Chapter 3

EXPRESSIONS

Support for expressions in RPG IV extends normal conditional logic, keyword support, and calculations through the use of natural expressions. Natural expressions are used to simplify the programming of these components as well as the implementation of traditional business rules. Expressions can be embedded as a parameter of a procedure. Any expression matching the parameter's type can be specified, provide that the parameter is a CONST parameter. Unlike prior versions of RPG, the RETURN operation accepts expressions in the extended factor 2.

EXPRESSIONS in RPG

Expressions are used in the following areas of an RPG program:

- **Declaration.** Most definition specification keywords support expressions. This support provides referential referencing of related data items defined in the program and simplified initialization of data items.

- **Assignment.** The calculation specification EVAL and EVALR operations fully support expressions of every type. Expressions can appear, in the assignment statements of the EVAL and EVALR operations, on either side of the = (equals) sign.

- **Comparison.** The conditional operations of the calculation specification include IF, WHEN, DOW, DOU, and FOR. These operations fully support conditional expressions. Note that assignment is never performed by conditional operations.

- **Procedures.** The CALLP and RETURN operations support expressions or parameters specified in factor 2.

NATURAL EXPRESSIONS

Expressions in RPG are specified in traditional mathematical infix notation. This kind of expression syntax is sometimes referred to as natural expressions. Natural expressions allow basic mathematics to be written with RPG in a form similar to traditional mathematics. Bertrand Russell, the British philosopher and mathematician, once said, "Mathematics may be defined as the subject in which we never know what we are talking about, nor whether what we are saying is true."1

In previous versions of RPG, the programmer always knew the outcome of an expression because only fixed-format, single-operator expressions were permitted. With natural expression support in RPG IV, programmers can write as complex or simple an algorithm as needed. As for complex algorithms, to paraphrase Bertrand, the programmer that follows the program author might never know what is expressed or if it is true. So write as basic an expression as possible.

Expressions are made up of *operands* and *operators*. Operands typically are data such as fields or numeric literals. Operators are mathematical symbols such as + (addition), - (subtraction), * (multiplication), or / (division); and conjuncts such as AND, OR, and NOT. The operators supported in RPG natural expressions are those listed in Table 3.1.

Table 3.1: Expression Operators

Operator	Type	Description
+	Unary	Indicates positive numerical value
-	Unary	Indicates negative numerical value
NOT	Unary	Opposite of evaluated result
+	Binary (alpha)	Concatenation of two character strings
+	Binary (numeric)	Addition
-	Binary (numeric)	Subtraction
*	Binary	Multiplication
/	Binary	Division
**	Binary	Exponentiation (powers and roots)
=	Comparison	Equal
=	Assignment	Set values equal
>=	Comparison	Greater than or equal
>	Comparison	Greater than
<=	Comparison	Less than or equal
<	Comparison	Less than
<>	Comparison	Not equal
AND	Logical	AND conjunction comparison
OR	Logical	OR conjunction comparison

Priority of Operators

Expressions are parsed and then evaluated in a defined order. To ensure that the equation always results in the same value, a precedence of the operations is applied. Fortunately, this precedence is the same as that used by most other programming languages as well as mathematics. Table 3.2 lists the priority of the operators used in expressions in RPG.

The priority of parentheses and built-in functions is interpreted as meaning that the operations inside the parentheses are performed independently of the operations outside the parentheses. Use parentheses for clarity or when the priority of the equation is uncertain. Parentheses can be used to override the priority. Hence, parentheses can force an addition operation to be performed before a multiplication operation.

Table 3.2: Order of Evaluation of Operators

Priority	Operator	Description
1	( )	Parentheses
2	Functions	Built-in functions or procedure functions
3	Unary operators	Unary +, -
3	Logical not	NOT
4	**	Powers and roots
5	* /	Multiplication and division
6	Binary operators	Binary +, -,
7	Comparison	Comparison operators, =, >=, >, <=, <,
8	Logical operators	Logical and, logical or
9	=	Assignment

The order of evaluation of an expression is important and is established with traditional mathematical precedence rules.

The power function ** is used for exponentiation. Mathematical rules state that, if a value is raised to the power of *n*, it is multiplied by itself *n* times. For example, 2**3 would result in 2*2*2, which evaluates to 8.

Mathematical rules also state that, if a value is raised to the power of 1/*n* (i.e., "one over *n*"), the result is the *n*th root of the value. For example, 16**(1/4) evaluates to the 4th root of 16, or 2. Table 3.3 lists a few example equations that use powers and roots.

Table 3.3: Powers and Roots Syntax

Equation	Description
4**2	Result is 4 squared or 4x4 (16).
4**.5	Result is the square root of 4 (2).
27**(1/3)	Finds the cube root of 27 (3).
16**(1/2)	Finds the square root of 16 (4).

Expressions can be categorized into three types, *Boolean*, *numeric*, and *string*. An expression is any list of tokens that represents a value, either a character string or numeric value, or an operator (see Table 3.1). In other words, any character string or numeric value, be it a simple number or complex math formula, is an expression.

RPG support for natural expressions is similar to that of CL, BASIC, COBOL, C++, and PL/I. Expressions can be used in the extended factor 2 of the alternate calculation specification, and in the keyword section of the definition specification. Expressions can be used in assignment statements, compare statements, or in declarations.

Expressions, known as *tokens,* are made up of literal values, numbers, fields, and symbols. Symbols are used to perform operations on the various values. An expression can be as simple as the number 12 or as complex as the equation: (4*PI)*R**2.

Expression Continuation

To continue an expression, place the next token of the expression on the next line in the extended factor 2 or the function/keyword area of the definition specification.

To continue a quoted character string, use either a + (plus) or - (minus) sign to continue the string. This directs the compiler to concatenate the value together either on the first nonblank position when using the + sign or the first position of the extended factor 2 or function/keyword area when using the - sign.

Expressions in Assignment Statements

When one value is assigned to another, the EVAL and EVALR operation codes are used. These operations copy the value on the right side of the equal sign, known as the *r-value*, to the variable on the left of the equal sign, known as the *l-value*. The assignment must

result in an r-value and an l-value that match. In other words, if the l-value is character, then the r-value must result in a character value.

The EVAL and EVALR operations work with all types of expressions. When used with character expressions, EVAL copies the value left justified. EVALR copies the value right justified.

The l-value can be an expression, but only when it is a character variable or an array name with an index value. The %SUBST() built-in function can be used as the l-value. In addition, numeric expressions can be used on the starting position and the length of the %SUBST() function. When an array index is specified, the index may be a literal, a field, or an expression.

Figure 3.1 contains a sample of several expressions used in assignment statements.

```
.....CSRn01Factor1+++++++OpCode(ex)Factor2+++++++++++++++++++++++++++++++++++++++
0001 C                   EVAL       A = B + C

0002 C                   EVAL       Amt_Due = Amt_Due - Amt_Paid

0003 C                   EVAL       PI = 3.1415926

0004 C                   EVAL       Area = 4 * PI * Radius ** 2

0005 C                   EVAL       Message = 'RPGIV is cool!'

0006 C                   EVAL       %subst(comp_name : start+3 : start+6) = 'Q38'

0007 C                   EVAL       ptrData = %ADDR(comp_name)

0008 C                   EVAL       Presidents(I + 1) = 'George '
0009 C                                + 'Washington'

0010 C                   EVAL       *INLR = (%EOF(CUSTMAST) or %EOF(ITEMMAST))
0011 C                   EVAL       *IN32 = Amount > 100 and Price - Cost < 25.50
```

Figure 3.1: Examples of assignment expressions.

In Figure 3.1, line 1 computes the sum of B plus C and stores the result in A. In line 2, the AMT_DUE field is reduced by AMT_PAID. In line 3, the field named PI receives the value of 3.1415926. In line 4, the area of a sphere is computed as 4*PI*R2.

Line 5 copies a character string expression to the field named MESSAGE. Line 6 copies the character string expression 'Q38' to a substring location of the COMP_NAME field.

Line 7 retrieves the address of the field COMP_NAME and copies it into the pointer variable named PTRDATA. Line 8 concatenates the string 'George ' 'Washington' into 'George Washington' and copies it into an array element of the PRESIDENTS array. The element index is calculated from the expression I+1.

Line 10 is a Boolean expression. Indicator LR is set on if the end-of-file condition exists for either the CUSTMAST or ITEMMAST file. Line 11 sets on indicator 32 if the AMOUNT field is greater than 100 and the difference between PRICE and COST is less than 25.50.

Expressions in Compare Statements

The RPG operations DOW, DOU, IF, FOR, and WHEN allow the use of expressions as compare statements. Expressions used with these operations do not assign their result to a field. Rather, the values are used to compare one value to another.

The DOW, DOU, IF, FOR, and WHEN operations have identical support for expressions. Figure 3.2 illustrates various conditional expressions.

```
.....CSRn01Factor1+++++++OpCode(ex)Factor2+++++++Result++++++++Len++DcHiLoEq....
.....CSRn01.............OpCode(ex)Extended-factor2++++++++++++++++++++++++++++++
0001 C                   if         A = B

0002 C                   if         Amt_Due > 10000 and DaysOvrDue >= 30

0003 C                   if         (Price - Cost) / Price < 10 or
0004 C                               Cost = 0
0005 C                   select
0006 C                   when       *IN01 = *ON

0007 C                   DOU        *INLR
0008 C                   read       Customer                              LR
0009 C                   endDo
0010 C                   endSL
```

Figure 3.2: Expressions on conditional statements.

In Figure 3.2, line 1 is a basic comparison expression. When the fields A and B are equal, the condition is considered TRUE. When they are not equal, the condition is FALSE.

Line 2 compares the AMT_DUE field to 10000. If AMT_DUE is greater than 10000, it then evaluates the next condition. If DAYSOVRDUE is greater than or equal to 30, the entire conditional expression is considered true. If either the first condition or second condition is not true, the entire condition is considered false.

Lines 3 and 4 perform in-line math. The equation of PRICE minus COST is performed first. The parentheses override the normal order of evaluation. The result of PRICE-COST is stored in a temporary result, and that result is divided by PRICE. The result of the division operation is stored in another temporary result. The next thing that happens is the comparison of the second temporary result to the number 10. If the result is less than 10, the condition is true. The OR operator is bypassed for now and the evaluation of COST = 0 is performed. If COST is equal to 0, that portion of the expression is true. The OR operator is used to test whether the left or right side operands evaluate to true. If either side is true, the condition is considered true.

Line 6 performs a WHEN (in-line case) testing indicator 01 (*IN01) for an ON condition. If the indicator is on, the condition is considered true.

Line 7 performs a DOU (Do Until) operation. The expression used as the conditioning expression is simply *INLR. This is a valid condition, and evaluates to either a true or false condition. Because an indicator is either off or on, when its condition is tested, it returns true when the indicator is on and false when the indicator is off. To reverse this conditioning, specify the NOT operator in front of the indicator or simply evaluate the indicator to be equal to *OFF, as in DOU *INLR = *OFF.

Expressions in Declarative Statements

Built-in functions can be used in any expression. On the definition specification, expressions are used as arguments for the various keywords. Expressions also are useful in establishing field referencing. When the properties of one item change, other items that depend on those properties also change. This is not the only use for expressions on the definition specification, but it is the most useful. Other uses include setting the initial value of a field, specifying the location of a data structure subfield, or calculating a mathematical formula.

Only expressions that can be analyzed at the time the source code is compiled may be specified. Hence, only literals, predefined values, and certain built-in functions are supported. Specifically, the value (content) of fields, arrays, and data structures cannot be used in an expression on the definition specification. Figure 3.3 illustrates several valid expressions as used on the definition specification function/keyword area.

```
.....DName++++++++++EUDS.......Length+TDc.Functions++++++++++++++++++++++++++++
0001 D Amt_Len         S               5P 0 INZ(%size(AmountDue))
0002 D Dft_Comp        C                    Const('Skyline Pigeon Productions')
0003 D Company         S                    LIKE(CORP_NAME) INZ(DFT_COMP)
0004 D Radius          C                    Const(16)
0005 D PI              C                    Const(3.1415926)

0006 D AreaSphere      S               7P 3 INZ(4*PI * RADIUS **2)
0007 D CubeRoot        S                    LIKE(AMT_LEN) INZ(Radius**(1/3))
0008 D NewName         S              +4A   LIKE(CORP_NAME)
0009 D                                      INZ('The ' + DFT_COMP)
```

Figure 3.3: Expressions on the definition specification.

Expressions in Parameters and Return Values

The CALLP and RETURN operations support expressions in the extended factor 2. The CALLP operation is used to evoke either a subprocedure or a separate program. The RETURN operation is used to specify the return value from a subprocedure to its caller.

An expression can be specified for a parameter of a subprocedure call when that parameter is defined as CONST. In other words, the CONST keyword must be specified for the parameter in order for the RPG IV syntax checker to accept an expression for the parameter.

1 Andrews, Robert. *The Columbia Dictionary of Quotations*. New York: Columbia University Press, 1993.