



EMC® Watch4net®

Version 6.3u2

XML Collector Administration Guide

P/N: pending

Rev : 46393

EMC²

Copyright ©2008 - 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
Intended Audience	iii
Prerequisites	iii
Where to get help	v
1 Product Overview	1
2 Architectural Overview	3
3 XML Sources	7
1 Successive Requests to a Web Service	8
1.1 Description	8
1.2 Java class	8
1.3 Parameters	8
1.4 Example	10
2 Cisco UCS Example	13
2.1 Example	13
4 XML Jobs	15
1 Xalan	15
1.1 Description	15
1.2 Java class	15
2 Saxon	16
2.1 Description	16
2.2 Java class	16
3 Nux	16
3.1 Description	16
3.2 Java class	16
4 JsonToSaxon	16
4.1 Description	16

4.2	Java class	16
5	JsonToNux	17
5.1	Description	17
5.2	Java class	17
5	Configuration	19
1	Collector Manager configuration	19
2	Collector configuration	20
6	Things to do after configuration	25
7	Troubleshooting	27
1	The collector won't start	27
2	The collector started successfully, but no data is showing up in APG	27
8	Appendix A: Configuration Example	29
9	Appendix B: Timestamps	33
1	Definitions	34
2	Examples	34

Preface

The purpose of this guide is to provide information about the XML Collector for APG. It contains all the necessary information to configure and troubleshoot the collector.

Intended Audience

This guide is intended to be read by any *APG* administrator who will configure the XML Collector to import XML data sources into APG.

Prerequisites

It is assumed that the *APG* application has been properly installed and configured on the server. It is also assumed that the XML Collector has been installed properly using the module manager. In order to be able to correctly configure the XML Collector, it is expected that the user understands the XPath notation. For good XPath explanations and tutorials, please refer to the XPath section of W3Schools' website (<http://www.w3schools.com/xpath/default.asp>).

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.

Product Overview

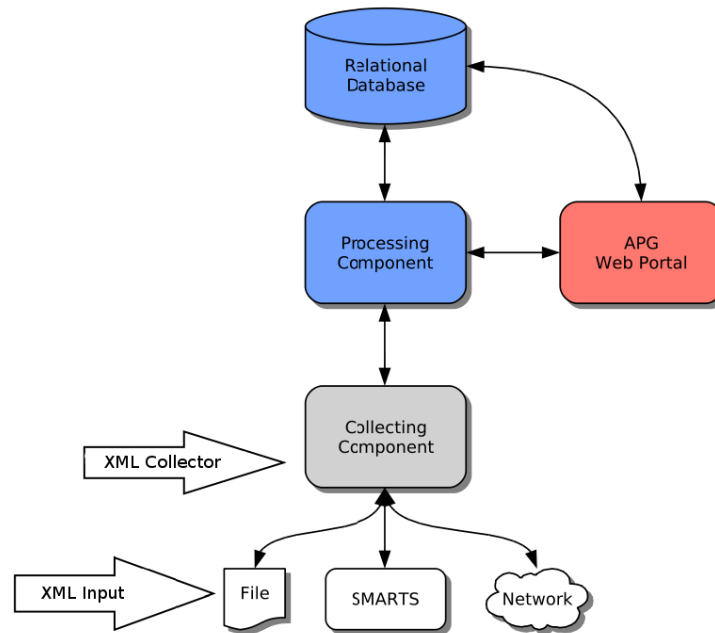
The XML Collector is a generic module for APG that allows data to be imported from any XML data source and pushed into APG. It relies on a proprietary library to acquire data from different sources. Currently supported sources are:

- Local xml files
- Remote xml files (via FTP)
- Local shell command execution
- Remote command execution (via SSH)
- SOAP Request/Response

2

Architectural Overview

The XML Collector is built to integrate seamlessly into the APG structure. The following diagram illustrates APG's global architecture and how collectors can be attached to it.



The XML Collector is composed of three main modules: the collector itself, the XML engine and the XML jobs. Each module's main responsibilities is defined as follows:

Collector Module

- Loads its configuration.
- Creates a thread pool to allow multi-threading for different collecting configurations
- Creates an XML engine for every *different* collecting configuration.
- Starts every XML engine.
- Retrieves data when parsing tasks are done.
- Forwards data to the APG Backend.

XML Converter

- Creates a thread pool to allow multi-threading on different hosts.
- Creates a XML job for every different host.
- Starts all XML jobs and waits for returned data.
- Returns APG values to the collector.

XML Job A XML job is an implementation of the task that uses XPath expressions to parse the input source and convert its content into APG values. XML/XPath libraries all have their strengths and weaknesses, which is why the XML collector is bundled with more than one implementation. Selecting the right implementation for the type of document that will be parsed will help achieving better performance and reliability. Although they use different underlying libraries, each parsing job must perform the following operations (for more information on the different implementations, see [section 4](#))

- Parses the input XML document.
- Retrieves needed information from the input.
- Converts the parsed information into APG values.
- Returns APG values to the XML engine.

XML Sources

XML data can be acquired in many different ways. The way data needs to be accessed can be specified in the XML Collector configuration file (see section 2) by setting the `<data-accessor>` value. Each different data access mechanism can be configured in a specific way by setting the correct parameters (using one or more `<parameter>` tags). Each access mechanism are coming from the APG Text Parsing Library. Please refer to the APG-Text-Parsing-Library.pdf documentation available with the APG Text Collector in the Executor section.

Each of these executor are available in the XML Collector. Simply transform the configuration example to fit the current collector configuration format. All the executor and host parameters must be replicate in a `<parameter>` tag under the tag `<data>`. The `<input>` tag from the XML Collector is normally replicated in the `<command>` tag of the Text Parsing Library.

Furthermore, the XML Collector use an optional custom parameter *connect-type* for every executor:

- all: If set to *all*, the command will be executed on all hosts listed inside `<host>` tags.
- any: If set to *any*, the command executor will try to execute the command on the listed hosts one at a time, and stop as soon as the command is successful on one of them.

1 Successive Requests to a Web Service

1.1 Description

Data is accessed by sending multiple requests to a web service. The response to each request must be transformed using an XSL transformation. The result of the transformation is then sent to the web service as the next request, and so on. 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.

1.2 Java class

`com.watch4net.apg.v2.collector.plugins.outputparsing.executing.remote.
soap.ChainedWebServiceRequestsExecutor`

1.3 Parameters

- connect-type (optional, default: all)
 - all: If set to all, the command will be executed on all hosts listed inside `<host>` tags.
 - any: If set to any, the command executor will try to execute the command on the listed hosts one at a time, and stop as soon as the command is successful on one of them.
- input-file (required)
 - The file containing the definition of the SOAP request to execute.
- data-retry (optional, default: 3)
 - 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.
- data-timeout (optional, default: 60)
 - Time before a request fails if no response is received (in seconds).
- connection-timeout (optional, default: 120)
 - Time before a request fails if no connection is established (in seconds).

- charset (optional, default: utf-8)
 - Sets the encoding to use for both reading the request file and sending the request to the server.
- soap-action (optional, default: No action is set). This parameter still works, but it is deprecated. Use *headers* instead.
 - Sets the SOAPAction parameter in the SOAP request.
- headers (optional, default: none)
 - Defines the HTTP headers to use in all requests. Multiple *value* tags can be used here. Each *value* tag must contain one header, formatted as such: *headerName:headerValue*.
- force-close-connections (optional, default: false).
 - If set to true, HTTP client is forced to close its connections between each polling.
- request-X (where X is the order of the request, starting at 1 for the first request to be executed) (at least 1)
 - This parameter must contain the following values, in this exact order (each in a separate *value* tag): the path to the request file (or the folder containing multiple request files), the path of the XSL transformation file, the error handling mode (strict or lenient), the URL of the Web Service, the specific headers for the request, on one line, separated by the separator specified in the *header-separator* parameter.
- close-X (where X is the order of the request to execute when closing/cleaning up, starting at 1 for the first request to be executed) (optional)
 - This parameter must contain the following values, in this exact order (each in a separate *value* tag): the path to the request file (or the folder containing multiple request files), the path of the XSL transformation file, the error handling mode (strict or lenient), the URL of the Web Service, the specific headers for the request, on one line, separated by the separator specified in the *header-separator* parameter.
- output-folder (mandatory)
 - This parameter contains the path to the folder where the final output will be placed. This output is what will be sent to the XML collector to fetch performance indicators.
- header-separator (optional, default is ;)
 - The string that is used to separate the headers in the header parameter for each request.

1.4 Example

```
<data>
  <data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.
remote.soap.ChainedWebServiceRequestsExecutor</data-accessor>
  <input></input>
  <host>10.128.9.97</host>

  <!-- Possible values: all:connect to all listed hosts, any: stop after first
successful connection -->
  <parameter name="connect-type">
    <value>any</value>
  </parameter>

  <!-- Number of retries to access data -->
  <parameter name="data-retry">
    <value>2</value>
  </parameter>

  <!-- Time before a request fails if no response is received (in seconds) -->
  <parameter name="data-timeout">
    <value>90</value>
  </parameter>

  <!-- Time before a connection attempt fails (in seconds) -->
  <parameter name="connection-timeout">
    <value>180</value>
  </parameter>

  <!-- Encoding for both reading input file and writing request -->
  <parameter name="charset">
    <value>UTF-8</value>
  </parameter>

  <!-- Generic HTTP headers (used for all requests) -->
  <parameter name="headers">
    <value>SOAPAction:http://www.watch4net.com/soap</value>
    <value>AnyHeader:AnyValue</value>
  </parameter>

  <!-- Force the HTTP client to close all connections between pollings -->
  <parameter name="force-close-connections">
```

```

        <value>true</value>
    </parameter>

    <!-- Connect to one of the defined hosts or to all of them. Possible values: any,
all -->
    <parameter name="connect-type">
        <value>any</value>
    </parameter>

    <!-- The input file -->
    <parameter name="input-file">
        <value>/path/to/input/file.xml</value>
    </parameter>

    <!-- The first request to execute -->
    <parameter name="request-1">
        <!-- The file to send to the web service -->
        <value>/path/to/request/file.xml</value>
        <!-- The XSL file to use to transform the response into the second request -->
        <value>/path/to/xsl/file.xsl</value>
        <!-- The error handling mode (either strict or lenient) -->
        <value>strict</value>
        <!-- The URL of the web service -->
        <value>http://(host):8080/webservice</value>
        <!-- The specific headers to include with this header -->
        <value>MyHeader1:Value1;MyHeader2:Value2</value>
    </parameter>

    <!-- The second request to execute -->
    <parameter name="request-2">
        <value>/path/to/folder/containing/requests/</value>
        <value>/path/to/transformation/file.xsl</value>
        <value>lenient</value>
        <value>http://(host):8080/webservice</value>
        <value>MyHeader1:Value1;MyHeader2:Value2</value>
    </parameter>

    <!-- The first request to execute when closing / cleaning up (this is usually the
right place to perform a disconnect operation -->
    <parameter name="close-1">
        <value>/path/to/closing/request/file.xml</value>

```

```
<!-- Sometimes the output doesn't matter, and a transformation isn't required...
this can be left empty in such cases -->
  <value></value>
  <value>http://(host):8080/webservice</value>
  <value>MyHeader1:Value1;MyHeader2:Value2</value>
</parameter>

<!-- The folder that will contain the final output (parsed by the XML Collector)
-->
<parameter name="output-folder">
  <value>/path/to/final/output/folder/</value>
</parameter>

<!-- String to use to separate headers, in case a semicolon won't work -->
<parameter name="header-separator">
  <value>any_string</value>
</parameter>
</data>
```

2 Cisco UCS Example

Here's an example of a configuration for the UcsWebServiceExecutor

2.1 Example

```
<data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.cisco.
UcsWebServiceExecutor</data-accessor>
  <input>http://(host):8080/nuova</input>
  <host><UCS_HOST></host>

  <!-- Possible values: all:connect to all listed hosts, any: stop after first successful
connection -->
  <parameter name="connect-type">
    <value>any</value>
  </parameter>

  <!-- This file contains the request that will be sent to the Cisco UCS Web Service -->
  <parameter name="input-file">
    <value><APG_FOLDER>/Collecting/Text-Collector/<INST>/conf/UCS/request-equipment.Chassis.xml
  </value>
    <value>conf/UCS/request-logout.xml</value>
  </parameter>

  <!-- Number of retries to access data -->
  <parameter name="data-retry">
    <value>2</value>
  </parameter>

  <!-- Time before a request fails if no response is received (in seconds) -->
  <parameter name="data-timeout">
    <value>90</value>
  </parameter>

  <!-- Time before a connection attempt fails (in seconds) -->
  <parameter name="connection-timeout">
    <value>180</value>
  </parameter>

  <!-- Encoding for both reading input file and writing request -->
  <parameter name="charset">
```

```
<value>UTF-8</value>
</parameter>

<!-- Specific soap action header (deprecated, shown only for legacy configurations). Use
headers instead. -->
<parameter name="soap-action">
  <value>http://www.watch4net.com/soap</value>
</parameter>

<!-- Generic HTTP headers -->
<parameter name="headers">
  <value>SOAPAction:http://www.watch4net.com/soap</value>
  <value>AnyHeader:AnyValue</value>
</parameter>

<!-- force the HTTP client to close all connections between pollings -->
<parameter name="force-close-connections">
  <value>true</value>
</parameter>

<!-- This file contains the authorization request that will be sent before querying for data
-->
<parameter name="authorization-request-file">
  <value>conf/UCS/request-auth.xml</value>
</parameter>

<!-- The XPath expression representing the location of the cookie in the authorization
response from the UCS -->
<parameter name="authorization-request-cookie-path" >
  <value>/aaaLogin/@outCookie</value>
</parameter>

<!-- The XPath expression representing the path where the cookie should be inserted in the
request file (which was specified using the "input-file" parameter) -->
<parameter name="insert-cookie-as" >
  <value>/configScope/@cookie</value>
  <value>/aaaLogout/@cookie;/aaaLogout/@inCookie</value>
</parameter>
</data>
```

XML Jobs

1 Xalan

1.1 Description

The Xalan XML job uses the default XML API provided by the Java environment. Therefore, it is heavily tested and widely used. However, it tends to get slower as the size of the XML input grows. Therefore, this job should be used when dealing with small XML chunks (smaller than 5 MB).

1.2 Java class

`com.watch4net.apg.v2.collector.plugins.xmlcollector.engine.XalanXmlJob`

2 Saxon

2.1 Description

The Saxon XML job uses the default the DOM XML API provided by Saxon. It's a little faster than the default Xalan job and allow the usage of XSLT functions inside the XPath queries.

2.2 Java class

`com.watch4net.apg.v2.collector.plugins.xmlcollector.engine.SaxonXmlJob`

3 Nux

3.1 Description

The Nux XML job uses the Nux library, which relies on Saxon for XPath queries. It is significantly faster than the default Java XML API, especially when dealing with large XML blocks (larger than 5MB). This implementation should be used when performance becomes an issue.

3.2 Java class

`com.watch4net.apg.v2.collector.plugins.xmlcollector.engine.NuxXmlJob`

4 JsonToSaxon

4.1 Description

The JsonToSaxon XML job uses the jackson pull parser to read json stream and generates XML SAX events to convert to a xml stream. It uses the default the DOM XML API provided by Saxon. It's a little faster than the default Xalan job and allow the usage of XSLT functions inside the XPath queries. For the XML element names generated from JSON keys to be legal, the patterns matching the regex `^[^a-z_]*|^[^a-z_0-9-._]` are removed. Also, the empty element names after applying regex are replaced with `Not-Available`. Hence the xpath expressions are to be appropriately framed.

4.2 Java class

`com.watch4net.apg.v2.collector.plugins.xmlcollector.engine.JsonToSaxonXmlJob`

5 JsonToNux

5.1 Description

The JsonToNux XML job uses the jackson pull parser to read json stream and generates XML SAX events to convert to a xml stream. It uses the Nux library, which relies on Saxon for XPath queries. It is significantly faster than the default Java XML API, especially when dealing with large XML blocks (larger than 5MB). This implementation should be used when performance becomes an issue. For the XML element names generated from JSON keys to be legal, the patterns matching the regex `^[^a-z_]*|^[^a-z_0-9-._]` are removed. Also, the empty element names after applying regex are replaced with `Not-Available`. Hence the xpath expressions are to be appropriately framed.

5.2 Java class

`com.watch4net.apg.v2.collector.plugins.xmlcollector.engine.JsonToNuxXmlJob`

Configuration

1 Collector Manager configuration

The XML Collector for APG must be added in the main collecting configuration file (collecting.xml), which is located in <APG Root>/Collecting/Collector-Manager/<INSTANCE>/conf. Add the following section if it is not present, and make sure that the *enabled* attribute is set to *true*. Also, if the collected data must go through additional filters before reaching the Backend, replace the value of the *next* attribute to fit your configuration.

```
<collectors>
...
  <collector enabled="true" name="XML-Collector" next="Backend"
config="XML-Collector/Default/conf/xmlcollector.xml" />
...
</collectors>
```

2 Collector configuration

This file is the main configuration file for the adapter. It contains information about how data will be interpreted, as well as how to access the data to parse. By default, this file can be found at <APG-Root>/Collecting/XML-Collector/<INSTANCE>/conf/xmlcollector.xml.

Global structure Here the global structure of the configuration file:

1. <configuration> : The main container for the collector configuration. Every other tag must be nested inside this one.
 - (a) <simultaneous-collecting> : The number of simultaneous requests that will be sent to the device.
 - (b) <polling-interval> : The time, in seconds, between each polling.
 - (c) <collecting-group> : The group in which the variables will be collected. This group *must* be defined in config.xml, or values will be ignored.
 - (d) <source> : The value of the *source* property, as it will appear in APG. The *source* property is used to identify the collector that pushed the given metric in APG.
 - (e) <refresh> : The number of seconds between each time properties are refreshed in the database. If this parameter is set to 0, properties will be refreshed every time that a change is detected. Otherwise, properties will be refreshed periodically, regardless of the changes that might have occurred.
 - (f) <collecting-configuration id=id [max-threads=threads timeout=timeout xml-job=job class namespace-handler=handler]> : This tag acts as a container for configuration options that relate to the same parsing operation (i.e.: parsing of a given command or text file). There is usually one collecting configuration per different file or command to parse.

id Mandatory. Uniquely identifies variables that were obtained using a given collecting configuration.

timeout Optional. Specifies the amount of time that is allocated to a xml-job before timing out, in seconds. Default value is 240.

max-threads Optional. Specifies the number of threads to use to parse the XML input on different hosts. Default value is 1.

xml-job Optional. Specifies the type of XML job that must be used by the XML engine. Default behavior is to use the XalanXmlJob implementation. See section 4 for more details.

namespace-handler Optional. Allows users to create their own namespace handling java classes. There should be no need to use that parameter unless told otherwise by a Watch4Net specialist.

variable Optional. Properties that will be used to generate the unique name for each variable, separated by a whitespace. Be very careful when changing this, as it will result in new variables being created, and may cause variable overwrites.

- i. `<for-each namespace-uri=namespace URI of the tag if any>` : Specify that the file can be separated in datasets defined by a single tag. This is really usefull when parsing a big file containing a repetition of the same entry. Note that after selecting a tab to loop on, the other XPath will need to refer this tag as the root element (Example: `/root/container/Device` with a for each Device means that every other XPath must start with `/Device/...`).
- ii. `<data>` : Acts as a container for configuration options that relate to the way data will be accessed.
 - A. `<include-contexts>` : Location of an external execution contexts file relative to the collector instance directory
 - B. `<contexts name=name>` : Defines an execution context. The data accessor will be run once for each execution context. An execution property can be referred almost anywhere in the data accessor configuration and in an hardcoded property with `@propertyName`.
 - `<properties name=name>` : The execution context property with its name and value
 - C. `<data-accessor>` : The Java class used to access the data (see section 3 for available classes).
 - D. `<input>` : The file or command to parse. The keyword (host) can be used anywhere in the input parameter. Every time that it is encountered, it will be replaced by the host on which the command is being executed (as specified in the `<host>` tag). When using this element, you can specify the optional **type** attribute for it. So far, only two types are supported: `string` and `file`. `string` is the default value for the attribute and tells the collector that the content of this `<input>` tag is

a random character string. However, using the `file` type, the collector knows that the `<input>` tag contains a path to a file. If the specified path is relative, the collector will handle the file to make sure that its absolute path is properly retrieved and used.

- E. `<host>` : This tag can be repeated as many times as necessary. Each `<host>` tag contains the IP address or hostname of a host from which data has to be polled.
- F. `<parameter name=name>` : A container for a data accessor parameter. The *name* attribute is literally the attribute's name, as it will be received by the data-accessor.
 - `<value>` : The value that must be given to the parameter. The *value* tag can be repeated as many times as necessary if a the parameter can contain multiple values.
- iii. `<value name=name value=value [type=type] [unit=unit] [use-start-timestamp=true/false]>`
: A metric that will be available in APG.
 - The *name* attribute is the name that will be given to this metric in APG.
 - The *value* attribute is the XPath expression that points to the value in the XML input.
 - The *type* attribute represents a calculation that will be performed before the value is inserted into APG. It can take one of the following values: *rate*, *ratep* (a positive rate), *delta*, *deltap* (a positive delta) or *counter*. The default value for this attribute is *counter*.
 - The *use-start-timestamp* attribute allows the rate and ratep values to use the start timestamp of the calculation as the output value timestamp.
 - The *unit* attribute represents the unit of the metric. If no unit is specified, no unit will be shown in APG for the current metric.
 - i. `<property name=name value=value [type=type]>` : A property that will be attached to the current value in APG.
 - The *name* attribute is the property name in APG.
 - The *value* attribute is the XPath expression that points to the property in the XML input or the literal value of the property, if type is set to hard-coded.
 - The *type* attribute can take one of the following values: *data* (property is found in the XML data), *hard-coded* (property is hard-coded in the value attribute) or *host* (property is the host from which data was polled). The default value for this attribute is *data*.

Property Sets Property sets can be included in collecting configurations. They contain `<property>` tags exactly like those nested in value tags. The advantage of defining property sets is that they can

be re-used for different values. Therefore, if many values in the same collecting configuration share the same properties, these properties will only need to be defined once. To include a property set in a value, simply nest the `<property-set>property-set-name</property-set>` tag in that value. Property sets are defined as such:

- `<property-set name=name>` : The *name* attribute is the unique identifier of this property set.
 - `<property name=name value=value [type=type]>` : This tag can be repeated as many times as necessary.

Timestamps A timestamp can be defined globally in the collecting configuration and locally for every value. If a timestamp is defined in both, the specific timestamp for a value will have precedence. A timestamp can be defined as such:

- `<timestamp format=format>` : The *format* attribute represents the way that the timestamp is formatted in the input file. The format string must follow a specific pattern, as described in Appendix A. In addition to these patterns, two special format strings can be used instead of the listed patterns:
 - ms : The timestamp is represented in the input file as the number of milliseconds since January 1, 1970.
 - s : The timestamp is represented in the input file as the number of seconds since January 1, 1970.
- `<timestamp type=type>` : The *type* attribute can be used to tell the collector to use the system time as a timestamp instead of looking for a timestamp in the XML input. In this case, it should be used as follows: `<timestamp type="use-system-time" />`

Substitutions Substitutions are helpful when a polled property or metric needs to be replaced before it is pushed into APG. They can be nested inside value or property tags. They are often used when a value appears as text in the collected data. If the data is left unchanged, APG will not be able to interpret the value, which needs to be a number. Substitutions will allow the administrator to configure the collector so that when a particular string is collected as the value for a given metric, the string will be converted to a number. Here is the structure of a substitution statement:

- `<replace value=value by=replacement-value>`
 - The *value* attribute represents the value that is directly collected.

- The *by* attribute represents the replacement value for the metric if collected data is equal to the *value* attribute.

In the following example, a substitution is used for the *fullDuplex* metric. If the collected value is *fullDuplex*, the value for the metric will be set to 1. If the collected value is *halfDuplex*, the value will be set to 0.

```
<value name="fullDuplex" value="duplex">  
  <replace value="fullDuplex" by="1" />  
  <replace value="halfDuplex" by="0" />  
</value>
```

6

Things to do after configuration

When the collector is installed and configured, APG is ready to collect new data from the XML source. The following steps can be taken:

1. Start the APG collecting service.
2. Look at the collecting logs to make sure that everything is starting up as it should.

Troubleshooting

1 The collector won't start

- Review your configuration for typos and XML well-formedness errors.
- Make sure that the contents of the root APG folder is accessible for the apg user.
- Look at the collecting logs that are located in
`<APG Root>/Collecting/Collector-Manager/<INSTANCE>/logs/`
These logs contain helpful messages that might help solving the problem.

2 The collector started successfully, but no data is showing up in APG

If the collector was just started, wait after a few collecting cycles have passed. Data usually takes some time to show up in APG for the first time. If there is still no data, check the following:

- In *collecting.xml*, make sure that the XML Collector is configured and that it is enabled (see section [1](#)).

- In *collecting.xml*, check the *next* attribute for the XML Collector. If the value for the *next* attribute is a filter, make sure that the corresponding filter has its *enabled* attribute set to either *true* or *skip*.
- If the data source is on a remote server, make sure that the APG server is able to access it. If a connectivity issue occurs, a trace will be available in the collecting logs (<APG Root>/Collecting/Collector-Manag

8

Appendix A: Configuration Example

```
<?xml version="1.0" encoding="utf-8"?>
<configuration
  xsi:schemaLocation="http://www.watch4net.com/XMLCollector/xmlcollector.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.watch4net.com/XMLCollector">
  <simultaneous-collecting>3</simultaneous-collecting>
  <polling-interval>300</polling-interval>
  <collecting-group>group</collecting-group>
  <source>W4N-XML-Collector</source>
  <refresh>0</refresh>
  <collecting-configuration id="Config-1" max-threads="1" timeout="300" >
    <data>

<data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.local.FileReader</data-accessor>
    <input>/path/to/XMLFile.xml</input>
    <host>localhost</host>
```

```

</data>
<timestamp format="ms">../relative/XPath/to/global/timestamp</timestamp>
<value name="metric-1" unit="%" value="/absolute/XPath/to/metric-1">
  <property-set>CommonProperties</property-set>
</value>
<value name="metric-2" type="delta" unit="octets"
value="/absolute/XPath/to/metric-2">
  <property-set>CommonProperties</property-set>
</value>
<value name="metric-3" type="rate" unit="octets/s"
value="/absolute/XPath/to/metric-3">
  <timestamp
format="s">../relative/XPath/to/timestamp/for/metric-3</timestamp>
  <property-set>CommonProperties</property-set>
</value>
<value name="metric-4" type="ratep" unit="packets/s"
value="/absolute/XPath/to/metric-4">
  <property name="additional-property-1" type="data"
value="../relative/XPath/to/property" />
  <property-set>CommonProperties</property-set>
</value>
<property-set name="CommonProperties">
  <property name="property-1" type="data"
value="../relative/XPath/to/property-1" />
  <property name="property-2" type="data"
value="../relative/XPath/to/property-2" >
    <replace value="ethernet10" by="10Mbps"/>
    <replace value="ethernet100" by="100Mbps"/>
  </property>
  <property name="property-3" type="hard-coded" value="hard-coded-value" />
</property-set>
</collecting-configuration>
<collecting-configuration id="Config-2">
  <data>

<data-accessor>com.watch4net.apg.v2.collector.plugins.outputparsing.executing.re
mote.soap.SOAPXmlRequestExecutor</data-accessor>
  <input>http://(host):8080/soap-api-invoke</input>
  <host>localhost</host>
  <host>failover-host</host>
  <parameter name="connect-type">
    <value>any</value>

```

```
</parameter>
<parameter name="input-file">
  <value>conf/soap-request.xml</value>
</parameter>
</data>
<timestamp type="use-system-time" />
<value name="metric-5" unit="%" value="/absolute/XPath/to/metric-5">
  <replace value="up" by="100" />
  <replace value="down" by="0" />
  <property name="property-1" type="data"
value="../relative/XPath/to/property-1" />
  <property name="property-2" type="hard-coded" value="hard-coded-value" />
</value>
</collecting-configuration>
</configuration>
```


Appendix B: Timestamps

1 Definitions

The format string can be composed of any of the following symbols:

Letter	Date/Time Component	Example
G	Era designator	AD; BC
y	Year	1998; 98
M	Month in year	January; Jan; 01
w	Week in year	38
W	Week in month	3
D	Day in year	352
d	Day in month	4
F	Day of week in month	2
E	Day in week	Friday; Fri
a	Am/Pm marker	AM; PM
H	Hour in day (0-23)	0
k	Hour in day (1-24)	24
K	Hour in Am/Pm (0-11)	0
h	Hour in Am/Pm (1-12)	12
m	Minute in hour	34
s	Second in minute	42
S	Millisecond	987
z	Time zone	PST; GMT-08:00
Z	Time zone	-0800

The number of times a letter is repeated determines the way data will be interpreted. A letter that is repeated four times or more signifies that the full representation is used, while fewer than four repetitions usually result in an abbreviation.

2 Examples

Timestamp in data	Format used
Monday 27 Oct, 2008 3:30 PM	EEEE d MMM, yyyy h:m a
2008/27/10 15:30	yyyy/d/M H:m
27-10-08 AD, 15:30:54 GMT+04:00	d-M-y G, H:m:s zzzz