



EMC® Watch4net®

Version 6.3u2

Text Parsing Library Admin Guide

P/N: pending

Rev : 46738

EMC²

Copyright ©2009 - 2014 EMC Corporation. All rights reserved. Published in the USA.

Published April, 2014

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

The information in this publication is provided as is. EMC Corporation makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose. Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

EMC², EMC, and the EMC logo are registered trademarks or trademarks of EMC Corporation in the United States and other countries. All other trademarks used herein are the property of their respective owners.

For the most up-to-date regulatory document for your product line, go to EMC Online Support (<https://support.emc.com>).

Contents

Preface	iii
Purpose	iii
Intended Audience	iii
Document Organization	iv
Where to get help	v
1 Product Overview	1
2 Configuration	3
3 Executor	7
1 CLICommandExecutor	8
2 FileReader	9
3 RemoteCLICommandExecutor	13
3.1 Public key file example	15
4 FtpFileGetter	15
5 SOAPXmlRequestExecutor	16
5.1 SSL	18
6 CASXmlRequestExecutor	20
7 Cisco UCS	21
8 Chained Web Service Requests Executor	23
4 Parsing Engine	27
1 Performance Limitations	27
2 Tree Type Specificity	28
2.1 Single Line Tree	28
2.2 Table Like Tree	28
2.3 Regex Group Tree	28
3 Single Line Tree Example	29
3.1 Configuration	30

- 3.2 Explanation 31
- 3.3 Output Tree 32
- 4 Table Like Tree Example 34
 - 4.1 Configuration 34
 - 4.2 Explanation 35
 - 4.3 Output Tree 35
- 5 Regex Group Tree Example 36
 - 5.1 Configuration 36
 - 5.2 Explanation 37
 - 5.3 Output Tree 38
- 5 Data Finalizer 41**

Preface

Purpose

The purpose of this guide is to provide information about the *APG* Text Collector architecture and how to use it.

Intended Audience

This guide is intended to be read by any user of the *APG* Text Collector.

Document Organization

1. Product Overview	This section gives an overview of the product.
3. Executor	This section describes each executor and how to use them.
4. Parsing Engine	This section describes each parser and how to use them.
5. Data Finalizer	This section describes the usage of the data finalizer.

Where to get help

EMC support, product, and licensing information can be obtained as follows:

PRODUCT INFORMATION - For documentation, release notes, software updates, or information about EMC products, go to EMC Online Support at:

<https://support.emc.com>

TECHNICAL SUPPORT - Go to EMC Online Support and click Service Center. You will see several options for contacting EMC Technical Support. Note that to open a service request, you must have a valid support agreement. Contact your EMC sales representative for details about obtaining a valid support agreement or with questions about your account.

1

Product Overview

The Text Parsing Library is a versatile tool that allows a user to retrieve a text output from various source and parsed to filter only the needed component and give it in a tree of a logical representation of the data.

2

Configuration

This section will show an empty configuration file base on the collectingconfiguration.xsd schema with all the possible tags.

```
<configuration>
  <max-threads>The number of threads who will run in parallel to run the collection
  <parser-config>
    <parser-class>The parser class
    <tree-type>The type of the tree build, single-line-tree, table-like-tree or regex-group-
tree
    <incremental-parser>Tells if the parser will received only fragment of dataset. To this
date, only the RegexParser support the incremental parsing
    <master-delimiter>The delimiter used to first split the input of the parser. Omit it for a
carriage return delimiter
    <sub-delimiter skip-empty-tokens="True to remove empty tokens (Default=true)">
      <delimiter>The sub delimiter used for each line previously split by the master delimiter
      <trimmer>The optional text wrapping each tokens that must be removed
      <exception-list>The list of exception not to split (No regex allowed)
    OR
    <condition-list>
      <prefix depth="0">The regex that must match the characters before the delimiter
```

```

        <suffix depth="0">The regex that must match the characters after the delimiter
        <matches>True if the prefix and suffix must match to split at the delimiter
    </condition-list>
</sub-delimiter>
<tree-schema>
    <name>The regex name of the node
    <leaf>
        <name>The regex name of the leaf
        <group>The name of the regex corresponding to this leaf followed by a dash and the
group number (Only used for the regex-group-tree)
    </leaf>
    <node>Same as the tree schema for subsequent nodes and leaves
</tree-schema>
<parameter name="The_value_key">
    <value>A list of values for the given key
</parameter>
<regex-group-config group="The_group_index_of_the_parent_regex">
    <regex multiline="false" name="The_unique_name_of_this_regex">A regex made with groups to
extract sub regexes or values
    <sub-regex group="0"/>
</regex-group-config>
</parser-config>
<data-formatter-config>The formatting class
<data-handler-config>The handling class
<include-contexts>Location of an external execution contexts file
<contexts name="The_name_of_the_execution_context_for_logging_purpose">
    <properties name="The_name_of_the_execution_context_property">The property value
</contexts>
<data-config>
    <data-accessor>An implementation of a command executor
    <stop-on-first-result>True if the commands must be executed only on the first host
answering
    <host-group-config>
        <existing-group groupName="The_name_of_the_group_in_the_given_file">The location of a
host group configuration file
    <custom-group>
        <group-name>The group name
    <host-config>
        <connection-type>The connection type if (Only used for the RemoteCLICCommandExecutor)
        <host-list>The name or ip or the hosts
        <host-ip-range>
            <ip-start>Inclusive IP start

```

```

        <ip-end>Inclusive IP end
    </host-ip-range>
    <parameter name="The_value_key">
        <value>A list of values for the given key
    </parameter>
</host-config>
</custom-group>
</host-group-config>
<command-configuration>
    <os-grouping>The commands to execute on particular operation systems
    OR
    <group-name>The commands to execute on particular hosts groups
    <command-list>
        <command type="defined_or_custom">The command to execute
        <argument>The arguments of the command
        <result-must-be-parsed>True if the result contains wanted data
    OR
        <result-expected>A regex containing the result expected to ensure a right execution
        <result-have-header>Flag to tells if the result will start with an header for logging.
Header are terminated by two subsequent dashes
    </command-list>
</command-configuration>
    <parameter name="The_value_key">
        <value>A list of values for the given key
    </parameter>
    <pre-execution>
        <value>The pre execution command. Next values must contains the parameters
    </pre-execution>
    <post-execution>
        <value>The post execution command. Next values must contains the parameters
    </post-execution>
</data-config>
</configuration>

```


3

Executor

The executors have the responsibility to retrieve from the desired source the text that will be parsed. Here's the list of the different implementations with a configuration example. Each executors will be run once per defined execution context. An execution property can be referred almost anywhere in the executor configuration with `@propertyName`. Moreover, each context value can be obfuscated using the *APG* obfuscation tool and the reference will be using the real value at runtime. Using execution contexts is the preferred way of loops in the executor compared to the previous hosts list with (host).

1 CLICommandExecutor

This executor will execute a process via a command line instruction of the local host and retrieve its output. Since it's executed on the local host, the host configuration part have an other usage. The host list will be use as the parameters of the single command that can be executed. The key word (host) will be replace in the command by the content of the hosts list in order of appearance. Furthermore, the actual command (first element in the `<command>` tag) will be tested for executive in a path relative to the component using the library. It's up to the actual script to find the relative path of the parameters. The following example shows a configuration that will locally execute the script `/usr/bin/myScript.sh` three times on each polling with the parameters 192.168.1.20, 192.168.1.21 and 192.168.1.22. This executor support the stop-on-first-result flag. When set, each command will be run synchronously until the first one that produce more than 2MB of output or has an exit value of 0 in less than 30 seconds.

```
<data-config>
  <data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.local.
  CLICommandExecutor</data-accessor>
  <stop-on-first-result>false</stop-on-first-result>
  <host-group-config>
    <custom-group>
      <group-name>MyGroup</group-name>
      <host-config>
        <host-list>192.168.1.20</host-list>
        <host-list>192.168.1.21</host-list>
        <host-list>192.168.1.22</host-list>
      </host-config>
    </custom-group>
  </host-group-config>
  <command-configuration>
    <group-name>MyGroup</group-name>
    <command-list>
      <command type="custom">/usr/bin/myScript.sh (host)</command>
    </command-list>
  </command-configuration>
</data-config>
```

2 FileReader

This executor will read the content of a local file and retrieve its output. Therefore the command will be simply the path of the single file to read. Just like the local CLICommandExecutor, it's possible to use the key word (host) to replace some part of the path with the hosts list parameters. The following example shows a configuration that will locally read the content of the logs.txt file in the home directory of alpha and beta on each polling.

```
<data-config>
  <data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.local.
FileReader</data-accessor>
  <stop-on-first-result>false</stop-on-first-result>
  <host-group-config>
    <custom-group>
      <group-name>MyGroup</group-name>
      <host-config>
        <host-list>alpha</host-list>
        <host-list>beta</host-list>
      </host-config>
    </custom-group>
  </host-group-config>
  <command-configuration>
    <group-name>MyGroup</group-name>
    <command-list>
      <command type="custom">/home/(host)/logs.txt</command>
    </command-list>
  </command-configuration>

  <parameter name="filename-properties-extraction">
    <value>(.*)\.log</value>
    <value>devName/1</value>
  </parameter>

  <parameter name="file-regex">
    <value>.*\.log</value>
  </parameter>

  <parameter name="automatic-reading">
    <value>true</value>
  </parameter>

  <parameter name="file-close-action">
```

```

    <value>delete</value>
  </parameter>

  <parameter name="retrieve-files">
    <value>MY_SOURCE</value>
    <value>sftp</value>
    <value>600</value>
    <value>127.0.0.1</value>
    <value>username</value>
    <value>password</value>
    <value>/files/to/downloads</value>
    <value>60000</value>
    <value>120000</value>
    <value>1000</value>
    <value>>false</value>
  </parameter>

  <pre-execution>
    <value>./unzipAndChown.sh</value>
    <value>--group</value>
    <value>apg</value>
    <value>--user</value>
    <value>apg</value>
  </pre-execution>
</data-config>

```

Here's the list of parameters used by this executor:

1. filename-properties-extraction: Used when properties coming from the file name should be added at the root of the parsed tree schema. The first parameter values must be a valid Java regex with groups (Ex: `(\w+)(\d+).xml`) followed by values of properties to extract and their group separated by slashes (Ex: `devName/1`).
2. file-regex: When the given command is a folder, a file filter is needed to read all the desired files in the folder. The file-regex parameter is also needed when the executor is configured for an automatic reading. The filter must be a valid Java regex.
3. automatic-reading: Enable the parsing of files in the given folder as soon as they are created init. This parameters doesn't need any values but requires the parameter file-regex to be configured with a Java regex.
4. file-close-action: Enable an action to do when the file is closed after reading. The first value is the action type (delete, move, exec). The other values are action dependent. Delete doesn't requires any. Move requires the destination directory and an optional suffix to the moved file. Exec requires the command to execute and all the parameters in separate values.
5. retrieve-files: Enable the executor to retrieve the remote files automatically. The first value is an optional parameter to name the source of the file-retriever. For the ftp and sftp protocol, it will represent the hostname, and for the local protocol it will represent the source directory. The default value is SOURCE. For example, if you put FTPSOURCE as the parameter name, this will create a leaf named FTPSOURCE with the value 127.0.0.1 (example) in the output tree. The second value is the protocol type (local, ftp or sftp). The third value is the number secondes between each retrieval. The rest of the parameter are protocol dependent.

FTPClient

- (a) hostString: The host string in the format hostname:port.
- (b) username: The username.
- (c) password: The password.
- (d) remoteDirectory: The remote directory.
- (e) timeout: The data timeout (in milliseconds). Timeout used when reading from the data connection. 0 means no timeout.
- (f) connectTimeout: The connection timeout (in milliseconds). Timeout used for the connection to the underlying TCP socket. 0 means no timeout.
- (g) readTimeout: The read timeout (in milliseconds). Timeout used when opening the socket. 0 means no timeout.

- (h) `mtimeThreshold`: The time (in milliseconds) that the client will wait before downloading the file. To make sure that the file is not written anymore.
- (i) `mode`: The FTP connection mode (passive or active).
- (j) `deleteRemoteFile`: The flag that tells if the remote file shall be deleted after it has been retrieved. Set to `true` if you want to delete the file, otherwise set it to `false`.

SFTPClient

- (a) `hostString`: The host string in the format `hostname:port`.
- (b) `username`: The username.
- (c) `password`: The password.
- (d) `remoteDirectory`: The remote directory.
- (e) `connectTimeout`: The connection timeout (in milliseconds). Timeout used for the connection to the underlying TCP socket. 0 means no timeout.
- (f) `readTimeout`: The read timeout (in milliseconds). This is the amount of time the read operation can be blocked before the timeout occurred. 0 means no timeout.
- (g) `mtimeThreshold`: The time (in milliseconds) that the client will wait before downloading the file. To make sure that the file is not written anymore.
- (h) `deleteRemoteFile`: The flag that tells if the remote file shall be deleted after it has been retrieved. Set to `true` if you want to delete the file, otherwise set it to `false`.

Local

- (a) `sourceDirectory`: The local directory containing the files.
- (b) `mtimeThreshold`: The time (in milliseconds) that the client will wait before downloading the file. To make sure that the file is not written anymore.
- (c) `deleteRemoteFile`: The flag that tells if the source file shall be deleted after it has been copied. Set to `true` if you want to delete the file, otherwise set it to `false`.

3 RemoteCLICommandExecutor

This executor will connect to some distant hosts via a Telnet or SSH connection and run a series of commands to retrieve one or more inputs. The following example show a configuration that will create 2 custom group name LinuxGroup and SunOSGroup with their respective configuration. The defined command CPU_util will be executed on each host of these groups at each polling. The command is defined for each OS in the directory ./conf.

```
<data-config>
  <data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.remote.cli.
RemoteCLICommandExecutor</data-accessor>
  <stop-on-first-result>false</stop-on-first-result>
  <host-group-config>
    <custom-group>
      <group-name>LinuxGroup</group-name>
      <host-config>
        <connection-type>ssh</connection-type>
        <host-list>192.168.1.22</host-list>
        <parameter name="auth-method"><value>password</value></parameter>
        <parameter name="username"><value>myuser</value></parameter>
        <parameter name="password"><value>mypassword</value></parameter>
      </host-config>
    </custom-group>

    <custom-group>
      <group-name>SunOSGroup</group-name>
      <host-config>
        <connection-type>ssh</connection-type>
        <host-list>192.168.1.20</host-list>
        <host-list>192.168.1.21</host-list>
        <parameter name="auth-method"><value>password</value></parameter>
        <parameter name="username"><value>myuser2</value></parameter>
        <parameter name="password"><value>mypassword2</value></parameter>
      </host-config>
    </custom-group>
  </host-group-config>
  <command-configuration>
    <os-grouping>Linux</os-grouping>
    <os-grouping>SunOS</os-grouping>
    <command-list>
      <command type="defined">CPU_util</command>
      <result-must-be-parsed>true</result-must-be-parsed>
```

```
<result-have-header>true</result-have-header>
</command-list>
</command-configuration>
<parameter name="command-definition-directory"><value>./conf</value></parameter>
<parameter name="max-connection-threads"><value>4</value></parameter>
</data-config>
```

Here's the list of parameters used by this executor:

1. Host parameters

- (a) username: Mandatory parameter defining the username.
- (b) password: Mandatory for a telnet connection and most of the SSH connection type (can be optional for a public-key authentication).
- (c) auth-method: Mandatory for a SSH connection. This parameter specify which mode of connection that will be use. Possible values are:
 - i. password: Simple username/password connection.
 - ii. keyboard-interactive: Still use a username/password connection, but it will emulate an interaction of with the SSH server of a password asking in a command line.
 - iii. public-key: Will use a username and public encrypted key file (.pem) for authentication. The parameter pem-file is mandatory when using this authentication mode and the parameter password can be use if the file is encrypted.
- (d) pem-file: The path of the public key file for a SSH authentication. See [Public key file example](#) below.
- (e) hardcoded-os: If the OS polled don't have the uname command, the hardcoded-os must be defined.
- (f) prompt-char: If using a shell type connection (telnet of shell SSH), this parameter will overwrite the character or regex at the end of a prompt line. Default is \$.
- (g) shell-connection: Optional parameter for a SSH connection that will specify to use a shell type connection. Use this mode only if the regular mode is not working.
- (h) ip-source: Optional parameter for a SSH or Telnet connection that will add the desired IP source binding.

2. Executor parameters

- (a) max-connection-threads: The maximum number of threads for the parallel connections.

- (b) max-command-threads: The maximum number of threads for the commands processing on a single connection.
- (c) command-definition-directory: The directory containing all the files of OS command definition (Ex: Linux.xml, SunOs.xml, ...). Mandatory if at least one command is of type "defined"
- (d) index-keeper-file: The file containing the line counts of each host for an incremental polling.
- (e) previous-index-param: Defines the parameter to add with the previous line counts to the current command for an incremental polling.

3.1 Public key file example

Here's an example of the required format for a public key file:

```
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQDepsKLQpdcUI/cT4mhWsr/I+p4ZTBz9Jyt5V4LeONT5orG09aL
aQsPd+TQeresayMdgOLuCHwaO+HFwT03coQcmg+dTkqllwERqbhyk4Ft9EqaXjEE
2S0+yCk2W/Q7raqri2eXZDt7BGbp0lHdsy90nPumuYF9MjbEFRzf1ut9ywIBJQKB
...
svhwhGMRBi99ZFq8qJJRuf7Js11WbgJsEfkiTQAhcLBrAOAkPUg3Msj3hip2IJJL
7rdzKAZ4JGtF+D337QJATfic7N6EU+NgNH6SxZVScte0z/QZXZH+khi7/48rozPJ
OeyEK5rJwVyFnnAAXld2KaDSbXhhvJhCnmDs66smrA==
-----END RSA PRIVATE KEY-----
```

4 FtpFileGetter

This executor will retrieve the content of a single file on a distant FTP server. The following examples shows a configuration that will retrieve an ASCII file event.txt on the server 192.168.1.25.

```
<data-config>
  <data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.remote.ftp.
  FtpFileGetter</data-accessor>
  <stop-on-first-result>false</stop-on-first-result>
  <host-group-config>
    <custom-group>
      <group-name>MyGroup</group-name>
      <host-config>
        <host-list>192.168.1.25</host-list>
      </host-config>
    </custom-group>
```

```
</host-group-config>
<command-configuration>
  <group-name>MyGroup</group-name>
  <command-list>
    <command type="custom">dir/component/logs/event.txt</command>
  </command-list>
</command-configuration>
<parameter name="username"><value>myuser</value></parameter>
<parameter name="password"><value>mypassword</value></parameter>
<parameter name="connection-mode"><value>active</value></parameter>
<parameter name="file-type"><value>ascii</value></parameter>
<parameter name="data-timeout"><value>60</value></parameter>
<parameter name="connection-timeout"><value>20</value></parameter>
</data-config>
```

Here's the list of parameters used by this executor:

1. Host parameters

- (a) username: The username for authentication
- (b) password: The password for authentication
- (c) connection-mode: The connection mode (active or passive)
- (d) file-type: The file type (binary or ascii)
- (e) data-timeout: The data retrieving timeout
- (f) connection-timeout: The connection timeout

5 SOAPXmlRequestExecutor

This executor will retrieve the result of soap commands. If more than one command is given, the executor will run them sequentially until a non prerequisite one run successfully. The following example shows a configuration that will invoke SOAP requests on 192.168.1.25 with the help of a input file.

```
<data-config>
  <data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.remote.soap.
  SOAPXmlRequestExecutor</data-accessor>
  <host-group-config>
    <custom-group>
      <group-name>MyGroup</group-name>
    </custom-group>
  </host-group-config>
</data-config>
```

```

    <host-list>192.168.1.25</host-list>
    <parameter name="username"><value>username1</value></parameter>
    <parameter name="password"><value>password1</value></parameter>
  </host-config>
</custom-group>
</host-group-config>
<command-configuration>
  <group-name>MyGroup</group-name>
  <command-list>
    <command type="custom">http://(host):8080/xmlapi/invoke [P,FC]</command>
    <command type="custom">http://(host):8080/xmlapi/login [G,SC,CC]</command>
    <command type="custom">http://(host):8080/xmlapi/invoke </command>
  </command-list>
</command-configuration>
<parameter name="input-file"><value>conf/SAM5620/InterfaceStats.xml</value></parameter>
<parameter name="http-version"><value>HTTP/1.0</value></parameter>
<parameter name="disable-ssl-authentication"><value>>false</value></parameter>
<parameter name="keystore-file"><value>conf/keystore.jks</value></parameter>
<parameter name="keystore-password"><value>password1</value></parameter>
<parameter name="headers">
  <value>SOAPAction:"Some-URI"</value>
  <value>Accept:text/plain</value>
</parameter>
</data-config>

```

This example is a typical cookie session one. The first command and the last one are the same in term of request. If the first one fails, the login command will be executed to refresh the login cookie before invoking the desired one again. The flags between [] give some request specific behavior separated by commas. The available flags are:

- P or POST: Specify that the request is done with POST.
- G or GET: Specify that the request is done with GET.
- FC or FAIL.CONT: Specify that the next request must be done when the current one returns anything else that a 200 OK and will otherwise stop and return the response when success.
- SC or SUCCESS.CONT: Specify that the next request must be done when the current one returns a 200 OK and will otherwise stop the request chain
- PT or PASS: Specify that the request is a pass through so the next request will be done either way.
- CC or CLEAR.COOKIES: Specify that the request must clear the current cookies before executing it.

If no flag is defined, a request will be considered as a POST and will return the response on a 200 OK. If no POST or GET flag is defined, the request will automatically be a POST. As soon as one POST request is defined, it's mandatory to include an input-file.

The *username* and *password* are optional. If they are specified, the Executor will be able to do a SOAP request using BASIC and DIGEST authentication.

If the remote host is not challenging correctly to the authentication, setting *preemptive-authentication* to true should resolve the problem.

The HTTP protocol version can be specified if the default version 1.1 is not supported by the remote server. To do so, use the parameter *http-version* with any of those values: HTTP/0.9, HTTP/1.0 or HTTP/1.1

The headers can be used to manually force some HTTP headers in the SOAP request, in case they are needed.

5.1 SSL

It is possible to send a soap command via SSL (with encryption). If the host is already trusted in the default Java keystore, then you don't have to add more parameters. Supported SSL protocol are : SSL 3.0 and TLS 1.0.

If your certificate is not trusted, then you can add it in the default Java keystore. Or you can create a new keystore, add the certificate to that keystore and pass that keystore to the SOAPXmlRequestExecutor with the parameters *keystore-file* and *keystore-password*. The advantage of that last option is if there is an update on your Java, then your keystore will not be overwritten.

In last resort, you can put the parameter *disable-ssl-authentication* to *true*. This will accept every self-signed certificates. If this parameter equal true, then the *keystore-file* parameter will have no effect. **For security reason, this parameter must be used with caution.**

Some useful keytool command (the keytool tool is located in the <Java>/bin directory). The following commands should be enough for your needs but if you want more information on the tool and the keystore file, refer to http://download.oracle.com/docs/cd/E17476_01/javase/1.5.0/docs/tooldocs/windows/keytool.html.

Create a new keystore file with the name *keystore.jks* :

```
keytool -genkey -keyalg RSA -keystore keystore.jks
```

Add a certificate to the keystore :

```
keytool -import -trustcacerts -alias MyAlias -file MyCertificate.cer -keystore keystore.jks
```

Add a certificate to the default java keystore :

```
keytool -import -trustcacerts -alias MyAlias -file MyCertificate.cer -keystore ../Java/lib/  
security/cacerts
```

Remove a certificate from a keystore :

```
keytool -delete -alias MyAlias -keystore keystore.jks
```

List the alias of all the certificates in a keystore :

```
keytool -list -v -keystore keystore.jks | grep Alias
```

6 CASXmlRequestExecutor

This executor will retrieve the result of soap command over a CAS authenticated server. This executor will execute a single command and will take care of the login request automatically. The following example shows a configuration that will invoke SOAP requests on 192.168.1.25.

```
<data-config>
  <data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.remote.soap.
CASXmlRequestExecutor</data-accessor>
  <host-group-config>
    <custom-group>
      <group-name>MyGroup</group-name>
      <host-config>
        <host-list>192.168.1.25</host-list>
        <parameter name="username"><value>username1</value></parameter>
        <parameter name="password"><value>password1</value></parameter>
      </host-config>
    </custom-group>
  </host-group-config>
  <command-configuration>
    <group-name>MyGroup</group-name>
    <command-list>
      <command type="custom">http://(host):8080/xmlapi/invoke [G]</command>
    </command-list>
  </command-configuration>
  <parameter name="redirection-header-key"><value>Location</value></parameter>
</data-config>
```

The redirection-header-key parameter is the optional header key of the post login redirection. By default it's set to Location and this value should be fine in most cases.

By default, a CAS server is authenticating strictly via a POST of the login form. Normally, the form requires hidden fields to be passed as parameters. If not all forms hidden fields are necessary for a successful login, they can be restricted by listing them in the desired-hidden-fields parameter. If the targeted CAS server has a wrapper allowing a basic authentication via a GET method, it's possible to do so by using the following executor parameter:

```
<parameter name="authentication-with-get"><value>USE-GET=TRUE</value></parameter>
```

As soon as the authentication-with-get parameter is present, the switch will be made. The values are optional headers that will be passed in the login request.

7 Cisco UCS

This executor access data by reading the response from a Cisco UCS web service. For each host specified in `host` tags, a custom port can be appended as such: `IP:port`. If no port is appended, the default http port (80) will be used. The executor first performs authentication, then performs the requests if authentication was successful. The user must provide an authentication request file, which is specified using the "authorization-request-file" parameter (see "Parameters" section below). Before each polling, the UCS adapter performs the authentication (using the provided request file) and feeds the retrieved authentication cookie into the request at the location specified by the "insert-cookie-as" parameter. The request is then sent to the UCS, and the response is parsed by the collector.

```
<data-config>
  <data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.cisco.
UcsWebServiceExecutor</data-accessor>
  <stop-on-first-result>>false</stop-on-first-result>
  <host-group-config>
    <custom-group>
      <group-name>MyGroup</group-name>
      <host-config>
        <host-list>UCS_HOST</host-list>
      </host-config>
    </custom-group>
  </host-group-config>

  <command-configuration>
    <group-name>MyGroup</group-name>
    <command-list>
      <command type="custom">http://(host):8080/nuova</command>
    </command-list>
  </command-configuration>

  <parameter name="data-retry"><value>2</value></parameter>
  <parameter name="charset"><value>UTF-8</value></parameter>
  <parameter name="data-timeout"><value>90</value></parameter>
  <parameter name="connection-timeout"><value>180</value></parameter>
  <parameter name="force-close-connections"><value>true</value></parameter>
  <parameter name="insert-cookie-as"><value>/configScope/@cookie</value><value>/aaaLogout/
@cookie;/aaaLogout/@inCookie</value></parameter>
  <parameter name="soap-action"><value>http://www.watch4net.com/soap</value></parameter>
  <parameter name="input-file"><value>conf/UCS/request-equipment.xml</value><value>conf/UCS/
request-logout.xml</value></parameter>
  <parameter name="authorization-request-file"><value>conf/UCS/request-auth.xml</value></
```

```
parameter>
  <parameter name="authorization-request-cookie-path"><value>/aaaLogin/@outCookie</value></
parameter>
  <parameter name="headers">
    <value>SOAPAction:http://www.watch4net.com/soap</value>
    <value>AnyHeader:AnyValue</value>
  </parameter>
</data-config>
```

Here's the list of parameters used by this executor:

- data-retry: Number of retries to access data. If no data is returned after the specified number of retries, the command executor will give up for the current polling cycle. In case a fatal error occurs (no route to host, for example), no retry will be attempted. Default is 3
- charset: Sets the encoding to use for both reading the request file and sending the request to the server. Default is utf-8
- data-timeout: Time before a request fails if no response is received (in seconds). Default is 60
- connection-timeout: Time before a request fails if no connection is established (in seconds). Default is 120
- force-close-connections: If set to true, HTTP client is forced to close its connections between each polling. Default is false
- input-file: Mandatory parameter for the files that contain the requests that will be sent to the Cisco UCS Web Service. The first *value* tag contains the actual request file. The second *value* file contains the logout request file.
- soap-action: This parameter still works, but it is deprecated. Use *headers* instead. It will set the SOAPAction parameter in the SOAP request.
- authorization-request-file: Mandatory parameter containing the path to the file that will be used to send the authentication request.
- authorization-request-cookie-path: Mandatory parameter containing the XPath expression that defines how to retrieve the authentication cookie from the authentication response. Usually, this value should be: /aaaLogin/@outCookie
- insert-cookie-as: Mandatory parameter containing the XPath expressions that defines where the authentication cookie should be inserted in the requests. There needs to be one *value* tag for each file specified in the *session-files* parameter. Multiple locations in the same file can be separated by ;.
- headers: Defines the HTTP headers to use in the request. Multiple *value* tags can be used here. Each *value* tag must contain one header, formatted as such:*headerName:headerValue*.

8 Chained Web Service Requests Executor

This executor will retrieve the result of a soap command and gives the opportunity to use this answer to dynamically create a new request based on it. The following example shows a configuration that will invoke a SOAP request on 192.168.1.1 with the help of a input file.

```
<data-config>
  <data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.remote.soap
  .ChainedWebServiceRequestsExecutor</data-accessor>
  <stop-on-first-result>>false</stop-on-first-result>

  <host-group-config>
    <custom-group>
      <group-name>MyGroup</group-name>
      <host-config>
        <host-list>192.168.1.1:8080</host-list>
      </host-config>
    </custom-group>
  </host-group-config>

  <command-configuration>
    <group-name>MyGroup</group-name>
    <command-list>
      <command type="custom"></command>
    </command-list>
  </command-configuration>

  <parameter name="connect-type"><value>any</value></parameter>
  <parameter name="data-retry"><value>2</value></parameter>
  <parameter name="data-timeout"><value>90</value></parameter>
  <parameter name="connection-timeout"><value>180</value></parameter>
  <parameter name="charset"><value>UTF-8</value></parameter>
  <parameter name="headers">
    <value>SOAPAction:http://www.watch4net.com/soap</value>
    <value>AnyHeader:AnyValue</value>
  </parameter>
  <parameter name="force-close-connections"><value>true</value></parameter>
  <parameter name="input-file"><value>conf/q.xml</value></parameter>
  <!-- The first request to execute -->
  <parameter name="request-1">
    <!-- The file to send to the web service -->
```

```
<value>conf/q.xml</value>
<!-- The XSL file to use to transform the response into the second request -->
<value>conf/q.xsl</value>
<!-- The error handling mode (either strict or lenient) -->
<value>strict</value>
<!-- The URL of the web service -->
<value>http://10.128.2.144:58080/APG-WS/wsapi/db</value>
<!-- The specific headers to include with this header -->
<value>MyHeader1:Value1;MyHeader2:Value2</value>
<!-- The optional content type -->
<value>xml or json</value>
</parameter>
<!-- The second request to execute -->
<parameter name="request-2">
  <value>tmp</value>
  <value>conf/file.xsl</value>
  <value>lenient</value>
  <value>http://10.128.2.144:58080/APG-WS/wsapi/db</value>
  <value>MyHeader1:Value1;MyHeader2:Value2</value>
</parameter>
<parameter name="output-folder"><value>out</value></parameter>
</data-config>
```

Here's the list of parameters used by this executor:

- host-group-config : see Cisco UCS
- command-configuration : this configuration part is mandatory but not used.
- parameter connect-type : see SOAPXmlRequestExecutor
- parameter data-retry : see Cisco UCS
- parameter data-timeout : see Cisco UCS
- parameter connection-timeout : see Cisco UCS
- parameter charset : see Cisco UCS
- parameter headers : see SOAPXmlRequestExecutor
- parameter force-close-connections : see Cisco UCS
- parameter input-file : see SOAPXmlRequestExecutor
- parameter disable-ssl-authentication : see SOAPXmlRequestExecutor
- parameter keystore-file : see SOAPXmlRequestExecutor

- parameter keystore-password : see SOAPXmlRequestExecutor
- parameter disable-ssl-authentication : see SOAPXmlRequestExecutor
- parameter header-separator : header separator for specific headers values. (optional, default is ;)
- parameter executor-type : Any other value than CAS (case insensitive) will result as the standard SOAPXmlRequestExecutor being used
- parameter dynamic-url : Must be used when all the transformers are generating fragments of URL (one per line) that will be passed in the next request and replaced for each occurrence of ~DYNAMIC_URL~
- parameter request-n (request-1, request-2, ...) : There can be multiple chained requests. The XSL writes the output file for the next request.
 - value : path to the file to send to the web service. The first request must be a file. All other requests must be a directory (all data in directory is deleted after usage).
 - value : path to the XSL file to transform the response into a new request. The XSL must create the new request and put it in the required folder for the next action (request, close or output). See <http://www.w3.org/TR/xslt20/> to learn about XSL. XSL 2.0 is required to be able to write files in directories.
 - value : error handling mode (either strict or lenient)
 - value : URL of the web service
 - value : specific headers, separated by header separator
 - value : the optional content type (xml or json)
- parameter close-n (close-1, close-2, ...) : these are used only when the sessions opened with the request need to be explicitly closed. The first close (close-1) will be called after all requests. They need as input the same values (in the same order) from the request. The XSL path value can be empty but the value tag must still be there.
- parameter output-folder : path to the output folder. The collector will extract properties and values from every files in this directory.

4

Parsing Engine

The parser handle the output and filter it to give a tree ordering logically. Since the implementation of the RegexTextParser, the CVSParser and NavicliOutputParser are still available but have been declared deprecated. The RegexTextParser is an implementation of the IncrementalParser that emphasize the use of regexes to parse a text input. With proper configuration, it should provide enough flexibility to parse a great any text input.

1 Performance Limitations

Since the RegexTextParser has been made to fit most of the situations, a matter of performance must be included for complex sub delimiter conditions. Be aware than even if the RegexTextParser can successfully parse the desired input, it may be wise to create a particular implementation of the TextParser if performance problems are encountered. However, a right configuration will greatly optimize the parser. Always give a list of delimiters without exception or condition before a list of delimiters with exception and finally a list of delimiters with condition if possible. Performance can also be improved if you give a regex that fit most of you sub delimiter instead of having more SubDelimiterConfig.

2 Tree Type Specificity

2.1 Single Line Tree

The single line tree use most of the tag in the configuration. The only ones skipped are the group definition in the leaves of the tree schema, the parameter list and the regex group configuration. A detailed example is available in the next section.

2.2 Table Like Tree

No tree schema is necessary since the node will be represented by the row number and the leaves will be each columns of the tables. However, a parameter is needed for every table that must be parsed like this:

```
<parameter name="table-params">
  <value>DEVICE_</value>
  <value>Time,Device,Path ID,Target ID,LUN,Product ID.*</value>
  <value></value>
</parameter>
```

A parameter table-params is needed for every tables. The first value describe the title of the node for each row of this table. The title will be followed by the absolute number of the line where the rows have been found. The second value is a regex that describe the line containing the titles of the columns of the desired table. The last value describe the line representing the first line after the end of the table.

2.3 Regex Group Tree

The regex group tree use a set of complex regex to extract the data without using delimiters. Thus, the master and sub delimiters must be omitted and a regex group configuration must be define like this:

```
<regex-group-config group="-1">
  <regex multiline="true" name="main">(S[0-9]{3}-[0-9]{6}|S[0-9]{2}[A-Z]-[0-9]{6})(.*\n.*)</regex>
  <sub-regex group="2">
    <regex name="sub">([A-Z])\s+(RMV LINE|TRC|REPT)(.*)</regex>
    <sub-regex group="3">
      <regex name="inen">*INEN=(.*)\s.*</regex>
    </sub-regex>
  </sub-regex>
</regex-group-config>
```

The group represent the index of the group resulting from the extraction of the parent regex where the sub regex must be applied. A regex group is defined by parenthesis so in the above example, the first regex have two groups. For example, the second sub regex will be applied on the result of the `(.*\n.*)` group of the first regex since the group is 2. The multi line flag will simply tell if the regex must be applied to a multi line input or not. The unique name of each regex is necessary to format the tree in a logical way. Here's an example on how to define the tree schema for a regex group tree:

```
<tree-schema>
  <name>Log_</name>
  <leaf>
    <name>ID</name>
    <group>main-1</group>
  </leaf>
  <leaf>
    <name>Level</name>
    <group>sub-1</group>
  </leaf>
  <leaf>
    <name>Property</name>
    <group>sub-2</group>
  </leaf>
  <leaf>
    <name>Inen Value</name>
    <group>inen-1</group>
  </leaf>
</tree-schema>
```

The name of each node is hardcoded and will be followed by a random number and will be created when needed. The name of the leaves are hardcoded and will be created when for every match of the corresponding regex group. For example, a Inen value leaf will be created for every match of the group 1 of the "inen" regex.

3 Single Line Tree Example

Here's a situation that will provide an example for most of the options available. Let's assume we want to parse the result of an ifconfig command on a Linux machine to retrieve some information on each interface with their transmitted and received bytes. Let's take this output:

```
eth0 Link encap:Ethernet HWaddr 00:0c:29:f8:79:6e
      inet addr:192.168.1.20 Bcast:192.168.1.255 Mask:255.255.255.0
```

```

inet6 addr: fe80::20c:29ff:fe8:796e/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:20829 errors:0 dropped:0 overruns:0 frame:0
TX packets:3173 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:1895232 (1.8 MB) TX bytes:2333651 (2.2 MB)
Interrupt:17 Base address:0x2000

lo  Link encap:Local Loopback
     inet addr:127.0.0.1 Mask:255.0.0.0
     inet6 addr: ::1/128 Scope:Host
     UP LOOPBACK RUNNING MTU:16436 Metric:1
     RX packets:34410 errors:0 dropped:0 overruns:0 frame:0
     TX packets:34410 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:0
     RX bytes:13650915 (13.0 MB) TX bytes:13650915 (13.0 MB)

```

3.1 Configuration

In this example, we could be interested in the parameters mac address, IPv4 and v6 addresses with the transfer value for each type of packets. Here's the complete configuration for this example and please notes that each section will be describe further down and since it uses regexes, it could have been done in a lot of different ways and maybe more optimal than this example.

```

<parser-config>
  <parser-class>com.watch4net.apg.v2.collector.plugins.outputparsing.parsing.generictext.
  RegexTextParser</parser-class>
  <tree-type>single-line-tree</tree-type>
  <incremental-parser>false</incremental-parser>
  <sub-delimiter>
    <delimiter> </delimiter>
    <exception-list>inet addr</exception-list>
    <exception-list>inet6 addr</exception-list>
    <exception-list>RX bytes</exception-list>
    <exception-list>TX bytes</exception-list>
  </sub-delimiter>

  <sub-delimiter>
    <delimiter>:</delimiter>

```

```

<condition-list>
  <prefix depth="3">[a-zA-Z]+</prefix>
  <suffix depth="1">[0-9] | [A-Z]</suffix>
  <matches>true</matches>
</condition-list>
</sub-delimiter>

<tree-schema>
  <name>eth.|lo.*</name>
  <leaf><name>Link encap|HWaddr|inet addr|Bcast|Mask|inet6 addr|Scope|MTU|Metric</name></leaf>
>
  <leaf><name>collisions|txqueuelen|Interrupt|Base address|RX bytes|TX bytes</name></leaf>
  <node>
    <name>RX|TX</name>
    <leaf><name>packets|errors|dropped|overruns|carrier|frame</name></leaf>
  </node>
</tree-schema>
</parser-config>

```

3.2 Explanation

```

<parser-class>com.watch4net.apg.v2.collector.plugins.outputparsing.parsing.generictext.
RegexTextParser</parser-class>
<tree-type>single-line-tree</tree-type>
<incremental-parser>false</incremental-parser>

```

This part of the configuration specifies the class implementing TextParser used and it's of course the RegexTextParser class. The next line tells that the input will be presented in the a single line input. The tree builder will assume that each line of the input will contain only one node, leaf or value and values are always written after their leaves after having the input be split by the delimiters. The last parameter specifies that the input data will contain complete datasets so the parser doesn't have to look for incomplete trees and keep them until the next input to complete them.

```

<sub-delimiter>
  <delimiter> </delimiter>
  <exception-list>inet addr</exception-list>
  <exception-list>inet6 addr</exception-list>
  <exception-list>RX bytes</exception-list>
  <exception-list>TX bytes</exception-list>

```

```
</sub-delimiter>
```

Since no master delimiter is provided before the sub delimiters, the tree builder will assumes that the master delimiter is the carriage return. This sub delimiter tells to the builder that each whitespace must be split with the exception of the expressions inet addr, inet6 addr, RX bytes and TX bytes. This is the only input that cannot be match with a regex.

```
<sub-delimiter>
  <delimiter>:</delimiter>
  <condition-list>
    <prefix depth="3">[a-zA-Z]+</prefix>
    <suffix depth="1">[0-9] | [A-Z]</suffix>
    <matches>true</matches>
  </condition-list>
</sub-delimiter>
```

The next sub delimiter is the colon and it's used mainly to separate the leaf key from it's value. The condition provides a great flexibility to split the input only when necessary. In this example, the colon will be split when the 3 characters before the colon are letters and the next character is a number or an uppercase letter.

```
<tree-schema>
  <name>eth.|lo.*</name>
  <leaf><name>Link encap|HWaddr|inet addr|Bcast|Mask|inet6 addr|Scope|MTU|Metric</name></leaf>
  <leaf><name>collisions|txqueuelen|Interrupt|Base address|RX bytes|TX bytes</name></leaf>
  <node>
    <name>RX|TX</name>
    <leaf><name>packets|errors|dropped|overruns|carrier|frame</name></leaf>
  </node>
</tree-schema>
```

This part will determines how the tree will be displayed. The first name is the root node of the tree. In this case, it will start with either eth or lo. Next is the leaf under the root nodes, it is the key for getting the value. After there is another sub-node named either RX or TX, with other leaves under it.

3.3 Output Tree

This section describe the tree expected as a result of the parsing. The above example will produce this tree at the output:

```
HOST
-eth0
  -HWaddr: 00:0c:29:f8:79:6e
  -inet addr: 192.168.1.20
  -inet6 addr: fe80::20c:29ff:fe8:796e/64

-RX
  -packets: 20829
  -errors: 0
  -dropped: 0
  -overruns: 0
  -frame: 0

-TX
  -packets: 3173
  -errors: 0
  -dropped: 0
  -overruns: 0
  -carrier: 0

-collisions: 0
-txqueuelen: 1000
-RX bytes: 1895232
-TX bytes: 2333651
-Interrupt: 17

-lo
  -inet addr: 127.0.0.1
  -inet6 addr: ::1/128

-RX
  -packets: 34410
  -errors: 0
  -dropped: 0
  -overruns: 0
  -frame: 0

-TX
  -packets: 34410
  -errors: 0
  -dropped: 0
```

```
-overruns: 0
-carrier: 0

-collisions: 0
-txqueuelen:0
-RX bytes: 13650915
-TX bytes: 13650915
```

4 Table Like Tree Example

Here's a situation that will provide an simple example for a parsing of a table. Let's assume we want to parse the result of a formatted ifconfig command on a Linux machine to retrieve some information on each interface with their transmitted and received bytes. Let's take this output:

```
Name      Mac address      IP address  RX bytes  TX bytes
eth0      00:0c:29:f8:79:6e 192.168.1.20 1895232  2333651
lo        N/A              127.0.0.1   34410    34410
Data retrieved in 0.025sec
```

4.1 Configuration

Like the previous example, we could be interested in the parameters mac address, IPv4 address with the received and transmitted bytes. Here's the complete configuration for this example.

```
<parser-config>
  <parser-class>com.watch4net.apg.v2.collector.plugins.outputparsing.parsing.generictext.
  RegexTextParser</parser-class>
  <tree-type>table-like-tree</tree-type>
  <incremental-parser>false</incremental-parser>
  <sub-delimiter>
    <delimiter> </delimiter>
    <exception-list>Mac address</exception-list>
    <exception-list>IP address</exception-list>
    <exception-list>RX bytes</exception-list>
    <exception-list>TX bytes</exception-list>
  </sub-delimiter>
  <parameter name="table-params">
    <value>IFCONFIG_</value>
    <value>^Name\s+Mac address.+</value>
```

```
<value></value>
</parameter>
</parser-config>
```

4.2 Explanation

```
<parameter name="table-params">
  <value>IFCONFIG_</value>
  <value>^Name\s+Mac address.+</value>
  <value></value>
</parameter>
```

In a table like tree output, there's no need to specify an output tree since the input is already logically ordered. The output will be made of a node for each table line found in the input containing a leaf for each column. The only configuration needed is a parameter table-params for each different tables format in the input. The first value is the global name of the resulting tree row that will be followed by the row number. The next value is a regex representing the row containing the column titles. Finally, the last row is an optional regex that describe the first line following the end of the table if available.

4.3 Output Tree

This section describe the tree expected as a result of the parsing. The above example will produce this tree at the output:

```
HOST
-IFCONFIG_0
  -Name: eth0
  -Mac address: 00:0c:29:f8:79:6e
  -IP address: 192.168.1.20
  -RX bytes: 1895232
  -TX bytes: 2333651

-IFCONFIG_1
  -Name: lo
  -Mac address: N/A
  -IP address: 127.0.0.1
  -RX bytes: 34410
  -TX bytes: 34410
```

5 Regex Group Tree Example

The regex group tree offer a more versatile parsing opportunity. The only prerequisite is a good understanding of regexes. It's recommended to use the other trees when possible since they easier to configure if the input change in the future. When the keys of the values aren't in the input, the regex group tree is the solution. Let's assume we want to retrieve some info on a server pool class configuration. Let's take this output:

service-policy: RESTRICTED_ENV										
Class	VIP	Prot	Port	VLAN	State	Conns	Hit Count	Conns	Drop	
REST_192.168.1.25-HTTP	192.168.1.25	tcp	eq 80	23	IN-SRVC	142	6069204		21	
REST_192.168.1.26-HTTP	192.168.1.26	tcp	eq 80	23	IN-SRVC	0	41996		0	
REST_192.168.1.27-HTTP	192.168.1.27	tcp	eq 80	23	IN-SRVC	0	10006		0	

5.1 Configuration

If we were only interested in the columns of the table, we could easily used the table like tree but in this example, we also need to retrieve the service policy of a classes table and it would be hard to retrieve the port number without the eq.

```
<parser-config>
  <parser-class>com.watch4net.apg.v2.collector.plugins.outputparsing.parsing.generictext.
  RegexTextParser</parser-class>
  <tree-type>regex-group-tree</tree-type>
  <tree-schema>
    <name>DETAIL_</name>
    <leaf>
      <name>policy</name>
      <group>policy-1</group>
    </leaf>
    <node>
      <name>CLASS_</name>
      <leaf>
        <name>class</name>
        <group>classCount-1</group>
      </leaf>
      <leaf>
        <name>hitCount</name>
        <group>classCount-8</group>
      </leaf>
    </node>
  </tree-schema>
</parser-config>
```

```

    <leaf>
      <name>connsDrop</name>
      <group>classCount-9</group>
    </leaf>
  </node>
</tree-schema>

<regex-group-config group="-1">
  <regex name="main">( ?s)service-policy(.+?)(?=(service-policy|\$))</regex>
  <sub-regex group="1">
    <regex name="policy">( ?s): ([\w|-]+\r?\n(.+)</regex>
    <sub-regex group="2">
      <regex name="tiles">( ?s)Class\s+VIP.+Conns Drop\r?\n(.+)</regex>
      <sub-regex group="1">
        <regex name="classCount">([\w|-]+\s+([\d|\.|.]+\s+(\w+)\s+eq(\d+)\s+(\d+)\s+([\w|-]+\s+
s+(\d+)\s+(\d+)\s+(\d+)\r?\n</regex>
      </sub-regex>
    </sub-regex>
  </sub-regex>
</regex-group-config>
</parser-config>

```

5.2 Explanation

```

<tree-schema>
  <name>DETAIL_</name>
  <leaf>
    <name>policy</name>
    <group>policy-1</group>
  </leaf>
  <node>
    <name>CLASS_</name>
    <leaf>
      <name>class</name>
      <group>classCount-1</group>
    </leaf>
    <leaf>
      <name>hitCount</name>
      <group>classCount-8</group>
    </leaf>
  </node>
</tree-schema>

```

```
<leaf>
  <name>connsDrop</name>
  <group>classCount-9</group>
</leaf>
</node>
</tree-schema>
```

This section describe the resulting for this input. The regex group tree is the only tree that use the tag group in his leaves. In this kind of tree, the node name are hardcoded and followed an incrementing index while the leaved name will simply be hardcoded. The group specify in which regex the value is contained and in which group in that regex. For example, the hitCount values are located in the 8th group of the regex classCount.

```
<regex-group-config group="-1">
  <regex name="main">(s)service-policy(.+)(?=(service-policy|\\$))</regex>
  <sub-regex group="1">
    <regex name="policy">(s): ([w|-])\r?\n(.+)</regex>
    <sub-regex group="2">
      <regex name="titles">(s)Class\s+VIP.+Conns Drop\r?\n(.+)</regex>
      <sub-regex group="1">
        <regex name="classCount">([w|-])\s+([d|.])\s+(\w+)\s+eq(d+)\s+(d+)\s+([w|-])\s+
+ (d+)\s+(d+)\s+(d+)\r?\n</regex>
      </sub-regex>
    </sub-regex>
  </sub-regex>
</regex-group-config>
```

This section is the most complex one. The regex group tree is the only one to use the tag regex-group-config. The first group index of this configuration is irrelevant since no regex have been applied. This example is composed of 4 regexes with 2 of them containing values in the tree (policy and classCount). The regex main is simply there to extract each fragment of text delimited by the word service-policy or when reaching the end of the file while the regex titles is simply there to remove the titles row.

5.3 Output Tree

This section describe the tree expected as a result of the parsing. The above example will produce this tree at the output:

```
HOST
```

```
-DETAIL_0
  -policy: RESTRICTED_ENV

  -CLASS_1
    -class: REST_192.168.1.25-HTTP
    -hitCount: 6069204
    -connsDrop: 21

  -CLASS_2
    -class: REST_192.168.1.26-HTTP
    -hitCount: 41996
    -connsDrop: 0

  -CLASS_3
    -class: REST_192.168.1.27-HTTP
    -hitCount: 10006
    -connsDrop: 0
```


5

Data Finalizer

The finalizer will take each raw value and check if the properties must be refreshed before sending them to the next component.