references to arrays and their indexes. The first version of RPG IV supported 10-character names, matching the size maximum in DDS for variable naming. A few years ago, IBM extended the variable-name length to its present limit of 4,096! Not many programmers are interested in using such long names, but it's sure nice not to be constrained either.

Free-format RPG has a "semi" restriction for variable naming. In free format, the Eval operation code may be dropped if no op-code extenders are needed. However, if a variable name uses the same spelling as an operation code—such as In, Out, Select, and others—you must specify the Eval operation code when the variable is used on the left side of the assignment. I suggest not using variables that are named the same as operation codes, to eliminate confusion and restrictions.

Programming Style

No other rules apply when entering source statements in free-format calculations. However, good programming style should prompt us to enter statements in an ordered way that makes a program's logic easier to understand during program maintenance. Good style would dictate entering a program's "outer" logic as far left as possible (position 8) and beginning "inner" logic two spaces to the right. Continue this indenting process until you're about halfway across the page. If you need deeper groups, you will have to decide whether to continue to the right or to start over again at position 8.

Listing 4-2 shows a free-format code block that uses indenting to make the program logic more apparent.

```
       Dou %eof;
         ReadC SubfileRec;
         If not %eof;
           Fielda = Fieldb;
           If Fieldc <> *zero;
             Error_Msg_1 = *On;
             RI_Fieldc = *On;
             Sflnxtchg = *On;
           Endif;
           Update SubfileRec;
         Endif;
         ReadC SubfileRec;
       Enddo;
```

Listing 4-2: Example of indenting free-format calculations

A Note About Case

RPG IV has no rules regarding the case of variables, operation codes, and comments in source statements, but some programmers suggest using a style that capitalizes each "word" in variables and uses lower case otherwise (for example, `SubfileRec`). Others recommend fully capitalizing all externally defined variables. The compiler translates all variables and operation codes (other than character strings within apostrophes) to upper case before analyzing the code, so whatever case options you choose are purely a personal decision.

Free-Format Operation Codes

Table 4-1 lists the 62 operations that free-format RPG IV supports as of V7.1. Appendix A describes each of these operations in detail. All built-in functions are also available to you in free format. Many built-in functions, such as %Check, %Lookup, and %Scan, provide operation code functionality. Some, such as %Check, provide exactly the same function as an operation code, while others, such as %Lookup, provide additional capabilities.

Table 4-1: Free-format operations

Operation	Description
Acq	Acquire a program device (used in ICF files).
Begsr	Begin a subroutine.
CallP	Call a prototyped procedure (you can also call procedures implicitly, omitting the CallP).
Chain	Access a record from a file directly by key or relative record number.
Clear	Set all items in a data structure, record format, array, or variable to zero or blank, depending on the data type.
Close	Close a file that has been opened using the Open operation.
Commit	Commit file changes made since the last Commit or Rolbk operation.
Dealloc	Deallocate dynamic storage.
Delete	Delete a record from a file.
Dou	Do until (a logical group ending with Enddo).
Dow	Do while (a logical group ending with Enddo).
Dsply	Display a message.
Dump	Dump a program (variables, record contents, and so on).
Else	Else (part of an If group).
Elseif	Else and If combined (part of an If group).
Endxx	End a group of logic or monitor operations; the suffix xx must match the starting operation (i.e., Enddo, Endfor, Endif, EndMon, or Endsl).
Endsr	End of subroutine.
Eval	Evaluate an assignment expression; if a character string is specified, the result is left-justified (assignment is possible without explicitly specifying the Eval operation code).
EvalR	Evaluate an assignment expression; character strings are right-justified.
Eval-Corr	Assign corresponding subfields in data structures. *Continued*

Table 4-1: Free-format operations (continued)	
Except	Perform exception output (program-described on output specifications).
Exfmt	Write and then read a record format (often called “Execute a format”).
Exsr	Perform a subroutine (often called “Execute a subroutine”).
Feod	Force end of data.
For	Begin a logic group, ending in Endfor, that uses a specified index and counts up or down to a specified limit.
Force	Force a specified file to be read next (used only if the cycle is desired).
If	Begin a logic group, ending in Endif, that may have Else or Elseif inside the logic group.
In	Retrieve data from a data area and load the named data structure.
Iter	Iterate, or jump to the most current Enddo or Endfor.
Leave	Leave, or jump out of the most current Do or For group to the next statement after the Enddo or EndFor.
LeaveSR	Leave a subroutine.
Monitor	Begin a monitor group, ending with an EndMon, to handle error situations in a section of code.
Next	Force the next input to come from the specified program device.
On-Error	Used in a Monitor group to specify an error condition and begin handling if true.
Open	Open a file that was specified as user open on the file declaration.
Other	Used in a Select/When group to handle the condition “none of the above.”
Out	Write the contents of the named data structure to a data area.
Post	Update the file information data structure for the named program device or file.
Read	Read next (forward in the file); the operand can be a record name or file name.
ReadC	Read a changed record (used only for subfiles within a display file).
ReadE	Read equal—A multifunction operation that compares a specified operand with the current file index, and if the operand matches the current key, reads a record. If they are not equal, eof is set.
ReadP	Read prior (backward in the file); the operand can be a record name or file name.
ReadPE	Read prior equal (the same as ReadE, but reading backward).
Rel	Release a program device.
Reset	Reset a data item to its initialized value.

Return	Return—Used in two contexts: at the end of a subprocedure with an optional expression or elsewhere to return to the caller of the procedure.
RolBk	Roll back—Used with commitment control to remove file changes made since the last Commit or RolBk operation.
Select	Begin a logic group requiring When statements and ending with Endsl.
Setgt	Set file pointer greater than (a parameter is used to set the file pointer to the record whose key value is closest to but greater than the parameter).
Setll	Set file pointer greater than or equal (used to be called “lower limit,” thus the LL. The file pointer is set to the record whose key is equal to or greater than the specified parameter).
SortA	Sort an array (the array to sort is specified as a parameter; the order is specified on the array definition).
Test	Test a date, time, or timestamp for validity, or test a character or numeric field for a valid date or time.
Unlock	Unlock a data area object, or release a record lock.
Update	Update a record previously read via a Read or Chain operation.
When	Part of a Select group used to specify a condition to test, similar to If.
Write	Write a new record to a file.
XML-Into	Parse an XML document into a variable.
XML-Sax	SAX Parse for an XML document.

Perhaps more interesting than which operations are present in free format is which fixed-format operations are not included in the set of free-format operations. Some frequently used fixed-format operations didn’t make the cut. Table 4-2 lists a sampling.

Table 4-2: Examples of fixed-format–only operation codes	
Add	Move
Call (Dynamic call)	Movea
CallB (Bound call)	Mult
CASxx (Case)	Mvr
Cat	Scan
Div	Setoff
Do	Seton
End	Sub
Goto	Subst
Lookup	Tag

Most of these operation codes have a free-format counterpart to which you can easily convert. The Seton and Setoff operations, for example, have been replaced by the Eval operation (as in Eval *In21= *On). Another example, Mvr, is easily converted to the %Rem built-in function. And you can replace a Lookup operation with one of five %Lookup or five %Tlookup built-in functions, depending on whether you are searching an array or a table.

It would be nice if all the unsupported operations from fixed format had a simple equivalent in free format, but this is not the case. Some fixed-format operations, such as Move, have no clean and easy conversion path to free format. In Chapter 11, I provide solutions for some of the tough ones (of which, fortunately, there are few). Perhaps one day IBM will provide additional built-in functions or other methods to address the more difficult conversion situations.

In the rest of this section, I describe some of free-format RPG IV's more commonly used operations (as well as built-in functions where needed) and show some examples to illustrate their use. If you are already writing calculations using the extended Factor 2 format rather than RPG's original fixed format, you will find much of this material familiar.

Evaluate

The most common operation in free-format RPG IV doesn't even use an operation code. It's the evaluate (Eval) operation, minus the Eval operation code. This form of evaluate is usually called an *assignment statement*. The evaluate operation evaluates the expression specified to the right of the equal sign (=) and assigns the result to the receiving item on the left side of the equal sign. The receiving side is cleared for the default length of the item or for the specified length if substringing is used. This is the way other free-format languages work, too. In fact, CL's CHGVAR (Change Variable) command functions this way for its VAR parameter.

When performing mathematical calculations, you sometimes want the result rounded (half adjusted). The assignment statement doesn't perform this rounding, but you can accomplish it by entering the Eval operation code to the left of the assignment statement and specifying the half-adjust operation extender, h. You specify other operation extenders, such as precision (r), in the same way.

Listing 4-3 shows several examples of assignment statements.

```
Dcl-S Field30 Char(30);
Dcl-S Field10 Char(10) Inz('ABCDEFGHIJ');
Dcl-S Field3 Char(3);

Field30 = Field10;     // Field30 is cleared, and then Field10 is
                       // moved to it, left-justified

Eval(h) Pay = Hourly_rate * hours;  // The math is performed (with
                       // rounding) and the result assigned to Pay


               // More complex forms

*In03 = F7 = F3;       // This statement checks to see whether F7
                       // is equal to F3. If yes, *In03 is set to
                       // *On, but if F7 is not equal to F3, *In03
                       // is set to *Off

*In21 = *In43 or (Pay > 100);  // This statement sets *In21 to *On
                       // if either *In43 is *On or Pay > 100

%subst(Field30:5:4) = 'xyz';  // Positions 5-8 of Field30 are
                       // cleared, then 'xyz' is moved to 5-7

Field3 = *In03 + (Pay > 100) + %eof(FileA);  // Three items are
                       // evaluated as true or false and then
                       // concatenated, with the result of Field3
                       // equal to '000', '001', '010', and so on
```

Listing 4-3: Sample assignment statements

If

Another popular operation code, supported both in fixed and free format, is the If operation, along with its associated operations Else, Elseif, and Endif. The free-format version of If is similar to the extended Factor 2 method, but it gives you the added freedom of being able to place the specifications anywhere within the line (or lines). Parentheses, in addition to the logical operators And and Or, provide a significant improvement in programming ease over the previous If*xx* operation.

Many people are not aware that RPG had no If at all until RPG III (circa 1979). IBM retrofitted the operation into RPG II around 1990. In case you're wondering how programmers performed "If" logic in RPG before If was invented, the operation used (and still supported today) was Comp (Compare), along with lots of indicators.

The Elseif operation is a relative newcomer to RPG IV, new with V5R1. Its use can simplify a long set of nested Ifs and Elses. Elseif is equivalent to an Else operation followed immediately by an If. You need only one Endif at the end of an If/Elseif group, regardless of how many Elseifs the group contains. One set of statements may be performed in the If and Elseif blocks of code, or possibly none. This functionality follows the same scheme as the Select and When operations (you'll learn more about those two operations in Chapter 6).

Interestingly, you can't shorten the Endif operation to End in free format. In fact, free format doesn't support the End operation at all. Correct "Ends" force us as programmers to put the appropriate "End*xx*" at the end of a set of logic. It's a small thing, but program clarity is improved.

Listing 4-4 illustrates the use of If, Else, Elseif, and Endif operations in free format.

```
If  A = B;                // Simple If, Else, and Endif
  FieldA = FieldB;
Else;
  FieldA = *Zero;
Endif;

       // An If, Elseif, Else, and Endif group
If Option = 'A' and (Type = 3 or Company = 73);
  Value = 1;
Elseif Option = 'B' and (Type = 3 or Company = 75);
  Value = 2;
Elseif Option = 'C' and (Type = 1 or Company = 99);
  Value = 3;
Else;
  Value = 4;
Endif;
```

Listing 4-4: Examples of If, Elseif, Else, and Endif