Criteria Expressions for Matching Values with the Message Responder

This topic explains the criteria expressions used to match values in the Message Responder and SQL Responder tools. Sections include:

Criteria Expressions Overview

Criteria Expressions is a concise syntax used for matching values in:

- **SQL Responder** parameter matching.
- **Message Responder** data source correlation.

This criteria expression syntax supports comparing strings and numbers as well as matching strings using wildcards and regular expressions. This syntax is used for:

- Each entry in a data source column that is used for correlation
- The SQL Responder's parameter criteria.

Virtualize evaluates an expression against the input value supplied by the responder. If the expression returns true, then the value is considered a match.

Using Criteria Expressions with the SQL Responder

To understand this syntax and its usefulness, consider a simple SQL Responder. Suppose that Virtualize captured the following two SQL queries when recording of database traffic:

```
select id, name from widget where yearCreated = 2008 and category = 'tool-metal'
select id, name from widget where yearCreated = 2011 and category = 'construction-molding'
```

Both of these statements correspond to the following SQL query template in the SQL Responder:

```
select id, name from widget where yearCreated = ${yearCreated} and category = ${category}
```

Using the recorded data, Virtualize generated a single query template with the following table of parameter criteria. Of interest are the "yearCreated" and "category" columns, which correspond to the values seen in the WHERE clause of the SQL statements.

yearCreated	category	ResultSet File	Response Time (ms)
2008	tool-metal	database_widgets\file1.csv	0
2011	construction-molding	database_widgets\file2.csv	0

As configured, the SQL Responder will be able to return a result set only for the two SQL statements that Virtualize captured during recording. Virtualize configured the parameter criteria to match the exact values seen in during recording. The responder does not have a result set for a "tool-metal" widget created in 2005. Nor does it have a result set for widgets created in 2011 (besides those categorized as "construction-molding").

Suppose that you want to handle all "tool-metal" widgets in your responder and that you can do so by grouping the widgets by the year they were created. For all "tool-metal" widgets created before 2008, you can return one result set. For all "tool-metal" widgets created during or after 2008, you can return another result set (the one that Virtualize captured during recording). To do this, you can use numeric comparisons for the "yearCreated" parameter. You will need to manually add another result set to the responder for the widgets created before 2008. In the end, your table of parameter criteria will look like this:

yearCreated	category	ResultSet File	Response Response
[< 2008]	tool-metal	database_widgets\file3.csv	0
[>= 2008]	tool-metal	database_widgets\file1.csv	0
2011	construction-molding	database_widgets\file2.csv	0

Note the entries in the "yearCreated" column for the rows with a "category" of "tool-metal". Both such entries are surrounded with square brackets, which indicates that instead of treating the entry as a literal string to match, the entry should be treated as a more sophisticated criteria expression. The expression contained within the brackets evaluates to true or false. The expression *[< 2008]* returns true if the input value is less than the number 2008. The input value is taken from the incoming SQL statement: the SQL Responder parses the SQL, finds the value that corresponds to "${yearCreated}" in the query template, and provides that value as the input value to the *[< 2008]* expression.

In most programming languages a less-than comparison has values on both sides of the less-than operator, such as `variable1 < variable2`. In criteria expressions, there is no value shown to the left of the operator: the value to the left of the operator is implicitly the input value. The expression *[< 2008]* checks if the input value is less than 2008, while *[>= 2008]* checks if the input value is greater than or equal to 2008.

In addition to < and >=, criteria expressions support <=, >, == (equals), and != (not equals). You can use the same comparison operators to compare both numbers and strings.

In the first two rows of the parameter criteria table, the "category" value is tool-metal. The expression tool-metal is equivalent to the expression *[== "tool-metal"]*. In the interest of concision, Virtualize uses the expression as a literal string to match unless the expression starts with the *[* character. A side effect of this is that to match a literal string that starts with "[", you must explicitly use the == operator. To match the literal string "[xyz", you must use the expression *[== "[xyz"]*.

In addition to the comparison operators that can be used for numbers and strings, Virtualize supports additional operators for analyzing strings. Suppose that you want to use another result set for all widgets created in 2011 with a category that starts with "construction-". To do this, you can add another result set to the responder and use the expression *[like "construction-*"]* for the "category" parameter; this would match inputs such as construction-support and construction-something-misc. The like oper-ator uses * and ? as wildcards. The * wildcard matches zero or more characters. The ? wildcard matches exactly one character.

Suppose that you want to use yet another result set for all widgets created in 2011 with a category of "circuit" or "circuit-" followed by any string, but not a category such as "circuitbreaker". To do this, you can add another result set to the responder to match the "category" parameter using a regular expression. You can use the criteria expression *[=~ /^circuit(-.+)?$/]*. The regular expression will match *circuit* or *circuit* followed by a - and one or more characters, such as *circuit-electric*.

Your table of parameter criteria now looks like this:

yearCreated	category	ResultSet File	Response Time (ms)
[< 2008]	tool-metal	database_widgets\file3.csv	0
[>= 2008]	tool-metal	database_widgets\file1.csv	0
2011	construction-molding	database_widgets\file2.csv	0
2011	[like "construction-*"]	database_widgets\file4.csv	0
2011	[=~ /^circuit(-.+)?$/]	database_widgets\file5.csv	0

Suppose that you always want the SQL Responder to return a result set for any SQL statement that matches the query template you've been using. For any widget that has not been matched by the other parameter criteria and was created between 2000 and 2007, you have one result set; for any otherwise unmatched widgets, you have a final result set.

To match a "yearCreated" between 2000 and 2007, you can use two numeric comparisons combined with the *and* operator. This expression will match the desired range: *[>= 2000 and <= 2007]*. The and operator combines two criteria expressions and returns true if and only if both expressions return true. The or operator works similarly but returns true if and only if one or both of the expressions return true.

To match any value for the "category" parameter, you can use the criteria expression *[*]*, which matches any input. Alternatively, you can use the expression *[]*, which means no criteria and therefore trivially evaluates to true. Virtualize supports *[*]* because its meaning is more obvious. Note that the empty expression — which would be represented in the table of parameter criteria as an empty table cell — does not match any input. Rather, it matches an input that is a zero-length string.

Finally, to match any otherwise unmatched widgets, you can add a final result set as the last row in the criteria table, and in that row use the expression *[*]* for each of the "yearCreated" and "category" parameters.

Your table of parameter criteria now looks like this:

yearCreated	category	ResultSet File	Response Time (ms)
[< 2008]	tool-metal	database_widgets\file3.csv	0
[>= 2008]	tool-metal	database_widgets\file1.csv	0
2011	construction-molding	database_widgets\file2.csv	0
2011	[like "construction-*"]	database_widgets\file4.csv	0
2011	[=~ /^circuit(-.+)?$/]	database_widgets\file5.csv	0
[>= 2000 and <= 2007]	[*]	database_widgets\file6.csv	0
[*]	[*]	database_widgets\file7.csv	0

While setting up data source correlation is different from configuring a SQL Responder, data source correlation in Virtualize supports the same criteria expression syntax.

Criteria Expression Syntax

A criteria expression is interpreted as either a literal string to match or as a more sophisticated expression.

If the expression does not begin with */*, then Virtualize interprets the expression as a literal string to match. An expression that starts with */* but does not end with */* is invalid: it cannot be parsed. An expression contained within square brackets supports several operators that evaluate against the input value — several comparison operators, and string pattern matching using the *like* operator and regular expressions — and boolean operators to combine criteria expressions.

Virtualize evaluates an expression against an input value, which can be either a number or a string. Each criteria evaluates to true or false. True indicates a successful match.

Comparison Operators

Virtualize supports several comparison operators than you can use to compare both numbers and strings. The supported operators are:

Operator	Meaning
<=	less than or equals
<	less than
==	equals
>=	greater than or equals
>	greater than
!=	not equals

For example, to check that the input, assumed to be a number, is less than 57, use */[< 57]*. The input value is implicitly the left operand. The right operand is the value against which you want to compare the input value.

A string is contained in double-quotes. For example: */[< "xyz"]*.

The expression *abc* is equivalent to the expression */[== "abc"]*.

Virtualize compares strings lexicographically. For example, A < AA < Z < Zoo < a < aa < z < zoo. Note that the comparison is case sensitive. Virtualize compares the Unicode value of each character; if two string are the same except that one string is longer, the shorter string is less than the longer string.

A string within double-quotes can contain the following escape sequences:

Escape Sequence	Meaning	ASCII Character Code (Hex)
\b	backspace	0x08
\t	tab	0x09
\n	newline	0x0A
\f	formfeed	0x0C
\r	carriage return	0x0D
\"	double-quote character	0x22
\'	single-quote character	0x27
\\	backslash character	0x5C

Java supports the same escape sequences. The \' escape sequence for the single-quote character is strictly never necessary, but it is offered for compatibility with Java string syntax.

Numbers and strings are treated differently. The expression */[== "30"]* matches the input string 30 but not the input string *30.0*. However, */[== 30]* matches both input strings. Both the right operand and the input value have a type: either number or string. If both are numbers, Virtualize performs a numeric comparison. If exactly one is a number, Virtualize attempts to convert the other operand to a number. If successful, Virtualize performs a numeric comparison; otherwise, Virtualize convert to a string the operand that is a number and performs a string comparison.

The presence or absence of double-quotes determines if the right operand is a number. The SQL Responder determines the type of an input value by parsing the SQL statement: a value in the SQL statement that is contained within single-quotes is a string; a numeric value that is not in single-quotes is a number. In data source correlation in the Message Responder, all input values are strings: to do a numeric comparison, the right operand must be a number.

The following table shows how you can use the comparison operators and the input values that match.

Expression	Matches	No Match
[== 55]	number or string: 55, 55.0	number or string: *60*
[> 55]	number or string: 56, 56.0, 3.1e24	number or string: *55, 10*
[== "55"]	number: 55, 55.0 string: 55	number: *60* string: *55.0, 60*
[> "55"]	number: 56, 56.0, 3.1e24 string: 56, 55.0	number: *55, 55.0, 10* string: *55, 050, 3.1e24, abc*
[== "greetings"]	*greetings*	*xyz*
greetings	*greetings*	*xyz*
[>= "abc"]	*abc, abca, greetings*	*ABC, ab, 11*
[== "this is \"quoted\""]	*this is "quoted"*	*this is \"quoted\"*
this is "quoted"	*this is "quoted"*	*this is \"quoted\"*

To further illustrate the syntax, the following table shows some unparseable expressions.

Unparseable Expression	Parse Error
[xyz	Literal text begins with bracket. Use == opera-tor.
[55]	No operator. Use == operator.
["something"]	No operator. Use == operator.
[== 55	Missing closing bracket.

String Pattern Matching

In addition to the comparison operators, you can use the like operator and regular expressions for more sophisticated string matching.

The *like* operator supports simple pattern matching using an expression of the form *[like alpha?beta*"]*. The wildcard character *?* matches any single character. The wildcard character *** matches zero or more characters. If the input value is a number, then Virtualize converts the input to a string.

Regular expressions support more powerful pattern matching using an expression of the form *[~= /pat-tern/]*. If the input value is a number, then Virtualize converts the input to a string. Similar to Perl, the =~ indicates a regular expression comparison and the pattern itself is delimited by slashes. Virtualize uses the same pattern syntax as that used by Java regular expressions. For complete documentation on Java's pattern syntax, see the Javadoc for the class java.util.regex.Pattern. The one difference from Java's pattern syntax is that to match a / character, you must escape it with a backslash.

A regular expression comparison returns true if some substring of the input string matches the pattern. If you want to match *abc* but not *abc123* or *123abc* then you must use *[=~ /^abc$/] rather than [=~ /abc/]*. The *^* and *$* match the start and end of a string, respectively.

The following table shows how you can use the comparison operators and the input values that match.

Expression	Matches	No Match
[like "a*b"]	*ab, axb, axxb,axyzb*	*abx*
[like "a?b"]	*axb, aab, abb*	*ab, axxb, axyzb*
[=~ /^a.*b$/]	*ab, axb, axxb,axyzb*	*abx*
[=~ /^a.b$/]	*axb, aab, abb*	*ab, axxb, axyzb*
[=~ /^id-\d+$/]	*id-1, id-456*	*id-, id-xyz, id-2b*
[=~ /beta/]	*beta, alpha-beta, alpha-beta-gamma*	*alpha, BETA*
[=~ /^beta$/]	*beta*	*alpha, BETA, alpha-beta, alpha-beta-gamma*
[=~ /(?i)^beta$/]	*beta, BETA*	*alpha, alpha-beta, alpha-beta-gamma*
[=~ /^.*\/.*$/]	*alpha/beta, /*	*alpha, alpha\beta*
[=~ /^.*\\.*$/]	*alpha\beta, *	*alpha, alpha/beta*
[=~ /^a\Q.*\Eb$/]	*a.*b*	*ab, axb, axxb, axyzb, abx*

Compound Expressions

Virtualize supports combining criteria expressions with the *and* and or operators. For example, *[>= 0 and <= 9]* returns true if the input value is between 0 and 9.

The and operator has higher precedence than the or operator. You can use parentheses around expressions. The expression *[like "*b*" or like "*a*" and like "*z*"]* is equivalent to *[like "*b*" or (like "*a*" and like "*z*")]*.

The operator names are case sensitive. This expression causes a parse error because *AND* is not a valid operator: *[> 0 AND < 10]*.

The following table shows how you can use the comparison operators and the input values that match.

Expression	Matches	No Match
[>= 0 and <= 9]	string or number: *0, 3.14159, 5, 9.0*	*string or number: -1, 10.12, 5.7e+12*
[like "*b*" or like "*a*" and like "*z*"]	*b, mnb, ba, bz, za*	*a, z*
[like "*b*" or (like "*a*" and like "*z*")]	*b, mnb, ba, bz, za*	*a, z*
[(like "*b*" or like "*a*") and like "*z*"]	*bz, za, zaabxyz*	*a, b, z, mnb, ba*
[(((== "abc")))]	*abc*	*xyz*

Using Criteria Expressions with the Message Responder

The general process of setting up data source correlation in the Message Responder — configuring values in a request to compare against values in a data source column — is described in Data Source Correlation Tab.

Note that the syntax described in this topic was introduced in Virtualize 9.2 and is specific to virtual assets (.pva files). The syntax from Virtualize 9.1 is no longer supported. Moreover, if you have imported any Parasoft .tst files with stubs into the Virtualize environment, Virtualize will use the legacy syntax for data source correlation for those items.