



EMC® Watch4net®

Version 6.3u2

Transaction Collector Administration Guide

P/N: pending

Rev : 40425

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 Installation	3
3 Configuration	5
1 Collector Attributes	5
2 Defaults for Transactions	7
3 Available Transactions	9
3.1 Transactions Sequence	10
3.2 http-head-transaction	11
3.3 http-transaction	11
3.4 sql-transaction	12
3.5 tcp-connect-transaction	14
3.6 dns-transaction	15
3.7 sip-transaction	17
3.8 ftp-transaction	17
3.9 sftp-transaction	18
3.10 ldap-transaction	19
3.11 radius-transaction	20
3.12 script-transaction	20
4 Collected Metrics	23
1 Transaction sequence (<i>meta-transaction</i>)	24
2 http-head-transaction	24

3	http-transaction	25
4	sql-transaction	25
5	tcp-connect-transaction	26
6	dns-transaction	26
7	sip-transaction	27
8	ftp-transaction	27
9	sftp-transaction	28
10	ldap-transaction	28
11	radius-transaction	29
5	Technical considerations	31
1	Flow	31
2	Password & Security	32
6	Transaction Collector Configuration Example	33

Preface

Purpose

This guide describes the configuration and usage of the Transaction Collector, which includes the available transactions, possible outputs, configuration properties and logging system.

Intended Audience

This guide is meant to be read by any person who will configure and/or use the Transaction Collector.

Document Organization

1. Product Overview	This section quickly describes the product
3. Configuration	This section describes how to configure the collector
4. Collected Metrics	This section describes the differents collected metrics for each transaction
5. Technical considerations	This section describes some technical parts of the collector
6. Transaction Collector Configuration Example	This section show an example of configuration file

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 Transaction Collector is a specific module for *APG* that allows transaction execution over the network and returns transaction KPIs to *APG* back-end. Here are the supported transaction types.

- **HTTP HEAD:** Transaction that implements an HTTP HEAD request over a specified url.
- **HTTP:** Transaction that implements an HTTP request over a specified url. It can do both GET or POST request and also have a recursive functionality, which download all images, css stylesheets, scripts files, etc. that are linked to the downloaded HTML content. The behavior and recursion level is similar to the one of an HTTP GET request over a page, in a web browser. There's only one level of recursion.
- **SQL-based queries:** Transaction that execute a SQL query over a SQL server, specified through an url.
- **TCP connect:** Transaction that connect to an host over a specified port and execute a command on the server. The content outputed form the server can be read and match to a specified regex.
- **DNS query:** Transaction that execute a query for DNS records over a certain resolver, by asking a resolver for some specific record type for a particular domain name.
- **SIP Request:** Transaction that execute a SIP INVITE transaction with a particular sip-address.

- **FTP:** Transaction that implements some basic FTP commands. Supported commands are RETR (RETRieve), STOR (STORe) and DELE (DELEte).
- **SFTP:** Transaction that is similar to the FTP transaction, but which performs some operations over the SFTP protocol (SSH file transfer protocol).
- **LDAP:** Transaction that performs a search query over a LDAP server.
- **RADIUS:** Transaction that performs an authentication request against a RADIUS server.
- **SCRIPT:** Transaction that execute a custom command (for example, executing a script) on the local machine. The script execution can then output some stuff to the standard output and the collector will read this output, parse it according to a pre-defined format, then generate the corresponding Raw Values.

2

Installation

The Transaction Collector module ships with APG5 but is not installed by default. If not, please use the module manager to do this. The Transaction Collector module name is `Transaction-Collector`.

In order to activate the Transaction Collector, you must edit the `collecting.xml` configuration file in order to add the collector declaration.

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

Then, you can edit the collector configuration file in the *conf* folder of the Transaction-Collector module. The default file name for this configuration file is `transaction-collector.xml`.

Configuration

The following section explains the different parameters for the Transaction Collector. A complete configuration file example is available at [section 6](#).

1 Collector Attributes

The transaction-collector.xml file is the main configuration file for the adapter. It contains information about transactions definitions and how data will be interpreted.

The file must always begins with the following section, which describe some general parameters for the collector.

```
<collector>
  <source>demo-transaction-collector</source>
  <max-simultaneous-transactions>10</max-simultaneous-transactions>
  <timestamp-threshold>10000</timestamp-threshold>
  <pending-transaction-soft-limit>1000</pending-transaction-soft-limit>
  <max-simultaneous-downloads-per-transaction>10</max-simultaneous-downloads-per-transaction>
  <property-refresh-frequency>43200</property-refresh-frequency>
```

```
</collector>
```

All elements related to the collector own configuration are nested inside this one.

- **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*.
- **max-simultaneous-transactions**: The global number of maximum simultaneous transactions that the collector is authorized to run. Keep in mind that each transaction instance that gets executed can also have multiple threads, so be careful when setting a high value here.
- **timestamp-threshold**: Defines the threshold limit about the maximum time (in **milliseconds**) a transaction can live in the execution queue before being executed. If the threshold limit is reached before the execution of the transaction, the transaction will be killed, a warning message will be logged and no raw values are going to be pushed in the backend.
- **pending-transaction-soft-limit**: When the number of pending transaction in the waiting queue is greater than this value, some warning messages will start appearing in the logs. This limit does not forbid transactions to run, but simply inform you when too many transactions are in the waiting queue. This can be useful to determine when your collector performance are struggling: it can be a warning that some transactions use a lot of execution time or that you are generating too many transaction instances comparing to the rythm at which they are executed.
- **max-simultaneous-downloads-per-transaction**: For some transactions (such as recursive *http-transaction*), many files can be downloaded at the same time. This value define the maximum number of file that can be downloaded simultaneously, by one transaction. Each file that gets downloaded creates a new thread.
- **property-refresh-frequency**: the Transaction Collector implements a cache that keeps in memory the hashcode (signature) of the properties of each Raw Value that were sent to the next component (one cache entry per variable). This cache is useful in order to avoid forcing the backend to refresh uselessly the properties of a variable. However, it needs to be resetted once in a while to avoid the cache to become too big in memory size. The specified value is in seconds. This parameter is optional (default value is 43200 seconds (12 hours)).

Note: It is strongly recommended to use a text editor that allow the validation of the XML configuration file against the XSD file

2 Defaults for Transactions

The next block defined some default value to be use as defaults values if the transactions configurations does not override these properties.

```
<defaults>
  <group>group</group>
  <period>300000</period>
  <delay>1000</delay>
  <repeat aggregate="average">0</repeat>

  <properties>
    <property name="propname" value="Some_Value_Here"/>
  </properties>

  <output-directory>tmp</output-directory>
  <transaction-instances-output-to-keep>true</transaction-instances-output-to-keep>

  <http-headers>
    <property name="user-agent" value="Mozilla/5.0_(X11;_Linux_i686;_en-US;_rv:1.9.0.11)
    _Gecko/2009060308_Ubuntu/9.04_(jaunty)_Firefox/3.0.11" />
  </http-headers>
  <url-encoding>UTF-8</url-encoding>
</defaults>
```

- **group**: The group in which the variables will be collected. This group must be defined in the backend configuration, or values will be rejected. Default value is **group**.
- **period**: The time interval (in **milliseconds**) before rescheduling transaction. Setting the frequency to 1000 will schedule the transaction to run every seconds, for example.
- **delay**: The start delay for the transaction (in **milliseconds**). This acts as a pause before executing the transaction. It can be useful when used inside a transaction sequence (also known as *meta-transaction*), to prevent two transactions instance from running too quickly. Default value is 0.
- **repeat**: The number of times the transaction will be repeated. Example : if repeat is zero, each transaction will be executed once; if repeat is two, each transaction will be executed 3 times.

If the repeat parameter is higher than 0, the metric values of the corresponding transaction will be determine by the aggregate attribute of the repeat parameter (Possible values : average, minimal, maximal, median and delta (max - min)). For instance, when using `<repeat`

`aggregate="average">2</repeat>`, the `elapsedtime` metric will be the average of the elapsed time of the 3 transaction executed.

There is an exception for the `repeat` parameter. If a `status` appears in the transaction with a repeat, the value of that metric will not be aggregate. It's value will be the `status` if the status is the same for all the transaction of the repeat, otherwise the `status` metric will not be send to the APG. Example: for an `http-get-transaction` transaction with a `repeat=2`, if status returned are {200, 200, 200} then the `status` metric will be 200, but if the status returned are {200, 200, 404}, the `status` metric will not be returned. This also applied on the `sql-transaction` for the `count` metric.

Default value is 0.

- `properties`: contains some APG properties with which any raw value will be tagged. Please note that with default settings, a property name cannot be longer than 8 characters. You can however use aliases in APG's frontend to display user-friendly names.
- `source-ip`: The interface on which the collector binds when issuing transactions. If the server has more than one interface, then specify the IP address of the interface to use. Default value is 0.0.0.0, which automatically selects one of the available interfaces. This setting may not be used by all transactions.
- `output-directory`: Directory path to use for transactions that requires to store output or downloaded data. The default path, relative to the collector installation directory, is `tmp`. This folder will automatically be deleted from the filesystem if empty at the end of the transaction.
- `transaction-instances-output-to-keep`: Value specifying the number of downloaded content folder to keep, for a particular transaction (one folder = one transaction instance). If value is set to 0, none will be kept and all the downloaded content will be cleaned up after the transaction instance. When the number of kept folder is equal or greater to the value defined by this tag for a particular transaction, the older folders are automatically deleted. Default value is 100.
- `http-request-headers`: This tag defines additional HTTP header properties you need to send with the transaction. Additional headers needs to be included using the `<property>` tag, using a name and value attribute. This can be useful to force some global cookies, mock a user-agent, etc.
- `url-encoding`: defines the encoding to use for HTTP URLs. It is highly recommended to let it to the default value, which is UTF-8.

3 Available Transactions

Now that we have defined the global and collector specifics parameters, we can define the transactions that we want to execute. The type of the transaction has to be defined as a specific attribute: `xsi:type`.

- `http-get-headers-transaction`: Retrieves only headers of an HTTP transaction.
- `http-transaction`: Allow the execution of an HTTP transaction. Supplying the `method` argument, we can set the transaction to either do a GET or a POST request to a specified URL. If the `recursive` argument is set to `true`, all the content from the specified URL is download, and also all the referenced `<img src=...>`, `<script src=...>`, `<link src=...>` from the response. It is possible to perform HTTPS transactions as well, simply by using the `https://...` notation rather than the `http://...` for the URL. If SSL validation is enforced, you might have to ensure that the related certificates are properly inserted into the Java TrustStore. For further details, visit <http://docs.oracle.com/javase/1.4.2/docs/guide/security/jsse/JSSERefGuide.html#CustomizingStores> and <http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html>.
- `sql-transaction`: Allow to execute a SQL query upon a distant server.
- `tcp-connect-transaction`: Allow to connect using TCP on a host with a specified port. A command can then be executed and it is also possible to wait for a response from the server. This is pretty similar to the TELNET executable (except that a "read-timeout" has to be provided to avoid the transaction to be hanging indefinitely).
- `dns-transaction`: Perform a DNS query over a resolver, for a particular host to resolve. Many query types are available and are going to be covered in section 3.6.
- `sip-transaction`: Perform a SIP Invite over a SIP address.
- `ftp-transaction`: Perform some basic FTP commands such as `RETR` (retrieving a remote file or directory), `STOR` (send a local file to a remote directory) or `DELE` (delete a remote file).
- `sftp-transaction`: Perform some basic operations such as `get` (retrieving a file or a folder), `put` (sending a local file on a server) or `rm` (deleting a file).
- `ldap-transaction`: Perform a search query over a LDAP server using a search filter.
- `radius-transaction`: Perform an authentication request against a RADIUS server.

Here's an example of a transaction as define into the configuration file. Each and every transaction is define by it's own subset of parameters, some being mandatory, some being optional. We will cover them later. This example show the simplest transaction possible, with only the required parameters filled.

```
<transaction xsi:type="http-head-transaction">
  <id>get-watch4net</id>
  <url>http://watch4net.com</url>
</transaction>
```

All transactions must define an `<id>` tag, which is a unique identifier for the transaction. At the beginning of the configuration file, a `<defaults>` tag allow the user to define some default values for various parameters, such as `<frequency>`, `<delay>`, `<output-directory>`, `<transaction-instances-output-to-keep>`, `<repeat>` and `<group>`. These parameters are used by the transactions but not mandatory to be define within the `<transaction>` tag itself. If the value isn't specified through the `<transaction>` tag, the value from the `<defaults>` will be used instead. This can be useful to avoid duplication of parameters, for parameters that aren't likely to change through each transaction (for example, if we want to execute the transaction at the same frequency or to use the same group for each transaction). If the value is define within the transaction, this value will be used instead of the `<defaults>` tag value. The following tags are mandatory in the `<transaction>` tag, but can be inherited from the `<defaults>` tag (except for `<id>`):

- `<id>`: The unique id the the transaction. This id must be unique within the xml file, and can only contain ascii letters, digits, dashes and underscores (regular expression: `[a-zA-Z0-9\_-]+`). This field is obviously mandatory.
- `<property-ref>`: This field is optional. It allows to add a property set to the transaction as specified by a `<property-set>` tag later defined in the configuration file. The value defined within this tag correspond to the id of the `<property-set>`.
- `<period>`: The value from `<defaults>` is taken if none is specified.
- `<delay>`: The value from `<defaults>` is taken if none is specified.
- `<output-directory>`: The value from `<defaults>` is taken if none is specified.
- `<transaction-instances-output-to-keep>`: The value from `<defaults>` is taken if none is specified.
- `<repeat>`: The value from `<defaults>` is taken if none is specified.
- `<group>`: The value from `<defaults>` is taken if none is specified.

3.1 Transactions Sequence

The transaction sequence (also known as *meta-transaction*) is, in all points, pretty similar to the regular transactions. It is virtually just a container for transactions. The transaction sequence will execute all

<transaction> tags inside it sequentially. Transaction sequence are useful to force transactions to execute in the specified order. The meta-transaction takes the following parameters:

- <id>: Same thing as the <id> defined earlier
- <period>: The value from <defaults> is taken if none is specified.
- <delay>: The value from <defaults> is taken if none is specified.
- <repeat>: Tthe value from <defaults> is taken if none is specified.

Note: A <meta-transaction> cannot be defined within another <meta-transaction>. Only transaction can be define within a <meta-transaction> tag.

3.2 http-head-transaction

- The HTTP Head Transaction supports the same parameters as the HTTP Transactions.

3.3 http-transaction

- <url>: The url to hit. **The url must end with a "/" if you are hitting a page that does not have an extension (for example, <http://www.watch4net.com/>). This is really important in order to have the transaction to be fully and correctly executed.**
- <request-headers> : Same thing and purpose as the defaults http-request-headers, but only for this transaction. You can use the *inherit* attribute to *false* if you don't want your transaction to inherit from the <http-request-headers> key/value pairs.
- <request-parameters>: For GET request, the different key/value parameters to be appended to the URL. This field is optional, but if specified, the different values will be appended to the <url>.

If <url> value was <http://example-url.com/page-requested/>, then the requested url, during the execution of the transaction would be:

<http://example-url.com/page-requested/?parameter1=value1¶meter2=value>

Note that these values can be automatically appended to the url in the <url> tag to avoid using this tag. However, don't use the <parameters> tag if some key/value have been appended in

the `<url>` tag.

For POST request, the different `<parameter>` defined within `<parameters>` are the one passed to the requested page using POST.

```
<request-parameters>
  <parameter name="parameter1" value="value1" />
  <parameter name="parameter2" value="value2" />
</request-parameters>
```

- `<url-encoding>`: The encoding format to use to encode the parameters from `<request-parameters>`. Default value is *UTF-8*.
- `<cookies-from>`: Specify the id of the transaction from which you want to use the cookies. For example, let's assume 2 transactions previously defined into a transaction sequence (*http-get-transaction1* and *http-get-transaction2*). Suppose that *http-get-transaction1* returned 3 cookies, you can use these cookies in *http-get-transaction2* by simply forcing the `<cookies-from>` field to be *http-get-transaction1*.

Note: You can only retrieve cookies from transactions that are defined **within** the same transaction sequence (meta-transaction). Forcing the tag in transactions that are defined outside a (or from another) transaction sequence will have no effect.

Note: It is a really good practice to mock a user agent using a `http-request-headers` property when doing HTTP based transactions. Nowadays, with all the different browsers and screen size, more and more websites check the user agent to ensure the best user experience possible. If there is no user agent, some website redirect to some custom basic landing page that might not fit what you would expect.

3.4 sql-transaction

- `<url>`: The JDBC url. Each database provider has a specific nomenclature for the url. A partial list is available below. It is very important to add the database name against which the query is going to be executed in the url (please refer to the driver documentation).

- `<sql-driver>`: The sql JDBC driver to be use. For a list of available driver see the official JDBC Drivers page : <http://developers.sun.com/product/jdbc/drivers/>. Note that all those driver are not shipped with the standard APG installation, you might need to download the jar, for a specific database, and copy it to the `lib` directory of the collector. This field is mandatory.
- `<sql-user>`: The username credential. This field is mandatory.
- `<sql-pass>`: The password credential. *This field is mandatory but can be empty. **However, it is not recommended to run your database engine without any password.*** The password can be obfuscated. See section [Password & Security](#) for further details.
- `<sql-query>`: The query to be executed on the server. This field is mandatory.

Note: Despite some database engine allow the username and password to be pass through the url, you have to specify these two attributes in their respective tags

```

<!-- Working example for MySql 5.1 database... -->
<transaction xsi:type="sql-transaction">
  <id>sqltrans1</id>
  <url>jdbc:mysql://127.0.0.1:53306/apg</url>
  <sql-driver>com.mysql.jdbc.Driver</sql-driver>
  <sql-user>apg</sql-user>
  <sql-pass>password</sql-pass>
  <sql-query>select 1 from data_variable</sql-query>
</transaction>

<!-- MySql example... -->
<transaction xsi:type="sql-transaction">
  <id>...</id>
  <url>jdbc:mysql://<host>:<port3306>/<database></url>
  <sql-driver>com.mysql.jdbc.Driver</sql-driver>
  ...
</transaction>

<!-- Oracle example... -->
<transaction xsi:type="sql-transaction">
  <id>...</id>
  <url>jdbc:oracle:thin:@<host>:<port1521>:<sid></url>
  <sql-driver>oracle.jdbc.OracleDriver</sql-driver>
  ...
</transaction>

```

```
<!-- SQL Server example... -->
<transaction xsi:type="sql-transaction">
  <id>...</id>
  <url>jdbc:microsoft:sqlserver://<host>:<port1433>;DatabaseName=<database></url>
  <sql-driver>com.microsoft.sqlserver.jdbc.SQLServerDriver</sql-driver>
  ...
</transaction>

<!-- DB2 example... -->
<transaction xsi:type="sql-transaction">
  <id>...</id>
  <url>jdbc:db2://<host>:<port50000>/<database></url>
  <sql-driver>com.ibm.db2.jcc.DB2Driver</sql-driver>
  ...
</transaction>

<!-- Informix example... -->
<transaction xsi:type="sql-transaction">
  <id>...</id>
  <url>jdbc:informix-sqli://<host>:<port>/<database>;informixserver=<dbservername></url>
  <sql-driver>com.informix.jdbc.IfxDriver</sql-driver>
  ...
</transaction>
```

Note: These examples might only work for a certain version of the database, see the official database provider documentation for the correct url and driver.

3.5 tcp-connect-transaction

- `<connection hostname="localhost" port="80" timeout="100">`: the host and port on which the tcp socket should be opened. This element is mandatory. The connection timeout, in **milliseconds**, is optional with a default value of 1000.
- `<command>`: the command to be sent once connected. This field is optional.
- `<read match="OK .*" encoding="UTF-8" timeout="1000">`: when this element is set, the transaction will wait for an answer from the target host. If the server is unlikely to return some content, do not set this element, to avoid the transaction to be hanging. `match` is an optional regular

expression against which the returned content from the server will be matched (to produce a raw value), `encoding` is an optional field that contains the response encoding (default is UTF-8), and the read timeout, in **milliseconds**, is also optional with a default value of 1000.

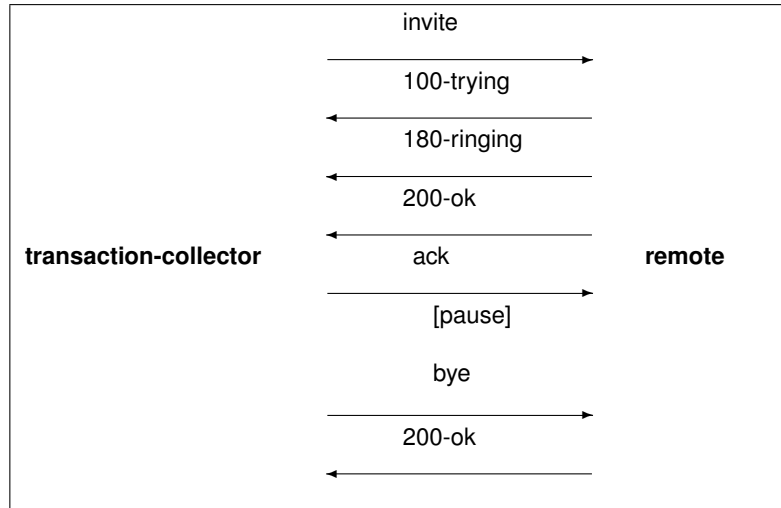
3.6 dns-transaction

- `<name>`: the name of the website to resolve. This field is mandatory.
- `<resolver>`: the resolver address to use. This field is mandatory. The value must be put into the `hostname` attribute. Optionally, the `port` attribute can be used to specify a custom port number. However, port 53 is used by default if the port number is not provided.
- `<timeout>`: the timeout, in **milliseconds**, after which the query will abort, if unresponding. This field is mandatory.
- `<record-type>`: the record type that we want to retrieve. This field is mandatory. The record type must be one of the following:
 - A: Address
 - A6: IPv6 address (experimental)
 - AAAA: IPv6 address
 - AFSDDB: AFS cell database
 - ANY: Matches any type
 - APL: Address Prefix List
 - ATMA: ATM address
 - CERT: Certificate
 - CNAME: Canonical name (alias)
 - DNAME: Non-terminal name redirection
 - DNSKEY: DNSSEC Key
 - DS: Delegation Signer
 - EID: Endpoint identifier
 - GPOS: Geographical position (withdrawn)
 - HINFO: Host information
 - IPSECKEY: IPSEC key

- ISDN: ISDN calling address
- KEY: Key
- KX: Key exchange
- LOC: Location
- MB: Mailbox domain name
- MD: Mail destination
- MF: Mail forwarder
- MG: Mail group member
- MINFO: Mailbox information
- MR: Mail rename name
- MX: Mail routing information
- NAPTR: Naming authority pointer
- NIMLOC: Nimrod locator
- NS: Name server
- NSAP: NSAP address
- NSEC: Next Secure Name
- NULL: Null record
- NXT: Next valid name in zone
- PTR: Domain name pointer
- PX: X.400 mail mapping
- RP: Responsible person
- RRSIG: Resource Record Signature
- RT: Router
- SIG: Signature
- SOA: Start of authority
- SPF: Sender Policy Framework (experimental)
- SRV: Server selection
- SSHFP: SSH Key Fingerprint
- TXT: Text strings
- WKS: Well known services
- X25: X.25 calling address

3.7 sip-transaction

The sip-transaction execute and INVITE Transaction (as defined in the RCF 3261), following this scenario :



- **<sip-address>**: the sip remote address. Format : *sip:name@hostname.com*. This field is mandatory.
- **<local-port>**: the udp local port used to receive the responses. This field is mandatory.
- **<sip-proxy-address>**: the sip proxy server. Format : *<ipaddress>:<port>/UDP*. This field is optional.
- **<timeout>**: the timeout, in **milliseconds**, after which the query will abort. This field is optional.
- **<pause>**: the delay between the time the remote address send the first OK message and the BYE message. In other term, the length of the conversation. This field is mandatory.

3.8 ftp-transaction

- **<connection>**: this mandatory tag contains the different informations needed in order to establish the connection with the remote ftp server. There are three required attributes: *hostname*

(the hostname or IP address of the FTP server), `username` and `password` (the username and password required to connect to the FTP server). The `password` attribute is mandatory, but can be an empty string, if there is no password associated with the specified user. The password can be obfuscated. See section [Password & Security](#) for further details. There is also two optional attributes: `port` (the port to connect to. Default value is **21**) and the `timeout` which specifies the timeout time, in **milliseconds**, for the connection. Default value is **1000**. Value of 0 means "no timeout".

- `<command>`: this tag is also mandatory and specify the command to be executed on the FTP server. The `type` attribute is mandatory, it describes the type of command that has to be executed. Value can be RETR (retrieve), STOR (store) or DELE (delete). The mandatory `remote-target` attribute specify the ressource on the remote FTP server against which the command should be executed. The `local-file` attribute is use with the STOR command to specify the local file to be uploaded.

Optionally, the `hash` attribute can be populated with the expected MD5 checksum value for the specified file. For a RETR over a single file, the checksum will be calculated once the file download will be completed. For a STOR, the hash will be calculated before uploading the file on the FTP server. No hash valiation is performed on RETR over a directory or during a DELE.

Similarly to the the `hash` attribute, the `match` attribute allow the user to specify a regular expression (regex) to be matched against the content of the specified file.

The optional `timeout` attribute specify the duration, in **milliseconds**, before a read timeout occurs, while downloading or uploading a file. Default value is **1000**. Value of 0 means "no timeout".

Note: Be careful with the FTP transaction, since the RETR or STOR on big files can be very long and thus, slow down or prevent other transactions. For big files, it is recommended to specify a `timeout` value for the `<command>` tag.

Note: For the RETR command over a directory, please note that all the files are downloaded sequentially, in comparison with the recursive HTTP transaction where the files are downloaded in parallel.

3.9 sftp-transaction

- `<connection>`: this mandatory tag contains the different informations needed in order to establish the connection with the remote sftp server. There are two required attributes: `hostname`

(the hostname or IP address of the FTP server), `username` (the username to use for the SSH connection over the server). You can also use the two following optional attributes: `port` (the port to connect to. Default value is **22**) and the `timeout` which specifies the timeout time, in **milliseconds**, for the connection. Default value is **1000**. Value of 0 means "no timeout".

The authentication can be done in two different ways, either by using the `password` attribute which means specifying the password directly into the config, or either by specifying the `private-key` attribute, which specify the path to a file containing the private key to connect to the server. When using the `private-key` attribute, a passphrase can be specify in order to decrypt **Proc-Type: 4, ENCRYPTED, DES-CBC, DES-EDE3-CBC, AES-128-CBC, AES-192-CBC and AES-256-CBC** encrypted private keys. The password can be obfuscated. See section [Password & Security](#) for further details.

- `<command>`: this tag is also mandatory and specify the command to be executed on the server. The `type` attribute is mandatory, it describes the type of command that has to be executed. Value can be `get` (retrieve), `put` (store) or `rm` (delete). The mandatory `remote-target` attribute specify the ressource on the remote FTP server against which the command should be executed. The `local-file` attribute is use with the `command` to specify the local file to be uploaded. Optionally, the `hash` attribute can be populated with the expected MD5 checksum value for the specified file. For a `get` over a single file, the checksum will be calculated once the file download will be completed. For a `put`, the hash will be calculated before uploading the file on the FTP server. No hash valiation is performed on `get` over a directory or during a `rm`. Similarly to the the `hash` attribute, the `match` attribute allow the user to specify a regular expression (regex) to be matched against the content of the specified file.

Note: For the `get` command over a directory, please note that all the files are downloaded sequentially, in comparison with the recursive HTTP transaction where the files are downloaded in parallel.

3.10 ldap-transaction

- `<connection>`: this mandatory tag contains the different informations needed in order to establish the connection with the LDAP server. There are few required attributes: `url` (the ldap url to use), `authentication-type` (the authentication level to be used for the connection. Possible values are **none** and **simple**). Optionally, the `principal` (the principal to use), the `credentials` (the credential (password) for the specified principal) and the `timeout` (specifies the timeout time, in **milliseconds**, for the

connection) attributes can be specified. In order to use anonymous authentication, you can either set the `authentication-type` value to **none** or omit the `principal` and `credentials` attributes. Please note that as soon as one of these two criteria is met, anonymous authentication is used. The credential string can be obfuscated. See section [Password & Security](#) for further details.

Finally, the boolean `useSSL` attribute can be use to enable SSL support. If SSL is unused, the password will be transmitted in clear text during the transmission with the LDAP server, hence creating an unnecessary vulnerability. In order to enable SSL, you must import the server certificate into the keystore (more information available at <http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html>) and make sure to set the `javax.net.ssl.trustStore` system property in the `modules.properties` file of the **Collector Manager**. The `javax.net.ssl.trustStore` property should contain the local absolute path to the keystore file containing the server certificate. If the property is not set, JAVA will choose the default keystore.

- `<query>`: this tag is also mandatory and specify the different attributes needed in order to perform the search query over the given server. There are three mandatory attributes: `base` (the base node where the search should start), `search-filter` (the filter to be used to match the different nodes in the specified scope) and `search-scope` (the scope of the search. Possible values are **object**, **one-level** and **subtree**). Also, two optionals attributes can be use: `timeout` (timeout value for the search, in **milliseconds** (0 means no timeout)) and `dereference-links` (boolean value that enables/disables link dereferencing during the search).

3.11 radius-transaction

- `<connection>`: this mandatory tag contains the different informations needed in order to contact the RADIUS server. The required attributes are the following: `hostname` (the RADIUS server host), `username` (the username to use), `password` (the password to use), `shared-secret` (the shared secret used to encrypt the transmission) and `authentication-type` (**pap** or **chap**). The password can be obfuscated. See section [Password & Security](#) for further details. The optional attributes are: `port` (the (auth, not accounting) port number to connect to. Default value is **1812**), `retries` (the number of times the authentication request is attempted. There is a retry only after a timeout occurred (if the server is down or not responding)) and `timeout` (timeout value for the request, in **milliseconds** (0 means no timeout)).

3.12 script-transaction

- `<script>`: this mandatory field specify the path of the command to be executed on the local server. Absolute path should be used in order to avoid issues.

Once the script got executed, the Transaction Collector will read the content that was outputed to the standard output (STDOUT). The output must match the following syntax:

Metric Name<semi-colon>**Value**<semi-colon>**Unit**

Each line needs a line break between them. Here's a valid example of output.

```
initial-delay;423;ms  
ping;23;ms  
records;154;  
uptime;99.674;
```

This would generate 4 different Raw Values that would be sent to the next collecting component. Please note that the unit field is optional.

4

Collected Metrics

Each transaction generates a pre-defined set of metrics. These values varies from one transaction type to another.

- **Availability:** This metric is returned by each and every transaction and specify the success or failure of a transaction. If all transactions iterations (repeat) failed, then the availability will be 0. However, if at least one iteration was able to end without fatal error, then the availability will be 100. This metric is available for all transaction, and shall always be returned.

For `http-transaction` with `recursive` attribute set to `true`, the availability is a little bit different. If one ressource (i.e. a stylesheet, JavaScript script, etc.) can't be downloaded successfully, the global availability of the transaction will be slightly changed. The availability is calculated according to:

$$\frac{\text{total \# of successfully downloaded ressources}}{\text{total\# of ressources to download}} \times 100 \quad (1)$$

- **RepeatFailureCount:** This metric can be returned by all transaction. It represents the number of transaction iterations that failed. The metric will be sent only if the repeat value in the config is greater than 0.

This extra metric can be very useful to determine when a particular transaction experienced some technical difficulties. Combined with the Transaction Collector logs and the availability metric, it can help monitoring the health status of a particular transaction.

All generated values will be tagged with the following properties, which can be overridden with static ones in the collector configuration.

- **source**: the collector ID
- **devtype**: the string 'Host'
- **device**: the target hostname or IP address
- **parttype**: the string 'Transaction'
- **trantype**: the transaction type (for instance, http, dns)
- **part**: the transaction ID
- **seqname**: the sequence ID, if the transaction belongs to a sequence
- **seqorder**: the sequence order, if the transaction belongs to a sequence
- **repeat**: `value:aggregation`, if the transaction repeat value is greater than 0

Some other properties can be set by each transaction. See below for a list of these properties.

1 Transaction sequence (*meta-transaction*)

Transaction sequence does not generate any raw values. However, all transactions within it will generate raw values.

2 http-head-transaction

- **totalsize**: The total number of kb that were downloaded (*kb*)
- **elapsedtime**: The total duration of the transaction, from the moment where it was started, up to the end of the last step (*ms*)
- **averagespeed**: The average speed of the transaction. Roughly, this is the totalsize divided by the elapsedtime (*kb/s*)
- **status**: The status of the request. Refer to [this page](#) for more informations. In case where the `repeat` attribute is greater than 0, if more than one different status are obtained during the different requests, no status metrics will be sent to the backend (*no units*)

The transaction-specific following properties will be added to the generated values:

- **target**: the request URL
- **method**: the request method - HEAD

3 http-transaction

- **filecount**: The total number of files that were downloaded (*# of files*)
- **totalsize**: The total number of kb that were downloaded (*kb*)
- **elapsedtime**: The total duration of the transaction, from the moment where it was started, up to the end of the last step (*ms*)
- **receivingtime**: The total duration of the transaction during which data was received, from the moment when we received the first byte up to the last byte (*ms*)
- **averagespeed**: The average speed of the transaction. Roughly, this is the totalsize divided by the elapsedtime (*kb/s*)
- **incomingspeed**: The average speed of receiving data. Roughly, this is the totalsize divided by the receivingtime (*kb/s*)
- **latency**: The total time required before receiving a first byte of data, since the beginning of the transaction (*ms*)
- **linearelapsetime**: The linear total time of download for all files. This value is a sum of all the individual download time for each files that were downloaded (*ms*)
- **status**: The status of the request. Refer to [this page](#) for more informations. In case where the `repeat` attribute is greater than 0, if more than one different status are obtained during the different requests, no `status` metrics will be sent to the backend (*no units*)

The transaction-specific following properties will be added to the generated values:

- **target**: the request URL
- **method**: the request method (with the 'R' prefix for recursive ones) - GET|RGET|POST|RPOST

4 sql-transaction

- **elapsedtime**: The total duration of the transaction, from the moment where it was started, up to the end of the last step (*ms*)

- **rows**: The total number of rows that were affected by the query. In case where the `repeat` attribute is greater than 0, if the number of rows is different between at least two repetitions, no `rows` metrics will be sent to the backend (*rows*)

The transaction-specific following properties will be added to the generated values:

- `target`: the connection URL
- `database`: the target database type (for instance, `mysql` or `oracle`)

5 tcp-connect-transaction

- **totalsize**: The total number of kb that were downloaded (*kb*)
- **elapsedtime**: The total duration of the transaction, from the moment where it was started, up to the end of the last step (*ms*)
- **averagespeed**: The average speed of the transaction. Roughly, this is the `totalsize` divided by the `elapsedtime` (*kb/s*)
- **latency**: The total time required before receiving a first byte of data, since the beginning of the transaction (*ms*)
- **regexmatch**: Tells if the expected regex matched the received output. In case where the `repeat` attribute is greater than 0, if the `regexmatch` value is different between at least two repetitions, no `regexmatch` metrics will be sent to the backend (*no units*)

The transaction-specific following properties will be added to the generated values:

- `target`: the hostname or IP address to reach
- `port`: the port used to connect to the host

6 dns-transaction

- **elapsedtime**: The total duration of the transaction, from the moment where it was started, up to the end of the last step (*ms*)
- **recordcount**: The number of records that were found. In case where the `repeat` attribute is greater than 0, if the `recordcount` value is different between at least two repetitions, no `recordcount` metrics will be sent to the backend (*no units*)

The transaction-specific following properties will be added to the generated values:

- **target:** the name to resolve
- **device:** the dns server name or IP address
- **port:** the port used to connect to the dns server

7 sip-transaction

- **tryingtime:** The duration between the trying message and the ringing message (*ms*)
- **ringingtime:** The duration between the ringing message and the first ok message (*ms*)
- **calllength:** The total duration of the dialog, from the moment the invite message is send, up to the moment the last ok message is received (*ms*)

The transaction-specific following properties will be added to the generated values:

- **target:** the SIP server address
- **proxy:** the SIP proxy, when one has been defined (and thus used)

8 ftp-transaction

- **filecount:** The total number of files that were downloaded (*# of files*)
- **totalsize:** The total number of kb that were downloaded (*kb*)
- **elapsedtime:** The total duration of the transaction, from the moment where it was started, up to the end of the last step (*ms*)
- **averagespeed:** The average speed of the transaction. Roughly, this is the totalsize divided by the elapsedtime (*kb/s*)
- **latency:** The total time required before receiving a first byte of data, since the beginning of the transaction (*ms*)
- **status:** The status of the request. Refer to [this page](#) for more informations. In case where the repeat attribute is greater than 0, if more than one different status are obtained during the different requests, no status metrics will be sent to the backend (*no units*)
- **DigestMatch:** Tells if the expected MD5 hash matched the one from the specified file. In case where the repeat attribute is greater than 0, if the DigestMatch value is different between at least two repetitions, no DigestMatch metrics will be sent to the backend (*no units*)

- **regexmatch**: Tells if the expected regex matched the file content. In case where the `repeat` attribute is greater than 0, if the `regexmatch` value is different between at least two repetitions, no `regexmatch` metrics will be sent to the backend (*no units*)

The transaction-specific following property will be added to the generated values:

- `port`: the port used to connect to the FTP server
- `target`: the URI target for the transaction

9 sftp-transaction

- **filecount**: The total number of files that were downloaded (*# of files*)
- **totalsize**: The total number of kb that were downloaded (*kb*)
- **elapsedtime**: The total duration of the transaction, from the moment where it was started, up to the end of the last step (*ms*)
- **averagespeed**: The average speed of the transaction. Roughly, this is the `totalsize` divided by the `elapsedtime` (*kb/s*)
- **latency**: The total time required before receiving a first byte of data, since the beginning of the transaction (*ms*)
- **DigestMatch**: Tells if the expected MD5 hash matched the one from the specified file. In case where the `repeat` attribute is greater than 0, if the `DigestMatch` value is different between at least two repetitions, no `DigestMatch` metrics will be sent to the backend (*no units*)
- **regexmatch**: Tells if the expected regex matched the file content. In case where the `repeat` attribute is greater than 0, if the `regexmatch` value is different between at least two repetitions, no `regexmatch` metrics will be sent to the backend (*no units*)

The transaction-specific following property will be added to the generated values:

- `port`: the port used to connect to the SFTP server
- `target`: the URI target for the transaction

10 ldap-transaction

- **elapsedtime**: The total duration of the transaction, from the moment where it was started, up to the end of the last step (*ms*)

- **rows**: The total number of results that were returned by the search query. In case where the `repeat` attribute is greater than 0, if the number of rows is different between at least two repetitions, no rows metrics will be sent to the backend (*rows*)

The transaction-specific following property will be added to the generated values:

- `target`: the URI target for the transaction

11 radius-transaction

- **elapsedtime**: The total duration of the transaction, from the moment where it was started, up to the end of the last step (*ms*)
- **status**: The status of the authentication attempt: 100 if we received a Access-Accept packet, 0 in all other cases.

The transaction-specific following property will be added to the generated values:

- `port`: the port used to connect to the RADIUS server
- `target`: the hostname for the transaction

Technical considerations

1 Flow

Once the collector is instantiated and started, the whole configuration is read and parsed. Each transaction and meta-transaction are setup and added to the timer that will generate an instance of the transaction according to the specified frequency. At this point, it is important to understand that once the config is read, it is kept in memory, which means that any change to the XML configuration file will require the collector to be restarted in order to have the change considered.

Once created, the transaction instances are put in a queue, waiting to be processed. The elements are popped out of the queue as soon as there is a slot to process it (the maximum number of slots is specified through the `<max-simultaneous-transactions>` argument of the config). Once a transaction instance is started, it can't be paused. All the raw values are pushed to the backend once the transaction instance is complete. All raw values will be pushed to the backend, even if all iterations failed. The only case where some raw values could be prevented from being pushed to the backend is when a `TransactionAbortedException` is raised, which isn't likely to happen.

2 Password & Security

Writing plaintext passwords into a configuration file can sometime rise some security concerns. The Transaction Collector offer a mechanism to enforce security surrounding the usage of plaintext passwords. Passwords, in the configuration file, can either be plaintext or obfuscated using the **crypt-password.sh** tool located in the **bin** directory of your *APG* installation. When using the configuration GUI, the passwords are automatically obfuscated. Obfuscated **DOESN'T** mean that it is impossible for someone to retrieve the password, but it makes this operation somewhat trickier. The collector automatically handle plaintext and obfuscated passwords, without any external intervention.

6

Transaction Collector Configuration Example

Here is a simple example of a valid Transaction Collector configuration file. It lists some of the transactions and their different parameters:

```
<?xml version="1.0" encoding="UTF-8"?>
<transaction-collector xmlns="http://www.watch4net.com/APG/Collector/TransactionCollector"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.
watch4net.com/APG/Collector/TransactionCollector_transaction-collector.xsd">

  <collector>
    <source>demo-transaction-collector</source>
    <max-simultaneous-transactions>10</max-simultaneous-transactions>
    <timestamp-threshold>1000</timestamp-threshold>
    <pending-transaction-soft-limit>10</pending-transaction-soft-limit>
    <max-simultaneous-downloads-per-transaction>10</max-simultaneous-downloads-per-
transaction>
    <property-refresh-frequency>43200</property-refresh-frequency>
  </collector>
```

```

<!-- Can use comments everywhere -->
<defaults>
  <group>group</group>
  <period>60000</period>
  <delay>1000</delay>

  <properties>
    <property name="propertyName" value="Some_Value_Here" />
  </properties>
  <source-ip>0.0.0.0</source-ip>

  <output-directory>tmp</output-directory>
  <transaction-instances-output-to-keep>100</transaction-instances-output-to-keep>

  <http-request-headers>
    <property name="user-agent" value="Mozilla/5.0_(X11;_U;_Linux_i686;_en-US;_rv:1
.9.0.11)_Gecko/2009060308_Ubuntu/9.04_(jaunty)_Firefox/3.0.11" />
  </http-request-headers>
  <url-encoding>UTF-8</url-encoding>
</defaults>

<transaction xsi:type="dns-transaction">
  <id>dnstrans</id>
  <description>Transactions can have a description. It does impact nothing, but can help
describing the purposes or some tricky part about a transaction.</description>
  <property-ref>subset1</property-ref>
  <name>watch4net.com</name>
  <resolver hostname="10.128.7.89" />
  <timeout>1</timeout>
  <record-type>A</record-type>
</transaction>

<transaction xsi:type="http-head-transaction">
  <id>http-get-headers</id>
  <url>http://google.com</url>
  <http-request-headers inherit="false" />
</transaction>

<transaction xsi:type="http-transaction">
  <id>http-post</id>
  <url>http://watch4net.com</url>

```

```

    <request-type method="POST" />
    <http-request-headers inherit="false" />
  </transaction>

  <transaction xsi:type="http-transaction">
    <id>http-recursive-get</id>
    <url>http://cnn.com</url>
    <request-type recursive="true" />
    <http-request-headers inherit="false" />
  </transaction>

  <transaction xsi:type="meta-transaction">
    <id>metal</id>
    <description>Transaction sequence can have a description too. Can be very useful to
describe the behaviour of a sequence. </description>

    <transaction xsi:type="tcp-connect-transaction">
      <id>tcpapache1</id>
      <connection hostname="localhost" port="80" />
      <command>GET /get.php</command>
      <read match=".*" timeout="3000"/>
    </transaction>

    <transaction xsi:type="tcp-connect-transaction">
      <id>tcpapache2</id>
      <connection hostname="localhost" port="80" />
      <command>GET /get.php</command>
      <read match="tcp-.*" timeout="3000"/>
    </transaction>
  </transaction>

  <transaction xsi:type="ftp-transaction">
    <id>ftp-get</id>
    <connection hostname="localhost" port="21" username="ftp" password="" timeout="1"
passive-mode="true"/>
    <command type="RETR" remote-target="/onliner.txt" timeout="10" hash="
1234567890123456789012345678901234567890123456789012" match=".*" />
  </transaction>

  <transaction xsi:type="sftp-transaction">
    <id>sftp-get</id>
    <connection hostname="10.128.1.1" port="22" username="user" password="pass" timeout="

```

```
5000"/>
    <command type="get" remote-target="/etc/grub.conf" hash="61
d512db4912eaf2d482fd9f4419d732" />
</transaction>

<transaction xsi:type="radius-transaction">
    <id>radius-prod</id>
    <connection hostname="10.128.0.111" username="bob" password="hello" shared-secret="
testing123" authentication-type="chap" retries="0" timeout="2"/>
</transaction>

<transaction xsi:type="script-transaction">
    <id>script-poll</id>
    <script>bash /home/admin/scripts/rnd.sh</script>
</transaction>

<transaction xsi:type="sql-transaction">
    <id>mysql</id>
    <url>jdbc:mysql://127.0.0.1:53306/apg</url>
    <sql-driver>com.mysql.jdbc.Driver</sql-driver>
    <sql-user>apg</sql-user>
    <sql-pass>watch4net</sql-pass>
    <sql-query>select COUNT(id) from data_variable;</sql-query>
</transaction>

<transaction xsi:type="ldap-transaction">
    <id>ldap-connect</id>
    <connection principal="CN=Admin,OU=watch4net,O=net" credentials="ld4ppass"
authentication-type="none" url="ldap://10.128.1.1" timeout="1000" />
    <query search-scope="subtree" base="OU=watch4net,O=net" search-filter="(&!(
objectClass=*))" timeout="3000" dereference-links="false" />
</transaction>

<transaction xsi:type="sip-transaction">
    <id>siptrans1</id>
    <sip-address>sip:6000@10.128.2.18</sip-address>
    <local-port>7088</local-port>
    <sip-proxy-address>10.128.2.18:5090/UDP</sip-proxy-address>
    <timeout>30000</timeout>
    <pause>1000</pause>
</transaction>
```

```
<property-set id="subset1">
  <property name="property1" value="value1" />
  <property name="property2" value="value2" />
  <property name="property3" value="value3" />
</property-set>

<property-set id="subset2">
  <property name="propertyA" value="valueA" />
  <property name="propertyB" value="valueB" />
  <property name="propertyC" value="valueC" />
</property-set>

</transaction-collector>
```