Explore and Analytics Settings

Access: WAE Live > Settings > Explore and Analytics tab

WAE Live supports multiple interface types that you can view on the Explore page and are used when creating reports from the Analytics pages. The Settings > Explore and Analytics tab lets you specify the default interface type used. You can override the default for the Explore pages, but not for the Analytics pages.

WAE Live acts on objects (network elements) that are discovered or created prior to WAE Live accessing the plan file. For example, interfaces might be discovered by WAE while demands thereafter can be added to the plan file. Each object has a set of default properties (attributes) that are tracked in and retrieved from the data store. For example, 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. In addition to these default objects, from the Settings > Explore and Analytics tab you can create user-defined properties to fit your specific needs.

Note These settings are configured for all networks; that is, they are not configurable on a per-network basis.

This section contains the following topics:

Default Interface Types Displayed

You can specify the default interface types to show in Explore and use in Analytics (reporting):

- Individual logical interface—An interface with an IP address that is not in a LAG. These names might have a decimal.

  Example: et-0/1/2.0, g2/0/0

- Individual physical interface—An interface that is not in a LAG and does not have an IP address. These interface names do not have a decimal.

  Example: so-0/2/2, g2/0/0

- LAG logical interface—An interface with an IP address that contains member interfaces.

  Examples: ae0.0, Bundle-Ether1, Port-Channel7

- LAG physical interface—The physical representation of an aggregated interface to which a LAG logical interface (such as ae0.0) is assigned.

  Example: ae0 for Juniper interfaces

- LAG member interface—An interface that is in a LAG.

  Example: g2/0/0

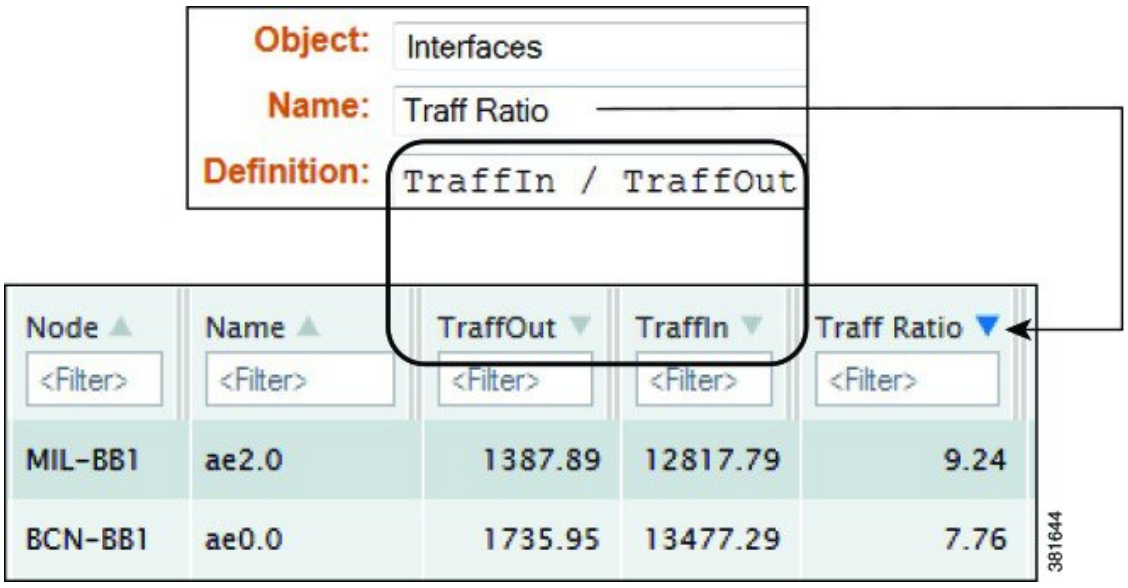
- Unknown—None of the preceding interface types.

User-Defined Properties

You can define your own properties to use when creating reports. User-defined properties 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 1: Example of a User-Defined Property Definition Using Simple Arithmetic Expression, on page 2).

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



As with default properties, these user-defined properties are available in the Explore and Analytics pages.

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

Adding a User-Defined Property

Procedure

Step 1 Navigate to **WAE Live > Settings > Explore and Analytics**.

Step 2 Click **Add Property**.

Step 3 From the Add New Property dialog box, choose the object type to which the newly created property applies (interfaces, interface queues, nodes, and so on).

Step 4 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.

Step 5 Enter a definition value.

Step 6 Click **Preview** to view the results that the object and definition return.

Step 7 Click **Save**.

Acceptable Definition Field Values

The definition field accepts alphanumeric characters and regular expressions. Each definition is an expression using one or more default properties. The following rules apply:

- 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 1: Example of a User-Defined Property Definition Using Simple Arithmetic Expression, on page 2)
  - Relational operators: >, <, =, <=, and >=
  - Logical operators: AND, OR, and NOT
  - Regular Expression Functions, on page 3
  - CASE Statements, on page 4
  - Edit (pencil icon)—Edit the user-defined property.
  - Delete (trash can icon)—Delete a user-defined property. There is no undo.

Regular Expression Functions

In the Definition field for your user-defined property, 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, the overall regexp_extract fails to match.

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

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, the overall regexp_extract fails to match.

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

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. 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

A simple query has the following syntax:

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 1: SELECT Syntax in EBNF Format , on page 5 lists the grammar in EBNF format.

Table 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

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