

Overview of Configuring Classification Scripts

Classification scripts are organized into rules. Each rule has a target and one or more match conditions. For example:

Subscriber Classifiers

```
subscriber-classifier {  
.  
.  
.  
rule rule-2 {  
    target <-unauthenticatedUserDn->;  
    condition {  
        "loginType == \"ADDR\"";  
        "loginType == \"AUTHADDR\"";  
    }  
}  
}
```

DHCP Classifiers

```
dhcp-classifier {  
.  
.  
.  
rule rule-2 {  
    target cn=default,<-dhcpProfileDN->;  
    condition {  
        1;  
    }  
}  
}
```

Interface Classifiers

```
interface-classifier {  
.  
.  
.  
rule rule-5 {  
    target /sample/junose/DHCP;  
    condition {  
        "interfaceName=\"fastEthernet*\"";  
        "interfaceName=\"atm*/*. *\"";  
    }  
}  
}
```

Classification Targets

A target is the result of the classification script that gets returned to the SAE. There are two special types of targets:

- No-match targets—Targets that begin with a - (single dash) are interpreted as no match. If the conditions of this target are matched, a no-match message is returned to SAE. You can use this type of target to exclude certain patterns or to shortcut known nonmatches. To speed up processing, use this target to specify interfaces that you do not want the SAE to manage.
- Script targets—The content of the script rule is interpreted when the classifier is initially loaded. The script rule can contain definitions of custom functions, which can be called during the matching process. Because you can insert arbitrary code into a script, you can use classification scripts to perform arbitrary tasks.

Because script targets use * (asterisks), you cannot use * in other types of targets.

Target Expressions

A target can contain expressions. These expressions can refer to an object in the SAE's memory or configuration, to specific matching conditions, or to another function or script.

Suppose the classification object in a subscriber classifier contains a field called `userName`. The classifier target `uniqueId = <- userName ->` is expanded to contain the actual content of the `userName` field before it is returned to the SAE; for example, for `userName = juser`, `uniqueId = juser` is returned.

Target expressions are enclosed in angle brackets and hyphens; for example, `<-retailerDn->`. The classifier expands expressions before it returns the target to the SAE. The expression is interpreted by an embedded Python interpreter and can contain variables and Python operations. In the simplest case an expression can be a single variable that is replaced with its current contents. Available variable names are all fields of the object passed to the classifier and names created with regular expression matching.

Because a scripting interpreter interprets expressions, more complex operations are possible. Examples are:

- Indexing—`var[index]` returns the element index of a sequence. The first element is at index 0.
- Slicing—`var[start : end]` creates a substring of the variable `var` starting at index startup to, but not including, index end; for example, `var = Hello`, `var[2:4] = ll`

Classification Conditions

You can configure multiple classification conditions for a rule. For example:

```
rule rule-2 {
  target /ent/EntDefault;
  condition {
    "pppLoginName=\"\"";
    "&interfaceName!=\"fastEthernet0*\"";
    "&interfaceName!=\"null*\"";
    "&interfaceName!=\"loopback*\"";
  }
}
```

If you prefix a condition with an & (ampersand) character, the condition is examined only if the previous condition matches.

If you prefix a condition with a | (pipe) character, the condition is examined only if the previous conditions have not produced a positive match.

You can use glob or regular expression matching to configure each target's conditions.

Glob Matching

Glob matches are of the form:

```
field = match
or
field != match
```

where match is a pattern similar to UNIX filename matching. Glob matches are case insensitive. “field != match” is true, if field = match is not true.

- *—Matches any substring.
- ?—Matches any single character.
- [range]—Matches a single character in the specified range. Ranges can have the form a-z or abcd.
- [!range]—Matches a single character outside the specified range.
- C—Matches the single character c.

The available field names are described for the specific classifiers. Examples are:

- interfaceName = fastEthernet3/0 # matches the string “fastEthernet3/0” directly.
- interfaceName = fast*3/1 # matches any string that starts with “fast” and ends with “3/1”
- interfaceName = fast*3/1.* # starts with “fast”, contains “3/1.” arbitrary ending
- interfaceName = fast*3/[2-57] # starts with “fast”, contains “3/” followed by 2,3,4,5 or 7

Regular Expression Matching

Regular expression matches are of the form:

```
field =~ re
or
field !~ re
```

where field !~ re is true if field = ~ re is not true. The regular expression is *re*. For a complete description of the syntax, see: <http://www.python.org/doc/2.0/lib/re-syntax.html>

You can group regular expressions with pairs of parentheses. If such an expression matches, the contents of the groups are made available for target expressions. Group

number n is available as $G[n]$, where n is the number of the opening parenthesis of the group. You can also name groups by using the special notation ($?P < name > \dots$).

Examples:

```
ifAlias =~ "SSP(.*)"
# match a string starting with "SSP". The remainder is stored
# in the variable "G[1]"
ifAlias =~ (?P<dn>name=(?P<name>[^,]*)).*
# match a string starting with " name=" . The whole match is
# stored in the variable " dn" . A submatch which does not
# contain any " ," -characters and starts after " name="
# is stored in variable " name"
```