Wily Best Practices

REGULAR EXPRESSIONS AND METRIC GROUPINGS IN INTROSCOPE: A Basic Guide

Abstract

Using regular expressions (regex) in Introscope Metric Groupings allows you harness the vast amount of data that Introscope gathers. This Best Practices paper provides a basic overview for using regex in Metric Groupings.

Related Information

- Wily Support Knowledge Base Article #229: Useful Introscope Regular Expressions
- Wily Education Services Course: Introscope 7: Up and Running

CA, Inc., Wily Technology Division www.wilytech.com wily-info@ca.com

Table Of Contents

1 Background

1.1 Metric Groupings

Metric Groupings serve a simple, yet essential purpose in Introscope. They are used to sort and filter large sets of data (Metrics) collected by an Enterprise Manager from its Agents. The created Metric Groupings can then be used by other Introscope Elements to further monitor that data, both graphically (in Dashboards and Report Templates) and programmatically (in Alerts and Persistent Collections).

1.2 Regular Expressions

A Regular Expression (regex) is a special text string that describes a search pattern. Wildcards, such as those used on an operating system command line, are only a small subset of the capability of a regex. However, they serve the same purpose — to condense the collection of objects to a short expression.

Introscope makes use of the Perl regex, which is well refined in its ability to filter data. It was originally derived from the Unix Version 8 regex standard. However, it has eclipsed the Unix standard in terms of features, and has been widely adopted by other languages and applications. Because it is so widely used, there is a great deal of good information available on it in both paper and digital form (links to Web sites and references to books appear in the last section of this document).

2 Usage

Note: This section offers only an introduction to Agent and Metric Expressions. For detailed information on how to create them, consult the Workstation Guide of the Introscope documentation.

Defining a Metric Grouping comprises filling in four fields: **Name**, **Management Module**, **Agent Expression**, and **Metric Expression.** The Agent Expression and Metric Expression fields are the two that describe the data that is being grouped, and both can be articulated using regex. Name and Management Module are not involved in the filtering process, so they will not be discussed in this document.

2.1 Agent Expressions

The Agent Expression defines to which Agents the Metric Grouping is to be applied. For example, the Agent Expression in Figure 1 will apply the Metric Grouping only on Agents in all domains that are named either WPS2-01 or WPS2-02. The Agent Expression is key when there are multiple JVMs that report to the same Enterprise Manager, but serve different purposes, such as running different Web sites in the same corporation. In this case, you need to limit which Agents your Metric Grouping are applied against by making use of regex to create that filter.

Agent Expressions can also be defined at the Management Module level. When defined there, the Agent Expression can be applied to every Metric Grouping defined in that Management Module, but may be overridden at the Metric Grouping level by defining an individual Agent Expression for that Metric Grouping. As seen in Figure 1, this choice is available to each Metric Grouping, allowing you to have a subset of your Metric Groupings all using the Management Module-defined expressions.

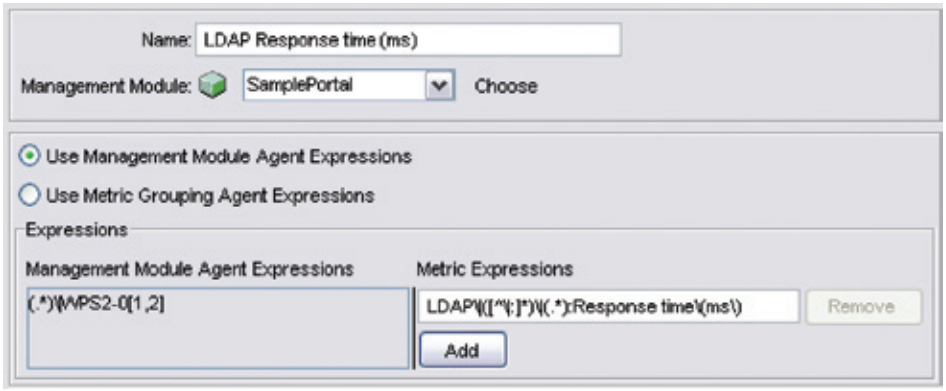

Figure 1: Properties of a Typical Metric Grouping

2.2 Metric Expressions

The Metric Expression is used to collect or filter all required data as delivered by those Agents defined by the Agent Expression. This includes the resource (the chain of folders leading to the Metric) and the Metric itself. For example, in the Metric Expression shown in Figure 1, all resources contained in the LDAP resource folder with Response Time (ms) Metrics in those resources, will be included in this Metric Grouping element. Metric Expressions typically require more careful consideration than Agent Expressions, as the number of Metrics to filter far exceeds the number of Agents.

2.3 Expression Structure

In order to choose which Agents and Metrics a grouping will return, the structure of the expressions needs to specify the branch of the Metric tree under which these Metrics appear. The mapping between levels in the tree and one-line expressions is a pipe (|). Each pipe signifies one level deeper in a tree. A colon at the end of the last level signifies that the statement following it signifies the actual Metric.

A fully-qualified Metric name follows the patterns seen here:

```
Hostname|Process|AgentName|Resource:Metric
```

```
Hostname|Process|AgentName|Category|SubCategory|...|SubCategory:Metric
```

Example: The Acme Company wants to track Bytes In Use of the GC Heap on their AcmeWest Agent, the fully-qualified Metric name would like like this:

```
AcmeHost|AcmeUSAProcess|AcmeWestAgent|GC Heap:Bytes In Use
```

Regex expressions specified for Agent and Metric Expressions are used to abstract the levels of Metrics in this way:

If the Acme Company wants to track all of the GC Heap stats for all of their Agents (across all hosts) that run the AcmeUSAProcess within one Metric Grouping, the Agent and Metric Expresisons would look like this:

```
Agent Expression: ([^\|:]*)\|AcmeUSAProcess\|([^\|:]*)
```

```
Metric Expression: GC Heap:(.*)
```

As will be discussed in the next section, special care has to be taken when applying regex in an Agent or Metric Expression. The various regex characters and their meanings will be discussed, and though this may initially seem difficult and overkill for simple filtering, you will soon discover the power and advantages of using regex for creating effective Metric Groupings.

3 Basics

|	Pipe (Level in Metric Tree)
\	Escape Character

As seen in the prior Acme example, pipes that signify a deeper level in a Metric are always preceded by a backslash (\). The backslash is an escape character, which directs the program to treat the following character as a literal character. Otherwise, pipes signify an OR statement in Perl regex. Other characters that commonly appear in Metric Expressions that may need to be escaped are double-quotes (“and ”), as they are used for regex groupings, and the period (.), which signifies any character.

Example: Acme wants to track Response Time (ms) of the AcmeServlet, which appears under the Servlets category. The Metric Expression would look like this:

```
Servlets\|AcmeServlet:Response Time \(ms\)
```

Notice that the pipe and parentheses need to be escaped.

()	Grouping

In Introscope, groupings are primarily used to logically split regex fragments within the expression. Groupings in Perl regex are also used to create variables to be read outside of the expression, but that does not apply to Introscope.

.	Any Character

The period (.) is commonly used with the asterisk (*), for example (.*), to specify any number of any characters. This combination should be used with care, as it often returns too much data.

*	0 or More (of the statement that precedes it)
+	1 or More (of the statement that precedes it)
?	0 or 1 Times

These three characters can follow a character or fragment to determine how many times that fragment is to be found. A plus (+) may be used instead of an asterisk (*) when it is necessary to ensure that the character, word, or fragment preceding it appears at least once.

Example: Acme wants to track Response Time (ms) of all servlets that appear under the Servlets category. The Metric Expression can look like this:

```
Servlets\|(.*):Response Time \(ms\)
```

[]	Character Classes — Inclusion/Exclusion
[abcde]	Any of the Letters or Numbers in That Set
[a–z]	Any Letters or Numbers in That Range
[^qwert]	Any Letters or Numbers **Other** Than That Set

Character classes are used to find any character that matches what is contained within the square brackets. For example, `[aeiou]` will match to any one vowel. Ranges can be used for letters or numbers (such as a–z, A–Z or 0–9) to find any letter or number within the specified range. This is useful when grouping a number of Agents, servlets, or portlets that have names specified with a naming or numbering standard. Also, multiple ranges can be specified in a character class, and are done so **without** a space between them. For example: `[a-ex-z0-3]` will find any character in the ranges a-e, x-z, or 0-3.

Using the carat (^) character at the head of the contents of a character class ensures that the class looks for any character **not** contained in the list that follows it.

Example: Acme wants to track Response Time (ms) of all servlets that start with the letters a, b, c, d, e, or f, and appear under the Servlets category. The Metric Expression can look like this:

```
Servlets\|[a-f](.*):Response Time \(ms\)
```

Notice that the period-asterisk (.*) appears after this fragment to pick up the rest of the characters in the servlet name.

4 Useful Fragments

(.*)	Wildcard

This fragment is used to find any number of any characters. As mentioned before, this statement should be used with care, as it often returns a large amount of data. However, it is useful when trying to take the "shotgun" approach of grouping Metrics to get the total picture.

([^|:]*)	Represents One Level of Metric Tree
([^|:]+)	Variation — Guarantees One Level of Metric Tree

This is a very commonly used fragment in constructing a Metric Grouping. If you break it down by segment, it has a character class that is looking for any character that is **not a pipe** (|) or a colon (:), and the asterisk (*) signifies that it is looking for any such characters any number of times. What this amounts to is a regex fragment that finds anything that exists between two pipes or a pipe and colon, which is any single level of the Metric tree.

This fragment is especially useful when trying to cut out "Blamed" Metrics from your Metric Grouping. These are the Metrics that appear as "Called Xxxx" below a resource. These Metrics are useful for deep-diving a problem, but can give misleading and/or redundant Metrics when grouped with their parent resource, and thus are not always desired in a grouping.

Example: Acme wants to track Response Time (ms) of all servlets that appear under the Servlets category, but not their blamed metrics (calls to other servlets, backends, and EJBs). The Metric Expression can look like this:

```
Servlets\|([^\|:]*):Response Time \(ms\)
```

The variation that appears above can be used to guarantee that one level of the Metric tree exists between other fragments specified in your expression. It can be customized to guarantee one level that has more specific data. This is most useful when trying to group the Metrics of members of a category (for example, Servlets) without grouping in the summary Metric for that category, if such exists.

Example: Acme wants to track Response Time (ms) of all servlets that appear under the Servlets category, but not their blamed Metrics, nor the summary Servlets|Response Time (ms). The Metric Expression can look like this:

```
Servlets\|([^\|:]+):Response Time \(ms\)
```

(w1|w2|w3...)	Alternative Word Grouping

Another important use of groupings in Introscope is in conjunction with the pipe as an OR statement as a way of expressing alternative phrases that can appear. For example, (Word1|Word2|Word3) will look for any of these three words at that place in the expression.

Example: Acme wants to track Response Time (ms) of servlets that start with the word "Acme," "American," or "National" under the Servlets category. The Metric Expression can look like this:

```
Servlets\|(Acme|American|National)([^\|:]*):Response Time \(ms\)
```

Notice that (`[^\|:]*`) appears after the (`Acme|American|National`) fragment to pick up the rest of the characters in the servlet name.

(?=xxxxx)	Positive Lookahead
(?!=xxxxx)	Negative Lookahead

The purpose of the lookahead fragments is to ensure that the fragment occurs (or doesn't occur) at that specific place in the expression, after all that precedes it in the expression. Negative lookahead is particularly useful if you want to exclude a branch of a Metric tree, as it will exclude all Metrics that have that fragment matched at that place in the expression.

Example: Acme wants to track Response Time (ms) of all methods in the MainEJB that begin with the word "Action" that appear under the EJB category. The Metric Expression can look like this:

```
EJB\|MainEJB\|(?=Action)([^\|:]*):Response Time \(ms\)
```

Notice that (`[^\|:]*`)appears after the (`?=Action`) fragment to pick up the rest of the characters in the EJB name.

Example: Acme wants to track Response Time (ms) of all methods in the MainEJB that **do not** begin with the word "Do" that appear under the EJB category. The Metric Expression can look like this:

```
EJB\|MainEJB\|(?!Do)([^\|:]*):Response Time \(ms\)
```

Notice that (`[^\|:]*`)appears after the (`?!Do`) fragment to pick up the rest of the characters in the EJB name.

5 Comprehensive Examples

```
WP_Custom\|([^\|:]+)BusinessObj\|(?!retrieve)([^\|:]*)(?=Action):
Response Time \(ms\)
```

This expression will find these metrics:

✓ `WP_Custom|ThisBusinessObj|performCustomerAction:Response Time(ms)`

✓ `WP_Custom|ThatBusinessObj|doAction:Response Time(ms)`

Because the (`[^\|:]*`) appears before (`?=Action`), nothing has to appear there, as the asterisk indicates "0 or more" times.

The expression will not, however, find these metrics:

✗ **`WP_Other`**`|ThisBusinessObj|doAction:Response Time (ms)`
- *Requires WP_Custom at front of Metric*

✗ `WP_Custom|`**`BusinessObjHelper`**`|performCustomerAction:Response Time (ms)`
- *([^\|:]+) requires something before "BusinessObj"*

✗ `WP_Custom|ABusinessObj|doAction|`**`Called WP_Custom`**`|init:Response Time (ms)`
- *"Called WP_Custom" has "|" surrounding it, prohibited by ([^\|:]*).*

✗ `WP_Custom|ThatBusinessObj|`**`retrieve`**`CustomerAction:Response Time (ms)`
- *This portion of the metric cant start with "retrieve" (?!retrieve).*

✗ `WP_Custom|ThisBusinessObj|getCustomer`**`Response`**`:Response Time (ms)`
- *This portion of the metric must end with "Action" (?=Action).*

6 Best Practices — Avoiding "The Clamp"

Metric Groupings incur overhead on the Enterprise Manager, as they require the EM to compile graphs and to render them on-screen. For this reason, the number of data points (each point in a graph line) that can be displayed is limited to 25,000. If there are more data points than this in the grouping, a red line will outline the graph (known as "the clamp") and the number of data lines shown will be limited in order to fall under the 25,000 limit.

There are several best practices that can be applied to avoid this situation:

1. If you are concerned with seeing the top offenders in a Dashboard (for example, highest response times) in a large set of data (for example, response times of all EJB methods), have one Metric Grouping collect all of the data, and enable a filter for the graph in the Dashboard to display the Top- (or Bottom-) N Metrics.
2. If you want to see all of the Metrics from a large set of data (for example, response times of all servlets), but there is too much to display in one Metric Grouping, logically split the data into several graphs using regular expressions. For example, these expressions can be used to alphabetically split servlet response times into three Metric Groupings:
   - `Servlets\|[a-iA-I]([^\|:]*):Response Time \(ms\)`
   - `Servlets\|[j-rJ-R]([^\|:]*):Response Time \(ms\)`
   - `Servlets\|[s-zS-Z]([^\|:]*):Response Time \(ms\)`
3. If you only want a subset of Metrics from a large set of data (for example, the response times of several servlets), you can either:
   a. Explicitly enter each metric in its own line in the Metric Grouping.
   b. Exclude metrics using the `(?!xxxxxx)` fragment. For example, this expression would group the response time of all servlets that don't begin with "Portal" or "WebSphere":

```
Servlets\|(?!Portal|WebSphere)([^\|:]*):Response Time \(ms\)
```

7 Further Reading

http://www.perl.com/doc/manual/html/pod/perlre.html
The documentation on perlre — the Perl Regular Expressions package.

http://www.regular-expressions.info/
This site offers a wealth of information about regular expressions, including a quick-start guide, tutorial, reference listing, and examples.

http://www.weitz.de/regex-coach/
The Regex Coach is an excellent tool to interactively build regular expressions and test them as you build them. It is freeware, and works on Windows and Linux.

Mastering Regular Expressions, by Jeffrey Friedl.
(Published by O'Reilly & Associates)
This book is widely accepted as one of the most comprehensive and well-written guides on regular expressions. It is an invaluable reference guide.

8 Bibliography

"Putting All JSP (or EJBs, Servlets, etc..) Metrics in One Graph!" Wily Knowledge Base. 27 January 2004. <http://support.wilytech.com/cgi-bin/wilytech.cfg/php/enduser/std_adp.php?p_faqid=56>

"Putting Useful Introscope Regular Expressions" Wily Knowledge Base. 26 March 2004. <http://support.wilytech.com/cgi-bin/wilytech.cfg/php/enduser/std_adp.php?p_faqid=229>

"Putting Regular expressions in the Agent and Attribute fields" Wily Knowledge Base. 12 March 2003. <http://support.wilytech.com/cgi-bin/wilytech.cfg/php/enduser/std_adp.php?p_faqid=260>

Friedl, Jeffrey. Mastering Regular Expressions. O'Reilly & Associates, Sebastopol, California, January 1997, ISBN 1565922573. <http://www.oreilly.com/catalog/regex2/index.html>.

Goyvaerts, Jan. "Regular-Expressions.info". May, 2006. <http://regular-expressions.info>.

"perlre — Perl regular expressions." Perl Documentation. Ed. Tom Christiansen. O'Reilly Network. May, 2006. <http://www.perl.com/doc/manual/html/pod/perlre.html>

About Wily Best Practices

Best Practices documents from CA's Wily Technology Division enable customers and partners to take advantage of Wily's knowledge and experience by providing downloadable papers on Web application performance management topics. If you have any questions about the information in this Best Practices document, please contact Wily Support at 1 888 GET WILY ext. 1. We welcome your comments on the content of this document.

Wily Services

In addition to our software products, CA's Wily Technology Division offers Professional Services and Education Services to better enable customers and partners to become world-class managers of Web applications. If you would like more information about how to take advantage of Wily's services offerings, please don't hesitate to contact us.

Professional Services

Members of Wily's Professional Services staff are industry experts in Web application management best practices. Working closely with your IT team, they apply their experience to maximize the performance and availability of your mission-critical Web applications.

U.S.
1 888 GET WILY (438-9459), ext. 2 < U.S. toll free >
+1 415 562 2000, ext. 2 < direct >

EUROPE
+44 (0) 1753 57773 < direct >

ASIA PACIFIC
+65 6337 2822 < direct >

Education Services

Wily's Education Services offers entry-level and advanced-level courses that give your staff complete knowledge of CA Wily Introscope and industry monitoring best practices. We offer regularly scheduled courses in two formats: Classroom and Virtual. We can also come to you. Contact us to discuss on-site training. Download our informational brochure from www.wilytech.com/services/education.html.

wily-education@ca.com
1 888 GET WILY (438-9459), ext. 3 < U.S. toll free >
+1 415 562 2000, ext. 3 < direct >