InternetVista Web scenario documentation

Version 2.2

1 Table of content

2 Introduction to Web Scenario

This is the documentation for the internetVista web scenario definition language.

A scenario is formatted using XML syntax. This document describes how to build XML scenario that will be used by internetVista monitoring centers to monitor several steps in a web site.

Transactional monitoring, also known as web scenario monitoring, makes it possible to check a process that takes place over many pages on a web site. For example, this monitoring makes it possible to check the performance of a search process, a purchase process or even a registration process on an e-commerce site.

A web scenario is composed of a succession of web pages that are called step by step. Every step of the process depends on the proper functioning of the preceding steps.

Transactional monitoring correctly manages the sessions and cookies. It allows you to simulate the submission of data forms or even the navigation via hyperlinks. JavaScript and Ajax executions are supported by our monitoring centers.

Scenario monitoring will be considered as a success if all the steps are a success. In case of error, this monitoring will enable you to directly identify the problematic stage and to be informed about the cause of the problem. Moreover, you will be able to know the response times of every step by consulting the detailed data of the checks via the manager.

3 XML scenario description

3.1 General scenario structure

A scenario is a set of steps that will describe how to monitor a full web process. See Annex A for scenarios monitoring real cases and Annex B for a template of a scenario.

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario>
     <step>
          <!-- here will come step description -->
     </step>
     <step>
          <!-- here will come step description -->
     </step>
     <!-- ...                          -->
     <!-- x more steps will come here -->
     <!-- ...                          -->
</scenario>
```

There are 5 kinds of steps allowing monitoring typical website interactions:

- URL direct access – typically the first step of a scenario
- Sending a click on a specific page element (image, button, hyperlink, …)
- Sending an event on a specific page element accepting events (image, list, …)
- Hyperlink access – **obsolete**, use a "click" step in place
- Form submission – **obsolete**, use a "click" step in place

At the scenario level, user can define which browser behavior will be used for the monitoring. This can be changed using the attribute ***browserVersion***. The allowed values are: Firefox (default value), Chrome, IE8, IE11.

```
<scenario browserVersion="Chrome">… </scenario>
```

This attribute has an influence on the user-agent request header used for the request. When *browserVersion* is set, the monitoring system uses corresponding user-agent, otherwise "internetVista monitor (Mozilla compatible)" user-agent is used.

3.2 Steps structure

A step is always marked by a start tag <step> and a final tag </step>.

```
<step> …</step>
```

Each step can be identified by an optional attribute **name**; this name is used in the interface and in the alert to identify the step in error. It is not mandatory to give a name to a step, but it is recommended.

```
<step name="…">…</step>
```

By default, a step is based on the result of the previous step. This behavior can be changed using the attribute ***baseStepIndex*** or the attribute ***baseStepName*** at the step level. Those attributes can be defined to indicate that the current step will be based on the content of a specified step.

"baseStepIndex" refers to a previous step identified by its index and its value is a numeric value (e.g. 1, 2…). *"baseStepName"* refers to a step identified by its name.

```
<step baseStepIndex="2">…</step>
```

It is possible to make a pause before executing a step through the usage of the ***sleepBefore*** attribute. You can also decide that in case of failure, a step should be retried (***retry*** attribute). In case of retry, you can decide to sleep a bit before the retry has to be done (***sleepBeforeRetry*** attribute).

The sleep durations are expressed in milliseconds like in the following sample:

```
<step name="home" sleepBefore="5000" retry="1" sleepBeforeRetry="3000">
```

Following sections will describe the 5 typical steps used by the internetVista scenario. The mostly common used types are the first one and the second one.

3.2.1 URL step structure

An URL step is used to call a given URL:

```
<step>
      <url>http://....</url>
</step>
```

The URL inside the url tags can be http or https.

This kind of step is always used as the starting point of any scenario; but it can also be used at any moment in the scenario if required.

When the scenario must call an URL defined in the HTML page, the **inPage** attribute is used:

```
<url inPage="true">http://...</url>
```

In this case, our monitoring service will call the first URL found in the page starting with the URL you specified. The default value for this attribute is false.

3.2.2 Click step structure

A click step is used to simulate a click on any element present into an HTML page. It can be a click on a hyperlink, on a button, on an image or any other element. If the click requires JavaScript or Ajax interaction, and if JavaScript support is enabled (see *enableJavascript* common parameter), the JavaScript will be executed by the monitoring engine.

A click step will look like:

```
<step>
      <click xPathExpression="">
            …
      </click>
</step>
```

Inside a click step, several user actions can be defined. Those user actions will take place before the main click action is executed. Typical user actions are filling a field, checking a check box, selecting a value in a selection list, …

To identify the element that will receive the click action, *xPath* expressions are used.

3.2.2.1 Identification of an element using xPath expression

xPath is a language allowing to select elements in an XML document. So *xPath* allows to identify elements in an HTML page. See more on *xPath* at http://en.wikipedia.org/wiki/XPath.

To use *xPath* identification in a click step, you have to use the *xPathExpression* attribute like in the following example:

```
<click xPathExpression="//img[@id='123456']"/>
```

In this example, a click is done on the image having id="123456".

Modern browsers provide a feature to get the *xPath* value of an html element in a page, generally by right clicking on the element and by selecting “Inspect Element” action.

3.2.2.2 Executing user actions before main click action is executed

Before click action is executed and connection to the web server is done, several user actions can be executed. All those user actions should be wrapped inside a <userActions> element like in the following sample:

```
<step name="submit form">
      <click xPathExpression="//*[@id='submit_btn']">
            <userActions>
                  <fillField xPathExpression="//*[@id='name']" value="John"/>
                  <fillField xPathExpression="//*[@id='age']" value="50"/>
                  <selectDropdown xPathExpression="//*[@id='city']" value="Paris"/>
                  <selectRadioButton xPathExpression="//*[@id='radio1']"/>
            </userActions>
      </click>
</step>
```

Existing user actions are:

- fill a field with a particular value

```
<fillField xPathExpression="//*[@id='some id']" value="some value"/>
```

- select an element from a selection list. You can select the element by the display value or by the hidden technical value

```
<selectDropdown xPathExpression="//*[@id='some id']" text="text in the select (displayed value)" value="value in the select (hidden value)"/>
```

- activate a radio button element

```
<selectRadioButton xPathExpression="//*[@id='some id']"/>
```

- check a checkbox element

```
<checkCheckBox xPathExpression="//*[@id='some id']"/>
```

- uncheck a checkbox element

```
<uncheckCheckBox xPathExpression="//*[@id='some id']"/>
```

3.2.2.3 Filling a form before submit is done by a click action – obsolete, use user actions in place

A form allows submitting data to the server. A form can be identified by its name, by its id, or by its order in the page.

A form tag can have the following attributes:

- ***name**:* the name of the form to submit.
- ***id:*** the id of the form to submit.
- ***order:*** if the form does not have a name or an id the order attribute will indicate which form the system must submit. The order starts at 1.
- ***submit-hidden (true|false):*** if true, the system will submit all hidden fields of the form. Default value is true.
- ***submit-fields (true|false):*** if true, the system will submit the value of all fields (e.g. radio button, checkbox, …) defined in the form with the values already present in the downloaded page. Default value is true.
- ***action:*** by default, the system uses the action defined in the HTML form presents in the downloaded page but this action can be overridden in case of needs.

Inside a form tag, parameters can be defined. A parameter is a couple name-value described by the syntax:

```
<parameter name="username" value="test"/>
```

Here is a sample click step that fill a username field and password field before clicking the first submit button found in the page:

```
<step name="click-login">
      <click xPathExpression="//input[@type='submit']">
            <form name="loginForm">
                  <parameter name="username" value="test"/>
                  <parameter name="password" value="test"/>
            </form>
      </click>
</step>
```

3.2.3 FireEvent step structure

A *fireEvent* step is used to simulate an event on any element present into an HTML page that accepts events. Events can be: *load*, *focus*, *change*, *submit*… For instance, you may want to fire event "change" on a selection list. If the *fireEvent* requires JavaScript or Ajax interaction, and if JavaScript support is enabled (see *enableJavascript* common parameter), the JavaScript will be executed by the monitoring engine. If the event on an element submits a form and if user wants to fill the form before submit is done, he can provide information about the form in the *fireEvent* step structure.

A *fireEvent* step accepts the same attributes as the click step: xPathExpression, tagName, tagAttribute, attributeValue and form.

3.2.3.1 *Identification of an element using xPath expressions*

See 3.2.2.1

3.2.3.2 *Filling a form before submit is done by a fireEvent action*

See 3.2.2.2

Here is a sample *fireEvent* step that fires a *change* event on a selection list after selecting value "1000" in the pick list "zipCode":

```
<step name="change-zip-code">
      <fireEvent event="change" xPathExpression="//input[@id='zipCode']">
             <form name="form1">
                    <parameter name="zipCode" value="1000"/>
             </form>
      </fireEvent>
</step>
```

3.2.4 Anchor step structure – obsolete use click step in place

The anchor step is used to check a link is present in a downloaded page and to follow the url corresponding to the hyperlink. The possible attributes to identify the link in the page are:

- ***name:*** when the anchor is identified by a name, the system follows the hyperlink defined in this anchor.

```
<step>
  <a name="…"/>
</step>
```

- ***id:*** when the anchor is identified by an id attribute, the system follows the hyperlink defined in this anchor.

```
<step>
  <a id="…"/>
</step>
```

- ***href:*** when the anchor is not identified by a name or an id, you can use the attribute href to identify the anchor in the page. When using the href attribute, the system takes the first link found in the html code beginning with the string you specified in the href attribute.

```
<step>
 <a href="…"/>
</step>
```

3.2.5 Form step structure – obsolete use click step in place

A form step allows submitting a form to the server. A form can be identified by its name, by its id, or by its order in the page.

A form tag can have the following attributes:

- ***name**:* the system will submit the form identified by its name.
- ***id:*** the system will submit the form defined by its id.
- ***order:*** if the form does not have a name or an id the order attribute will indicate which form the system must submit. The order starts at 1.
- ***submit-hidden (true|false):*** if true, the system will submit all hidden fields of the form. Default value is true.
- ***submit-submit (true|false):*** if true, the system will submit the value of the submit button, in case submit button has a name. Default value is true.
- ***submit-fields (true|false):*** if true, the system will submit the value of all fields (e.g. radio button, checkbox, …) defined in the form with the values already present in the downloaded page. Default value is false.
- ***action:*** by default, the system uses the action defined in the html form presents in the downloaded page, but this action can be overridden, typically when the action is empty in the page and set by a javascript program.
- ***validate-parameters (true|false):*** the system will validate that each parameter is provided before submitting the form. If parameter validation is required be sure that the parameter that the scenario wants to add are existing in the html form in the downloaded page (can be input field, select field or textarea). Default value is false.

Inside a form tag, parameters can be defined. A parameter is a couple name-value described by the syntax:

```
<parameter name="username" value="test"/>
```

```
<step>
     <form name="loginForm" submit-hidden="true" submit-fields="true">
             <parameter name="username" value="test"/>
             <parameter name="password" value="test"/>
      </form>
</step>
```

3.2.6 Parameters common to all steps

At the step level, there are several commons parameters that apply to all kind of steps:

- ***maxDownloadSize:*** by default, the system download the first 256Kb of the HTML pages, this value can be overridden. A value of -1 indicates that no download limit is set and the whole page is downloaded. This field is expressed in bytes.

```
<maxDownloadSize>64000</maxDownloadSize>
```

- ***connectionTimeout:*** by default, the connection timeout (timeout to establish a connection with the physical server) is set to 30 seconds, this value can be overridden, and the value must be expressed in milliseconds.

```
<connectionTimeout>60000</connectionTimeout>
```

- ***responseTimeout:*** by default, the response timeout (timeout to get a response from application server once a connection is established) is set to 30 seconds, this value can be overridden, and the value must be expressed in milliseconds.

```
<responseTimeout>60000</responseTimeout>
```

- ***maxDuration:*** allows to define the maximum duration permitted for a check response. The value must be expressed in milliseconds.

```
<maxDuration>5000</maxDuration>
```

- ***matchingSentence:*** allows to define a matching sentence that must be present in the page content. If the page does not contain the matching sentence, the verification fails. This verification is done on the downloaded content (see also *maxDownloadSize* attribute) ...

```
<matchingSentence>hello world</matchingSentence>
```

- ***matchingSentenceOperator (contains|notcontains|regex):*** this operator is used in conjunction with the matching sentence entry. It allows specifying that the HTML content should contain or not the matching sentence, or that the matching sentence express a regular expression (see https://en.wikipedia.org/wiki/Regular_expression).

```
<matchingSentenceOperator>contains</matchingSentenceOperator>
```

- ***handleRefresh (true|false):*** allows to specify if the monitoring system must manage HTML meta tags doing a page refresh. Default is false.
  An example of refresh meta tag is:
  <meta http-equiv="refresh" content="2;url=http://www.internetvista.com/">
  This kind of tag, when used, is present at the beginning of the HTML page, in the <head> section.

```
<handleRefresh>true</handleRefresh>
```

- ***followHtmlRedirections (true|false):*** allows to specify if the system must follow a redirection defined in a refresh meta tag of the HTML code. Default is false.
  An example of refresh meta tag is:
  <meta http-equiv="refresh" content="2;url=http://www.internetvista.com/">

  This parameter is obsolete and has been replaced by *handleRefresh* parameter.

```
<followHtmlRedirections>true</followHtmlRedirections>
```

- ***username:*** it is used when the web server requires credentials for a basic authentication.

```
<username>monitoringUser</username>
```

- ***password:*** it is used when the web server requires credentials for a basic authentication.

```
<password>monitoringPassword</password>
```

- ***preemptiveAuthentication (true|false)***: in case of basic authentication (see *username* and *password* attributes), when preemptive authentication is enabled, we will send the credentials

at first request before the server gives an unauthorized response. Otherwise, we will wait an unauthorized response before sending the credential. By default, preemptive authentication is activated.

```
<preemptiveAuthentication>false</preemptiveAuthentication>
```

- ***enableJavascript (true|false)***: allows to decide if you want JavaScript and Ajax to be executed in case of needs. If JavaScript is enabled, all linked JavaScript files are also downloaded and are available for execution. The time needed to download linked JavaScript files are included in the total step download duration. Default is false.

```
<enableJavascript>true</enableJavascript>
```

- ***allowJavascriptError (true|false)***: allows to decide that in case of javascript execution error, the scenario should not fail and that the execution of the scenario should continue. Default is true.

```
<allowJavascriptError>false</allowJavascriptError>
```

- ***downloadCSS (true|false):*** allows to decide if the linked CSS files must be downloaded. The time needed to download linked CSS files is included in the total step download duration. Default is false.

```
<downloadCSS>true</downloadCSS>
```

- ***storeOriginalServerResponse (true|false)***: by default, the monitoring engine stores the result given by the server response; this result is used to validate the matching sentence mechanism. By setting this attribute to *false*, the default behavior can be changed in order to use the result given by the server response and the JavaScript execution. Typically, this can be useful after an Ajax call if you want to store the final result: Ajax execution and result integration in the HTML page. The default value of this attribute is true.

```
<storeOriginalServerResponse>false</storeOriginalServerResponse>
```

- ***clearCache (true|false):*** by default, the monitoring engine uses caching rules defined by your server during a whole scenario execution. If you need to not use caching rules and thus force the loading of all resources and all elements during the execution of each scenario step, you can set this attribute to true. Default is false.

```
<clearCache>true</clearCache>
```

- ***request-headers:*** allows to specify some specific value for HTTP request headers

```
<request-headers>
      <request-header name="Accept-Language" value="en-us,en,fr"/>
      <request-header name="Accept-Charset" value="UTF-8"/>
      <request-header name="User-Agent" value="Mozilla/5.0,…"/>
</request-headers>
```

- ***validateCertificate (true|false):*** allows to define if the monitoring engine must verify the Security Certificate installed on the server. This parameter is used in the case of HTTPS protocol. The expiration date of the certificate is checked. The default is true.

```
<validateCertificate>false</validateCertificate>
```

- ***sslProtocol:*** in case of https communication, this attribute allows to specify the SSL protocol that should be used in the negotiation process. Allowed values are TLSv1, TLSv1.1, TLSv1.2, SSLv3, SSLv2Hello. The default is TLSv1.

```
<sslProtocol>TLSv1.2</sslProtocol>
```

- ***clientCertificateFileName:*** allows to define a client certificate used during the authentication process. This field represents the file name of your certificate. You must contact the support team to set up the certificate.

```
<clientCertificateFileName>someCertificate.p12</clientCertificateFileName>
```

- ***clientCertificatePassword:*** if a client certificate has to be used, this is the password for the certificate.

```
<clientCertificatePassword>some password</clientCertificatePassword>
```

- ***excludeResources***: in case JavaScript is enabled or CSS download is enabled, you may want to avoid downloading some resources. Typically, you may not want Google analytics scripts to be downloaded and executed. You can exclude several resources and for every resource, you need to set the *url* and the *match* type. The *match* attribute can be “contains”, “equals”, “startsWith” and “endsWith”.

```
<excludeResources>
      <resource url="www.google-analytics.com" match="contains"/>
</excludeResources>
```

3.3 Use of variables and functions

In a scenario, a user can use variables and functions.

3.3.1 Variable

A variable is defined anywhere in the scenario and can be used after its definition. Its value can be a "constant" or compute via the call to a function.

The syntax is:

```
<variables>
      <variable name="variable_name" type="string" value="xxx"/>
</variables>
```

A variable has 3 attributes:

- ***name:*** name of the variable
- ***type:*** type of the variable. Only "string" at this time.
- ***value:*** initial value of the variable, it can be a constant or compute by a function.

After its definition, this variable can be used in any steps of the scenario. The syntax is:

```
<parameter name="xxxxxxxx" value="VARIABLE::variable_name::"/>
```

3.3.2 Function

A function can be used in a scenario to get computed data at runtime. The syntax of a function must respect the following pattern:

```
FUNCTION::<function_name>([arguments])::
```

Functions are case-sensitive. Here is a sample usage of a function, allowing to computer a form parameter value:

```
<parameter name="DEPART_DD" value="FUNCTION::FORMAT_DATE(TODAY(),'dd/MM/yyyy')::"/>
```

Available functions are:

- *Date TODAY()*

  returns the current date.

- *Date TODAY(String timezone)*

  returns the current date in the specified time zone. The parameter timezone can be: "Europe/Paris", "Europe/Brussels", "America/New_york", "Asia/Macau"… or "Etc/GMT+2", "Etc/GMT-6"… The default value is "UTC".

- *Long GET_TIMESTAMP()*

  returns a unique timestamp based on the current date and time. Useful to generate a unique Id, unique email address, …

- *Long GET_TIMESTAMP(Date date)*

  returns a unique timestamp based on the provided date.

- *Date GET_DATE(String date, String pattern)*

  returns a date object from a string date and a pattern. Patterns are Java patterns.

- *String FORMAT_DATE(Date date, String pattern)*

  returns a date formatted using specified pattern.

- *Date ADD_DAYS(Date date, Integer days)*

  adds a number of days to a date.

- *Date ADD_MONTHS(Date date, Integer months)*

  adds a number of months to a date.

- *Date ADD_YEARS(Date date, Integer years)*

  adds a number of years to a date.

- *Date REMOVE_DAYS(Date date, Integer days)*

  removes a number of days to a date.

- *Date REMOVE_MONTHS(Date date, Integer months)*

  removes a number of months to a date.

- *Date REMOVE_YEARS(Date date, Integer years)*

  removes a number of years to a date.

- Date CHANGE_TIMEZONE(Date dateToChange, String timezone)

  Convert the received date to the wanted timezone. Contact support to get the timezone code matching your needs.

- *String DECRYPT(String encryptedScript)*

  decrypts an encrypted string (used to hide a password for instance).

  The call to our encryption toolbox will allow you to encrypt a string (authentication is required before accessing URL).

  https://www.internetvista.com/manager/tools/webscenario-toolbox/

- *Integer ADD(Integer numberA, Integer numberB)*

  returns the sum of numberA and numberB.

- *Boolean CONTAINS(String sourceText, String textToFind)*

  returns true if *sourceText* contains *textToFind*, false otherwise. This function is useful when sourceText is provided by GET_CONTENT() function.

- *String GET_CONTENT()*

  returns the content of the downloaded page of the ***previous step*** of the scenario. This function returns nothing when used in the first step.

- *String IF(Boolean expression, String trueValue, String falseValue)*

  returns *trueValue* if expression is true, returns *falseValue* otherwise.

- *String URL_ENCODE(String textToEncode)*

  Encodes the *textToEncode* so it can be used inside an URL (GET request for instance).

- String GET_RANDOM_ALPHABETIC(Long length)

  Creates a random string whose length is the number of characters specified. Characters will be chosen from the set of alphabetic characters.

- String GET_RANDOM_ALPHANUMERIC(Long length)

  Creates a random string whose length is the number of characters specified. Characters will be chosen from the set of alpha-numeric characters.

- String GET_RANDOM_NUMERIC(Long length)

  Creates a random string whose length is the number of characters specified. Characters will be chosen from the set of numeric characters.

3.4 Annex A: Sample XML scenario

3.4.1 Login sample

This sample scenario allows verifying a login process on an extranet. The first step gives the URL and the second step will click the submit button and fill the form with authentication information.

```
<?xml version="1.0" encoding="utf-8"?>
<scenario>
      <step name="home">
            <url>http://extranet.mywebsite.com</url>
            <matchingSentence>Home</matchingSentence>
            <matchingSentenceOperator>contains</matchingSentenceOperator>
      </step>
      <step name="login">
            <click xPathExpression="//input[@type='submit']">
                  <userActions>
                        <fillField xPathExpression="//*[@name='user']" value="test"/>
                        <fillField xPathExpression="//*[@name='pwd']"
value="FUNCTION::DECRYPT('F2A56D4A6')::  "/>
                  </userActions>
            </click>
            <matchingSentence>Welcome on the extranet</matchingSentence>
            <matchingSentenceOperator>contains</matchingSentenceOperator>
      </step>
</scenario>
```

3.4.2 Checkout process sample

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario>
    <step name="home page">
        <url>http://www.my-ecommerce.com</url>
        <matchingSentence>My big deal product</matchingSentence>
        <matchingSentenceOperator>contains</matchingSentenceOperator>
    </step>
    <step name="select product">
        <click xPathExpression=="//*[@id='product1']" />
        <matchingSentence>Description of my product 1</matchingSentence>
        <matchingSentenceOperator>contains</matchingSentenceOperator>
        <maxDownloadSize>32000</maxDownloadSize>
    </step>
    <step name="add to shopping cart">
        <click xPathExpression="//input[@type='submit']">
            <userActions>
                <fillField xPathExpression="//*[@name='qty']" value="5"/>
            </userActions>
        </click>
        <maxDownloadSize>32000</maxDownloadSize>
        <matchingSentence>Total price</matchingSentence>
        <matchingSentenceOperator>contains</matchingSentenceOperator>
```

```
    </step>
    <step name="login">
        <click xPathExpression="//input[@type='submit']">
             <userActions>
                    <fillField xPathExpression="//*[@name='user']" value="test"/>
                    <fillField xPathExpression="//*[@name='pwd']"
value="FUNCTION::DECRYPT('F2A56D4A6'):: "/>
             </userActions>
        </click>
        <matchingSentence>Choose the delivery method</matchingSentence>
        <matchingSentenceOperator>contains</matchingSentenceOperator>
    </step>
    <step name="checkout process">
        <click xPathExpression="//*[@id='checkout_btn']">
             <userActions>
                    <fillField xPathExpression="//*[@name='deliveryMethod']" value="UPS"/>
                    <fillField xPathExpression="//*[@name='tracking']" value="true"/>
             </userActions>
        </click>
        <matchingSentence>Your order is pocessed</matchingSentence>
        <matchingSentenceOperator>contains</matchingSentenceOperator>
    </step>
</scenario>
```

3.5 Annex B: XML template for a new scenario

```
<scenario browserVersion="Firefox|Chrome|IE8|IE11">
      <step name="my first step" sleepBefore="0|x{ms}" retry="0|1|2"
sleepBeforeRetry="0|x{ms}">
            <url inPage="false|true">http://someUrl</url>
            <matchingSentence>some matching sentence</matchingSentence>
            <matchingSentenceOperator>contains|notcontains|regex</matchingSentenceOperator>
            <maxDownloadSize>-1|x{byte}</maxDownloadSize>
            <handleRefresh>false|true</handleRefresh>
            <enableJavascript>false|true</enableJavascript>
            <allowJavascriptError>true|false</allowJavascriptError>
            <downloadCSS>false|true</downloadCSS>
            <storeOriginalServerResponse>true|false</storeOriginalServerResponse>
            <clearCache>false|true</clearCache>
            <validateCertificate>true|false</validateCertificate>
            <sslProtocol>TLSv1|TLSv1.1|TLSv1.2|SSLv3|SSLv2Hello</sslProtocol>
            <clientCertificateFileName>certificate name</clientCertificateFileName>
            <clientCertificatePassword>certificate password</clientCertificatePassword>
            <username>username</username>
            <password>password</password>
            <preemptiveAuthentication>true|false</preemptiveAuthentication>
            <responseTimeout>x{ms}</responseTimeout>
            <connectionTimeout>x{ms}</connectionTimeout>
            <maxDuration>x{ms}</maxDuration>
            <submit submit-method="get|post">
                  <parameter name="some param" value="some value"/>
            </submit>
            <request-headers>
                  <request-header name="some_header" value="some_value"/>
             </request-headers>
            <excludeResources>
                  <resource url="some_url" match="equals|startsWith|endsWith|contains"/>
            </excludeResources>
            <variables>
                  <variable name="SOME_VARIABLE" type="string"
value="FUNCTION::GET_TIMESTAMP()::"/>
            </variables>
      </step>
      <step name="my second step" sleepBefore="0|x{ms}" retry="0|1|2"
sleepBeforeRetry="0|x{ms}">
            <click xPathExpression="//*[@id='some id']">
                  <userActions>
                        <fillField xPathExpression="//*[@id='some id']" value="some
value"/>
                        <selectDropdown xPathExpression="//*[@id='some id']" text="text
in the select (displayed value)" value="value in the select (hidden value)"/>
                        <selectRadioButton xPathExpression="//*[@id='some id']"/>
                        <checkCheckBox xPathExpression="//*[@id='some id']"/>
                        <uncheckCheckBox xPathExpression="//*[@id='some id']"/>
```

```
                        </userActions>
                </click>
                <matchingSentence>some matching sentence</matchingSentence>
                <matchingSentenceOperator>contains|notcontains|regex</matchingSentenceOperator>
                <maxDownloadSize>-1|x{byte}</maxDownloadSize>
                <handleRefresh>false|true</handleRefresh>
                <enableJavascript>false|true</enableJavascript>
                <allowJavascriptError>true|false</allowJavascriptError>
                <downloadCSS>false|true</downloadCSS>
                <storeOriginalServerResponse>true|false</storeOriginalServerResponse>
                <clearCache>false|true</clearCache>
                <validateCertificate>true|false</validateCertificate>
                <sslProtocol>TLSv1|TLSv1.1|TLSv1.2|SSLv3|SSLv2Hello</sslProtocol>
                <clientCertificateFileName>certificate name</clientCertificateFileName>
                <clientCertificatePassword>certificate password</clientCertificatePassword>
                <username>username</username>
                <password>password</password>
                <preemptiveAuthentication>true|false</preemptiveAuthentication>
                <responseTimeout>x{ms}</responseTimeout>
                <connectionTimeout>x{ms}</connectionTimeout>
                <maxDuration>x{ms}</maxDuration>
                <submit submit-method="get|post">
                        <parameter name="some param" value="some value"/>
                </submit>
                <request-headers>
                        <request-header name="some_header" value="some_value"/>
                 </request-headers>
                <excludeResources>
                        <resource url="some_url" match="equals|startsWith|endsWith|contains"/>
                </excludeResources>
                <variables>
                        <variable name="SOME_VARIABLE" type="string"
value="FUNCTION::GET_TIMESTAMP()::"/>
                </variables>
      </step>
</scenario>
```