User-Defined Properties

Access: WAE Live > Settings and select User-Defined Properties tab

WAE Live acts on a set of objects (network elements) that are either discovered by WAE Collector or added via the snapshot process. For example, interfaces are discovered while demands can be created as part of the snapshot. Each object has a set of default properties (attributes) that are tracked in and retrieved from the datastore. For instance, an LSP's Shortest TE Path, a node's CPU, or an interface's Traffic In are all properties that are collected and stored by default. For a complete list of objects and properties, see the *WAE Live User Guide*.

Additionally, you can create user-defined properties to fit your specific needs. These user-defined properties are defined by and thus, derive their values from other properties. For example, rather than tracking incoming traffic (Traffic In) and outgoing traffic (Traffic Out), you might need to track the ratio of incoming to outgoing traffic (Figure 5-1).

As with default properties, these user-defined properties are available in the Explore and Analytics pages. These properties are available for all networks. That is, they are not configurable on a per-network basis.

Add Property Fields

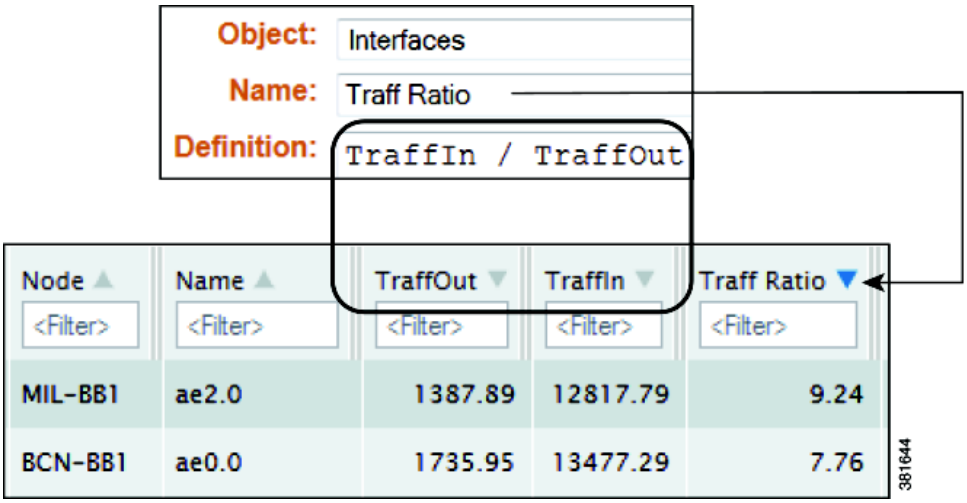
Each user-defined property consists of an object, a user-specified name, and a user-created definition.

- Object—Select the object type to which the newly created property applies.
- Name—Enter a property name. Once the property is saved, this name appears as a selectable property for the given object in Explore and Analytics pages.
- Definition—Each definition is an expression using one or more default properties. A few rules apply, as follows.
  - Properties used in the definition must be default properties. That is, you cannot use user-defined properties in defining other user-defined properties.
  - Property names in the Definition field are not case-sensitive.
  - A user-defined property can be an integer, a double (floating point number), string, or boolean.
  - You can use any combination of the following in the definition. You can group any of these with parenthesis, which are executed in the order in which they appear.
    - Arithmetic expressions: +, -, *, /, and % (Figure 5-1)
    - Relational operators: >, <, =, <=, and >=
    - Logical operators: AND, OR, and NOT

  - Regular Expression Functions
  - CASE Statements
  - Edit (pencil icon)—Edit the user-defined property.
  - Delete (trash can icon)—Delete a user-defined property. There is no undo.
- Preview—View the results that the object and definition together return.

Note For two-word properties, use one word in the definition. For example, for the Remote Node property, use "remotenode." For Traffic In and Traffic Out properties, use "traffin" and "traffout" in the definition.

Figure 5-1 Example of a User-Defined Property Definition Using Simple Arithmetic Expression



Regular Expression Functions

In the Definition field you can use regular expressions to filter results using regexp_extract and regexp functions.

regexp_extract

regexp_extract Syntax
regexp_extract(formatted_result, identifier1, pattern1, identifier2, pattern2, ...)

This searches a string with a list of regular expressions to produce a formatted string result compiled from back references. Back references are the results of capturing groups from the regular expressions and referencing them in order by $N, where N is a number that is greater than or equal to 1. This N identifies the captured string returned by a back reference in the order it appears in the matching pattern. At a minimum, only three arguments are needed for regexp_extract. You can use any number of pairs of attributes and expressions.

Example: In this example, both captured groups ($1 and $2) come from the regular expression for Description. The expression for Node does not have any capturing groups. It is used only as an additional filter. If any of the regular expressions fail to match, then the overall regexp_extract fails to match.

"regexp_extract("$1 ($2)", Description, ";interconnect to ([^\(]*) \(([^\)]*)\)", Name, "^(?!.*lo)")

regexp

boolean regexp Syntax
regexp(identifier1, pattern1)

This regexp syntax returns true, false, or null.

Example: If Node starts with 'palt', then sum the Traffic In and Traffic Out values. If the Node starts with something else, or is null, then no value is displayed.

case when regexp(node, '.*PALT.*') then traffin + traffout else null end

Example: If your node names include site names, this regexp syntax is useful for creating site properties for interfaces.

regexp_extract('$1', node, '([^-]+)-(.*)')

CASE Statements

The CASE statement allows conditional values based on a set of expressions. All result-expressions must be of the same type. That is, they can all be numeric or all be string, but they cannot be a mixture of both string and numeric.

CASE Statement Syntax
CASE WHEN <boolean-expression> THEN <result-expression>
[WHEN ...]
[ELSE <result-expression>]
END

Example: This example creates a user-defined property if a node name matches a remote node and if neither name is null.

case

when regexp_extract('$1', node, '(....).*\.(..)\..*') isnull then null

when regexp_extract('$1', remotenode, '(....).*\.(..)\..*') isnull then null

when regexp_extract('$1', node, '(....).*\.(..)\..*') = regexp_extract('$1', remotenode, '(....).*\.(..)\..*') then true else false end

Expression Syntax

Two syntaxes are used for the supported expressions: EBNF (Extended Backus–Naur Form) and a simpler query. Note that you can use one or multiple words for isNull and isNotNull.

- “isNull” can be represented by “is null”
- “isNotNull” can be represented by “is not null”

Simple Query

Following is the syntax for a simple query.

Simple Syntax
expression := value { +, -, *, /, %, IN, is [not] null, case } [expression]
when-expression := expression [{AND, OR, NOT } when-expression]
case := CASE WHEN when-expression THEN expression [ELSE expression] END
value := { object-attribute, number, string, regexp(), regexp_extract() }

EBNF

Table 5-1 is the grammar in EBNF format.

Table 5-1 SELECT Syntax in EBNF Format

SELECT Rule	Syntax
expression	case |
	logical
logical :=	relational {(OR, AND, SEMICOLON, COMMA) relational }
relational :=	additive {(<, <=, > , >=, =, ==, !=) additive }
additive :=	multiplicative {(+, -, ||) multiplicative }
multiplicative :=	unary {(*, /, %) unary }
unary :=	(!, not, ~, +, -) factor |
	factor isnull |
	factor

Table 5-1 SELECT Syntax in EBNF Format (continued)

SELECT Rule	Syntax
factor :=	NUMERIC | STRING | NULL | TRUE | FALSE | ID | regexp | regexp_extract | in_expr | '(' expression ')'
regexp :=	REGEXP '(' literal COMMA STRING ')' |
regexp_extract :=	REGEXP_EXTRACT '(' STRING COMMA reg_arg_list ')' opt-as
reg_arg_list :=	literal COMMA STRING |
in_expr :=	IN '(' string_list ')'
string_list :=	STRING COMMA string_list STRING
case :=	CASE when when_list opt_else END
when_list :=	when when_list |
when :=	WHEN expression THEN expression empty
opt_else :=	ELSE expression | empty
opt_as :=	AS ID |
literal :=	ID | STRING

Related Topics

- General Settings
- Map Settings