

Expressions in Parameters

An expression in a parameter definition can take one the following values:

- An explicit value; for example, 1000000
- Another parameter; for example, a parameter called bodDestPort
- A mathematical expression that can include a combination of:
 - Parameters
 - Numbers—Integers and floating point numbers
 - Strings
 - IPv4 addresses
 - Ranges of numbers, strings, and addresses
 - Lists of values, such as lists of protocols
 - Maps—List of pairs of attributes and corresponding values
 - One keyword, **not**
 - Separators
 - Operators

For example, `x = 1 ? rate : 2*rate`

The syntax for mathematical expressions is based primarily on Java syntax, although a few items use a proprietary syntax. When evaluating mathematical expressions, the SRC software:

- Follows a defined order for the precedence of operators (see “Expressions in Parameters” on page 1).
- Performs all evaluations in long integer format until it finds an argument or result that is in Java floating point number format. Subsequently, the software performs evaluations in Java double floating point number format.
- Evaluates only subordinate expressions that meet the conditions for evaluation.
 - Evaluates only subordinate expressions that contain numbers and not parameters.
 - Stops the evaluation and substitutes the partial evaluation if an argument in double floating number format becomes an argument to an operator that takes only integers.
- Behaves in the same way as a Java evaluation if intermediate evaluations exceed or fall below the long integer range or the double floating point number range.
- Follows the Java rules for raising exceptions. For example, the software raises an exception if:
 - An evaluation involves a division by zero.

- Literal numbers exceed the long integer limit or the double floating point number limit.

The following sections describe how to format the items that you can use in an expression.

Specifying Parameter Names

Observe the following rules when you are specifying parameter names:

- Enter a string of alphanumeric characters starting with a letter.
- Do not use spaces or special characters. For example, do not use the at sign (@) in a parameter name.
- You can use the underscore (_) and the dollar sign (\$). Use the dollar sign to encode special characters by entering the Unicode equivalent of the character in hexadecimal format after the dollar sign. For example, use \$0040 to encode the at sign (@).

Formatting Numbers

Observe the following rules when you are formatting numbers:

- Enter a digit after the decimal point in a floating point number. For example, you can use the number 4.0, but not the number 4.
- Do not enter characters that specify the type of number after that number. For example, do not enter the character L after a number to indicate that the number is a long integer.

Formatting Strings

Use Java syntax for strings; enclose strings in double quotation marks.

Example—" engineering"

Observe the following rules when you are formatting strings:

- Do not use octal escape sequences in strings. For example, do not use the escape sequence \137 in a string.
- Do not use Unicode escape sequences. For example, do not use the escape sequence \u80A6 in a string.

Using IPv4 Addresses

Use the following format for IP addresses:

<string>.<string>.<string>.<string> | '<string>.<string>.<string>.<string>'
 <string> is a set of digits in the range 0–255
 Example—"192.0.2.1"

Single quotation marks around an item indicate that it represents an address; however, for IPv4 addresses, the quotation marks are optional.

Specifying Ranges

To specify a range of numbers, strings, and addresses, use two dots between the arguments.

Example—192.0.2.1..192.0.3.1

Formatting Lists

To specify a list of values, enclose a set of subordinate expressions separated by commas in a pair of square brackets.

Example—[ip, icmp, ftp]

Formatting Maps

Maps are used to specify values that have optional and interdependent attributes. For example, when you define an application object through the Enterprise Manager portal, you can select a number of attributes and specify particular values for them. Depending on the value of the attribute, other attributes are possible or required.

To format a map, specify a list of pairs of attributes and corresponding values. Separate the pairs with commas, and enclose the list in curly brackets (braces).

Example—{applicationProtocol="ftp" , sourcePort=123, inactivityTimeout=60}

Using Keywords

The SRC software ignores all Java keywords in substitutions, so that you can use Java keywords for identifiers such as variable names, function names, and attribute names in maps. The SRC software accepts one keyword, **not**, which is used to indicate conditions that do not match a specified value. For more information about the **not** keyword, see “Expressions in Parameters” on page 1.

Using Separators

You cannot use a dot (.) as a separator. You can use other Java separators in the ways that Java supports.

Using Operators

Table 1 on page 4 shows the operations and corresponding operators that the SRC software supports for substitutions. Most of the operators are Java operators, although a few operators are proprietary. You cannot use Java operators that do not appear in this table.

Table 1: Operations That You Can Use in Expressions

| Operation | Operator | Number of Arguments | Result If Different from Java Conventions | Conditions for Evaluation | Example |
|---|----------|---------------------|---|---------------------------------|-----------------|
| Bitwise AND of the arguments | & | Two | | Both arguments must be integers | 234567 & 876543 |
| Bitwise exclusive OR of the arguments | ^ | Two | | Both arguments must be integers | 234567 ^ 876543 |
| Bitwise inclusive OR of the arguments | | Two | | Both arguments must be integers | 234567 876543 |
| Bitwise negation of the argument | ~ | One | | Argument must be an integer | -234567 |
| Difference between two arguments | - | Two | | Both arguments must be numbers | 876543 - 234567 |
| Division of the first argument by the second argument | / | Two | Result of operation in double format | Both arguments must be numbers | 589 / 756 |
| Equal | = | Two | Nonzero number if the arguments are equal | Both arguments must be numbers | rate == 5 |
| Greater than | > | Two | Nonzero integer if the first argument is greater than the second argument | Both arguments must be numbers | rate > 5 |
| Greater than or equal to | >= | Two | Nonzero integer if the first argument is greater than or equal to the second argument | Both arguments must be numbers | rate >= 5 |

Table 1: Operations That You Can Use in Expressions *(continued)*

| Operation | Operator | Number of Arguments | Result If Different from Java Conventions | Conditions for Evaluation | Example |
|---|----------|---------------------|---|---------------------------------|--------------------------|
| If... then... else... operation | ?: | Three | If the first argument is nonzero, then the result is the second argument, else the result is the third argument | First argument must be a number | "x == 1 ? rate : 2*rate" |
| Less than | < | Two | Nonzero integer if the first argument is less than the second argument | Both arguments must be numbers | rate < 5 |
| Less than or equal to | < = | Two | Nonzero integer if the first argument is less than or equal to the second argument | Both arguments must be numbers | rate < = 5 |
| Logical AND | && | Two | Nonzero integer if both the arguments are nonzero | Both arguments must be numbers | x == 1 && y > = 5 |
| Logical NOT | ! | One | Zero if the argument is nonzero | All arguments must be numbers | ! x == y |
| Logical OR | | Two | Nonzero integer if at least one of the arguments is nonzero | Both arguments must be numbers | x == 1 y > = 5 |
| Maximum of the arguments, max() = -infinity | max() | Zero or more | | All arguments must be numbers | max (1, 3, 2, 4) |

Table 1: Operations That You Can Use in Expressions *(continued)*

| Operation | Operator | Number of Arguments | Result If Different from Java Conventions | Conditions for Evaluation | Example |
|--|----------------------|---------------------|--|---|--------------------------------|
| Minimum of the arguments, <code>min()</code> = + infinity | <code>min()</code> | Zero or more | | All arguments must be numbers | <code>min (1, 3, 2, 4)</code> |
| Negation | <code>-</code> | One | | Argument must be a number | <code>-5</code> |
| Not equal | <code>!=</code> | Two | Nonzero integer if the arguments are not equal | Both arguments must be numbers | <code>rate != 5</code> |
| Not match | <code>not</code> | One | | None – expressions with this operator cannot be evaluated | <code>not 192.0.2.1</code> |
| Product of the arguments | <code>*</code> | Two | | Both arguments must be numbers | <code>rate*2</code> |
| Raise the first argument to the power of the second argument | <code>**</code> | Two | | Both arguments must be numbers | <code>2**16</code> |
| Range from the first argument to the second argument | <code>..</code> | Two | | None—expressions with this operator cannot be evaluated | <code>0..49</code> |
| Remainder of division of the first argument by the second argument | <code>%</code> | Two | | Both arguments must be integers | <code>5%2</code> |
| Round off the argument to the closest number | <code>round()</code> | One | Integer closest to the argument | Argument must be numbers | <code>round(986532.654)</code> |

Table 1: Operations That You Can Use in Expressions *(continued)*

| Operation | Operator | Number of Arguments | Result If Different from Java Conventions | Conditions for Evaluation | Example |
|---|------------------------|---------------------|--|---------------------------------|--|
| Round the argument down | <code>floor()</code> | One | Biggest integer less than or equal to the argument | Argument must be numbers | <code>floor (986532.654)</code> |
| Round the argument up | <code>ceiling()</code> | One | Smallest integer greater than or equal to the argument | Argument must be numbers | <code>ceiling (986532.654)</code> |
| Shift the first argument left by the number of bits in the second argument | <code>< <</code> | Two | | Both arguments must be integers | <code>986532 < < 2</code> |
| Shift the first argument right by the number of bits in the second argument | <code>> ></code> | Two | | Both arguments must be integers | <code>986532 > > 2</code> |
| Sum of the arguments | <code>+</code> | One or two | | Both arguments must be numbers | <code>876 + 345</code> <code>+ 855</code> |

The precedence of the Java operators is the same as the precedence in Java; if you are unsure of the precedence of the operators, you can use parentheses to ensure that the software evaluates expressions in the desired way. For example, the following logical OR expression does not need parentheses.

```
x==1 || y>=5
```

You can, however, include parentheses as follows:

```
(x==1) || (y>=5)
```

The following list shows the precedence of the operators from lowest precedence to highest precedence:

- not
- ..
- ?:
- ||
- &&
- |
- ^
- &
- = , = , !=
- < , > , < = , > =
- < < , > >
- + , - (binary)
- * , / , %
- **
- + , - (unary)
- ~ , !

- Related Topics**
- Parameters and Substitutions
 - Formatting Substitutions
 - Adding Comments to Substitutions
 - Parameter Names and Types