EVEREST 2.0 - HTTP TEST TOOL

1. OVERVIEW

Everest is a regression testing tool for REST web services. It provides an efficient method for creating and maintaining large sets of tests over time.

2. INSTALATION

2.1. REQUIREMENTS

- Operational system:
  - Windows 9x or newer or Linux (kernel 2.4 or newer). The tool will also probably run on other OS supported by Python, but those are not recommended.
- Software:
  - Python 2.7.x (avoid versions 3.x). 32-bit version is recommended.

2.2. INSTALLING

1. If on Windows, add the Python installation directory (usually "C:\Python27\") to the PATH environment variable. Linux users have this by default.
2. Get the Everest 2.0 Package
3. At the PPL Test Tool directory, run the script EVEREST.PY from command line using Python as follows:

   ```
   python.exe everest.py
   ```

4. You should get a message starting with "Welcome to Everest Test Tool […]". If you get an error instead, make sure you've got the Python properly installed.

3. RUNNING

3.1. SINTAX

python.exe everest.py [–c config.cfg] [–t tag] testsets/test_script.xml

The runtests.py is the default executable for running testsets. Optional arguments are:

-c CONFIG_FILE	- Sets the config file to be used (Default: config.cfg)
-t TAG	- Optional tag selection (for only running testcases with specific tags)

Examples:

```
python.exe runtests.py testsets/sanity_check.xml
python.exe runtests.py -c staging_server.cfg testsets/test_script.xml
python.exe runtests.py -t smoke -t bugs testsets/test_script.xml
```

3.2. THE CONFIG FILE

The config file that is used describes basic target server information. The syntax is as follows:

[**SERVER**]
BASE-URL = **http://service.com/ppl-qa/**
PROXY = **web-proxy:8088**

All fields are required, the fields are as follows:

- **BASE-URL**: this is the base of URL in which all Test Request will extend. For example, a Request that has its path: "/data" will submit the request to "**http://service.com/ppl-qa**/data", joining both values.
- **PROXY**: If needed, a proxy can be set using <URL>:<PORT> (no "HTTP://"). If no proxy, you can leave the field blank or fill with "no" or "false".

3.3. RUNTIME FEEDBACK AND LOGS

The tool is fairly verbose, providing runtime feedback on test cases, requests, assertions and variables. The Runtime output has the following pattern:

Running [testset file]

* global variable set: [name] = [value]
---
* Running rest case "[Test Case Name]"
    * variable set: [name] = [value]
    * running request "[Request name]"… [OK/*FAILED*]
* Test Case "[name]" [PASSED/*FAILED*]
[…]
Test execution finished [successfully/with errors]
Reports are available at reports/[reports file location]

After the test is finished, the executable will exit with code 0 if successful or -1 if any case failed. This is useful if you're using the tool to validate the system and check if it's ok. Either way, report files are generated on the "\reports" folder.

3.4. REPORTS

By default, the test tool generated reports is two different formats: CSV (comma-separated values for quick Pass/Fail reference) and HTML (for a detailed execution report). Both files are generated after each execution on the folder "\reports" with the following file name pattern:

    YYYY-MM-DD HH-MM-SS [TestSet File].[CSV/HTML]

4. WRITING TESTS

Test scripting uses XML files and optional resources such as files and csv data.

4.1. THE TEMPLATE FILE

Templates can be created together or separately from test cases. Templates are the same as requests, except they aren't executed, instead, they are meant to be extended by requests to avoid repeated test data. To create templates, all you need is to create them within test XML files.

4.1.1. FORMAT

```
<TestSet>

        <Var name="global_name">value</Var>      <!-- Sets a global variable -->

        <Template id="template" extends="parent_template"> <!-- template for requests -->
        <!-- when extending a parent it receives all parent's data, any value overwrites the extended data -->
                <Desc>Template description</Desc>
                <Path full="false" base-url="default">test/</Path>
                <Method>GET</Method>
                <Body enctype="text/plain">
                        <field name="field">value</field>
                        <file name="name">file_from_resources.jpg</file>
                        optional request body text
                </Body>
                <Header name=" name">value</Header>
                <Var name="name" save_var="name" save_global="name">value</Var>
                <assert type="TypeOfAssertion" argument="value" inverse="false" save-var="name" save-
        global="name">expected value</assert>
                </Assertions>
        </Template>

        […]
</TestSet>
```

4.2. THE TESTSET FILE

4.2.1. FORMAT

```
<TestSet>
        <Desc>Optional testset description</Desc>

        <Include file="testset_file.xml" />   <!—includes the content of another testset file -->

        <Var name="global_name">value</Var>      <!-- Sets a global variable -->

        <Request id="request" extends="parent_template" use-certificate="false">
        <!-- requests outside a TestCase becomes a TestCase as well -->
        <!-- when extending a parent it receives all parent's data, any value overwrites the extended data -->
                <Desc>Request description</Desc>
                <Path full="false" base-url="default">test/</Path>
                <Method>GET</Method>
                <Body enctype="text/plain">
                <!-- if clear is true, any body set from parent is cleared -->
                        <field name="field">value</field>
                        <file name="name">file_from_resources.jpg</file>
```

```
                optional request body text
            </Body>
            <Header name=" name">value</Header>
            <Var name="test" save_var="name" save_global="name">value</Var>
            <assert type="AssertionType" argument="value" inverse="false" save_var="name"
        save_global="name" overwrite="false" save_file="filename">expected value</assert>
        </Request>

        <TestCase id="test case" csv="csvfile.csv">
            <Desc>Test Case Description</Desc>

            <Var name="name" save_global="global">value</Var>  <!—Variable within the TestCase -->

            <Request id="Request" extends="parent_template">
                […]
            </Request>
            […]
        </TestCase>

        […]
</ TestSet >
```

4.3. XML TAGS REFERENCE

4.3.1. VARIABLES

Variables can be created at any level and used in any tag value. There are three levels of variables:

```
<TestSet>
        <Var name="global_name">value</Var>
        <TestCase id="testcase">
            <Var name="testcase_var">value</Var>
            <Request id="request">
                <Var name="local_var">value</Var>
            </Request>
        </TestCase>
</TestSet>
```

Globals are usable in any context (even globals from Templates can be used on TestSets). They can be used/referenced by using **Value Scripting** (see section **4.5**). The Var tag format is as follows:

<**Var** name="**name**" save_var="name" save_global="name">**value**</**Var**>

The Var **name** argument is required and is used to reference the variable value.
Optional fields:

- **save-var="var_name"**: If any value is set, the value becomes the name of a new variable within the context of the current <TestCase> with the processed value of the current var. Default value: None.
- **save-global="global_name"**: If any value is set, the value becomes the name of a new variable in Global context with the processed value of the current var. Default value: None.

4.3.2. TEMPLATES

Templates are similar to Requests, except that they aren't executed on their own. Instead work as a reference for other Templates or Requests that extend them to copy their information. They aim to describe target services and allow the re-use of their information to avoid re-writing every request. The Template sintax is as follows:

```
<TestSet>
    <Template id="template" extends="parent_template">
        <Desc>Template description</Desc>
        <Path full="false" base-url="default">test/</Path>
        <Method>GET</Method>
        <Body enctype="text/plain">
            <field name="field">value</field>
            <file name="name">file_from_resources.jpg</file>
            optional request body text
        </Body>
        <Header name=" name">value</Header>
        <Var name="name" save_var="name" save_global="name">value</Var>
        <assert type="AssertionType" AssertionName argument="value" inverse="false"
save_var="name" save_global="name" overwrite="false">expected value</assert>

    </Template>
</TestSet>
```

Optional arguments include:

- **extends="parent"**: that contains the id of a parent Template to copy it's information. If used, fields that are set by the parent don't need to be set again, unless you want to overwrite them. Default: **None**
-

The child tag fields are:

- **Desc**: The value is it's description for reference and reporting
- **Path**: Target path that will be joined with the "BASE-URL" field from the config file to send the request. Optional attributes:
    - **full** indicates if it should consider the path as either partial or full. If true, it will ignore the base-url and use the path data as the full URL (example: http://www.google.com/) while the default value (false) consider the base-url+path (example: "/").
    - **base-url** uses one of the [OTHER-BASE-URL] set in the config file by name (default: **default**).
- **Method**: Can be either GET, POST, PUT or DELETE
- **Body**: The request body content, which can include plain text and/or fields and files. Optional attribute: **enctype** that can be be either *application/x-www-form-urlencoded*, *multipart/form-data* or *text/plain* (default: **text/plain**).
    - **Field**: Optional field that is appended to body in mimetype standard to simulate a POST form submission. Optional attribute **remove="false"** if set to **true**, clear any previously set Field by the parent. Required attribute: **name**
    - **File**: Optional field that includes the file content to body to simulate a file upload. Required attributes: **name** and **source** (to specify the filename).

- **TIP:** for writing a XML request body, use the CDATA delimiter to safely write XML within the XML test code.
  Example: <**Body**> <![CDATA[**<xml body>**]]> </**Body**>

- **Header**: required argument **name**=”name” defines the header name, and the tag value sets the value
- **Var**: required argument **name=”name”** sets the variable name, value sets the variable value. Optional arguments include **save-var=”name”** that saves the value to a test case variable with the given name and **save-global=”name”** that saves the value to a global variable with the given name.
- **Assertion**: It’s an assertion tag that describes a validation of the request; please refer to section **4.3.4** for information.

4.3.3. REQUESTS

Requests are calls made to the target server to submit information and validate (with assertions) its response. They can be one or more steps of a test case, where the request assertions define whether the Test Case will pass or fail (any failed assertion results in a failed test case). If not inside a **TestCase** tag, a test case exclusively to hold the request.

```
<TestSet>
        <TestCase id=”testcase”>
                <Request id=”template” extends=”parent_template”>
                        <Desc>Template description</Desc>
                        <Path>test/</Path>
                        <Method>GET</Method>
                        <Body>request body</Body>
                        <Header name=” name”>value</Header>
                        <Var name=”name” save-var=”name” save-global=”name”>value</Var>
                        <assert type=”AssertionType” argument=”value” inverse=”false” save-var=”name” save-
global=”name”>expect</assert>
                        </Assertions>
                </Request>
        </TestCase>
</TestSet>
```

The Request tag format is exactly the same as a template (so that it can extend templates). For field reference, refer to section **4.3.2** for tags and **4.3.4** for assertions specifically.

4.3.4. ASSERTIONS

Assertions are validations of the response for a **Request** (4.3.3), if any assertion fails to meet the expected value, the current test case fails. An Assertion is a direct comparison (equals expected) of a response value against an expected value using one of the Assertion Types available. The format is as follows:

<**assert** type=”AsssertionType” optional-arg=”value” inverse=”false” save-var=”var” save-global=”global” save-file=”filename”>expected</**assert**>

AssertionType indicates the type of Assertion, which can be one of the following:

- **Status**: Compares the response status with the expected value. No additional arguments.
  Example:

<**assert** type=”**Status**”>200</**assert**>

- **HeaderExists**: Verifies that the expected header name exists. No additional arguments.
  Example:
  < **assert** type=”**HeaderExists”**>header-name</ **assert** >

- **HeaderValueEquals**: Verifies that the value of the given response header (argument header) equals the expected value. Argument **header=”header name”** is required.
  Example:
  < **assert** type=”**HeaderValueEqual”** header=”header-name”>value</ **assert** >

- **HeaderValueContains**: Verifies that the value of the given response header (argument header) contains the expected value. Argument **header=”header name”** is required.
  Example:
  < **assert** type=”**HeaderValueContains”** header=”header-name”>value</ **assert** >

- **BodyAny**: Expects any content on the body response with length bigger than 0. Useful for saving the full response body content to a variable or file. No additional arguments.
  Example:
  < **assert** type=”**BodyAny”** />

- **BodyEquals**: Compares the response body with the given expected value. No additional arguments.
  Example:
  < **assert** type=”**BodyEquals”**>full body</ **assert** >

- **BodyContains**: Verifies that the response body contains the given expected value. No additional arguments.
  Example:
  < **assert** type=”**BodyContains”**>this text exists in the body</ **assert** >

- **BodyLengthEquals**: Verifies that the response body length matches the given value. No additional arguments.
  Example:
  < **assert** type=”**BodyLengthEquals”**>300</ **assert** >

- **BodyLengthLessThan**: Verifies that the response body length is smaller the given value. No additional arguments.
  Example:
  < **assert** type=”**BodyLengthLessThan”**>400</ **assert** >

- **BodyLengthMoreThan**: Verifies that the response body length is larger the given value. No additional arguments.
  Example:
  < **assert** type=”**BodyLengthMoreThan”**>100</ **assert** >

- **BodyXPathEquals**: Verifies that the first XPath query result from the argument **query=”xpath query”** equals the expected value. Additional argument **query** is required.
  Example:
  < **assert** type=”**BodyXPathEquals”** query=”//data”>12345</ **assert** >

- **BodyXPathContains**: Verifies that the first XPath query result from the argument **query=”xpath query”** has the expected value anywhere in its value. Additional argument **query** is required.
  Example:
  < **assert** type=”**BodyXPathContains”** query=”//data”>12</ **assert** >

- **BodyXPathAny**: Verifies that XPath query from in the tag value (not argument) returns any value other than none. No arguments, the query is in the tag value.
  Example:
  < **assert** type=”**BodyXPathAny”**>query</ **assert** >

- **BodyXPathCountEquals**: Verifies that the result from XPath query argument returns the number of results stated in the tag body. Additional argument **query** is required.
  Example:
  < **assert** type=”**BodyXPathCountEquals”** query=”//data”>2</ **assert** >

- **BodyXPathLessThan**: Verifies that the result from XPath query argument returns a number of results that is less than the value from tag body. Additional argument **query** is required.
  Example:
  < **assert** type=”**BodyXPathLessThan”** query=”//data”>5</ **assert** > <!—expect less than 5 results -->

- **BodyXPathMoreThan**: Verifies that the result from XPath query argument returns a number of results that is more than the value from tag body. Additional argument **query** is required.
  Example:
  < **assert** type=”**BodyXPathLessThan”** query=”//data”>2</ **assert** > <!—expect less than 2 results -->

- **BodyRegexEquals**: Verifies that the first regular expression query result from the argument **query=”regex query”** equals the expected value. Additional argument **query** is required.
  Example:
  < **assert** type=”**BodyRegexEquals”** query=”[A-Za-z0-9]”>12345</ **assert** >

- **BodyRegexAny**: Verifies that the first regular expression query result from the argument **query=”xpath query”** contains the expected value. Additional argument **query** is required.
  Example:
  < **assert** type=”**BodyRegexAny”** query=”[A-Za-z0-9]”>123</ **assert** >

- **BodyRegexCountEquals**: Verifies that the number of regular expression matches from the argument **query=”xpath query”** match the expected amount in the value. Additional argument **query** is required.
  Example:
  < **assert** type=”**BodyRegexCountEquals”** query=”[A-Za-z0-9]”>2</ **assert** >

- **BodyRegexLessThan**: Verifies that the number of regular expression matches from the argument **query=”xpath query”** match the expected amount in the value. Additional argument **query** is required.
  Example:
  < **assert** type=”**BodyRegexLessThan”** query=”[A-Za-z0-9]”>2</ **assert** >

- **BodyRegexMoreThan**: Verifies that the number of regular expression matches from the argument **query=”xpath query”** match the expected amount in the value. Additional argument **query** is required.
  Example:
  < **assert** type=”**BodyRegexMoreThan”** query=”[A-Za-z0-9]”>2</ **assert** >

- **BodyRegexCountEquals**: Verifies that the number of regular expression matches from the argument **query=”xpath query”** match the expected amount in the value. Additional argument **query** is required.
  Example:
  <**assert** type=”**BodyRegexEquals”** query=”[A-Za-z0-9]”>2</ **assert** >

- **BodyJsonPathAny**: Verifies that the JsonPath query from in the tag value (not argument) returns any value other than none. No arguments, the query is in the tag value.
  Example:
  <**assert** type=” **BodyJsonPathAny”**>*$.item.subitem.subsubitem*</ **assert** >
  * For reference on the JsonPath query language, please refer to http://goessner.net/articles/JsonPath

- **BodyJsonPathEquals**: Verifies that the one of the JsonPath query result from the query attribute equals the expected value. Additional argument **query** is required.
  Example:
  <**assert** type=” **BodyJsonPathEquals” query=”***$.item.subitem.subsubitem***”**>value</ **assert** >
  * For reference on the JsonPath query language, please refer to http://goessner.net/articles/JsonPath

- **bodyJsonPathContains**: Verifies that the one of the JsonPath query result from the query attribute contains the expected value anywhere in its value. Additional argument **query** is required.
  Example:
  <**assert** type=” **bodyJsonPathContains” query=”***$.item.subitem.subsubitem***”**>value</ **assert** >
  * For reference on the JsonPath query language, please refer to http://goessner.net/articles/JsonPath

- **Bodyjsonpathcountequals**: Verifies that the count of the JsonPath query results from the query attribute equals the expected value. Additional argument **query** is required.
  Example:
  <**assert** type=” **Bodyjsonpathcountequals” query=”***$.item.subitem.subsubitem***”**> 5 </ **assert** >
  * For reference on the JsonPath query language, please refer to http://goessner.net/articles/JsonPath

- **Bodyjsonpathcountmorethan**: Verifies that the count of the JsonPath query results from the query attribute is more than the expected value. Additional argument **query** is required
  Example:
  <**assert** type=” **Bodyjsonpathcountmorethan” query=”***$.item.subitem.subsubitem***”**> 2 </ **assert** >
  * For reference on the JsonPath query language, please refer to http://goessner.net/articles/JsonPath

- **Bodyjsonpathcountlessthan**: Verifies that the count of the JsonPath query results from the query attribute is less the expected value. Additional argument **query** is required.
  Example:
  <**assert** type=" **Bodyjsonpathcountlessthan" query="***$.item.subitem.subsubitem***"**> 1 </ **assert** >
  * For reference on the JsonPath query language, please refer to http://goessner.net/articles/JsonPath

- **BodyJsonPathBiggestAny**: Returns the large numeric value result from the JsonPath query. Will fail if no numeric values were found.
  Example:
  <**assert** type=" **BodyJsonPathBiggestAny"** > *$.item.subitem.subsubitem* </ **assert** >
  * For reference on the JsonPath query language, please refer to http://goessner.net/articles/JsonPath

Optional arguments for assertions are:

- **Inverse="boolean"**: if **true**, it inverts the result (for Pass to Fail and Fail to Pass) unless the failure is due to error (AssertName not found, for example). This is used mostly for testing of the tool itself, but can be used as assertions that expect failure rather than success. Default: **false**.
- **overwrite="boolean"**: by default, the same assertion name will overwrite the previous. For example, if the template sets a Status assertion that expects 201 and the child request sets a new one expecting 200, if overwrite equal true, only the Status for 200 will be maintained, if overwrite if set to false, both will still exist. Default: **False**.
- **save-var="var_name"**: will save the result (received value which is compared to the expected value, even on failure) to a test case variable with the name given in as value. Default: None.
- **save-global="global_name"**: will save the result (received value which is compared to the expected value, even on failure) to a global variable with the name given in as value. Default: None.
- **save-file="filename"**: will save the content received from the assertion to the given filename. The file is written in **resoures/save_files/<filaname>**. Default: None.

4.3.5. INCLUDE

Format:

<Include parent="my_template" repeat="false" file="testset_file.xml" **/>**
<!—include the content of tests/testset_file.xml -->

Include the content of a test set file into the current test set. This tag enables test sets that pack several smaller test sets, for example, a pack of all sanity test sets into one file that can be executed instead of all individually.

Optional Arguments:

- **parent**: defines a template named **parent** identical to the template indicated by its value (default_template="temp" means that the template "**parent**" is now a copy of "temp"). For example, including a generic testset, you can set the parent as the template you want the tests from this include to cover. **Default: None**
- **repeat**: If this file has already been imported, allows it to be included again or simply ignore it. This is often useful when the included file contains variables that are meant to replace existing ones multiple

times. On the other hand, not allowing it to repeat avoids content to be repeated or tested again. **Default: false**

4.3.6. LOOP FROM CSV RESOURCE FILE DATA

Format:

```
<TestCase id="name" csv="data.csv">
        <Var name="var_from_csv">{{ loop0 }}</Var>
        <Var name="second_var_from_csv">{{ loop1 }}</Var>
        <!—testcase content-->
</TestCase>
```

The csv argument from the TestCase tag allows a test case to be multiplied by the lines of a csv file from the resources/ directory. This is useful when you need to do several test cases that vary only a few values.

Each line of the csv file becomes one testcase and each column of the csv file becomes a variable with the name loop<**n**> where **n** is it's index (0, 1, 2, … - i.e. loop0, loop1, loop2, …).

4.3.7. SLEEP TIMER

Format:

```
<sleep>5</sleep> <!-- replace 5 with the amount of seconds to sleep -->
```

The sleep tag will make the test execution pause for a given amount of seconds. The value is in seconds and must be an integer. **Default: 0**

4.3.7. PRINTING MESSAGES

Format:

```
<print>hello world</print> <!—variables can be used here -->
```

The print tag allows you to print out messages during the test execution. This is a handy way to publish information before generating the report. The message will also appear on the HTML report details.

4.4. CALL FUNCTIONS

Call Functions are non-standard requests that use actual code from the within the test tool to execute tasks. Special cases are used for situations where only requests aren't enough to perform an action or run a test.

Call functions are regular Python functions that can be called from within the XML. Their return value can be stored in test variable. To call them, the sintax is as follows:

Example of a Special Case Template:

```
<CallFunction name="GenerateTag" source="scripts.py" save-var="var" save-global="var">
        <param name="name">value</param>
```

</**CallFunction**>

This will call the function “GenerateTag” Within the “tests/**scripts.py**” python script. The python script file follows the following format:

```
def GenerateTag(params={}):
    #code here
    return value_to_be_returned
```

If the script is not found or the function is not properly created, the CallFunction will fail as in any test case.

4.5. VALUE SCRIPTING

Value scripting allows the usage of variables and dynamic content within the value fields of all XML tags of Templates and TestSets. This is possible using a template-style scripting language named **Jinja2** that is executed on runtime.

To use variables on runtime (set by the XML test code) the tag value format is as follows:

```
<Tag>{{ variable }}</Tag>
```

This the **{{ variable }}** will be replaced by the variable (local, test case or global) with the name “variable” on runtime, just before the execution. This can be applied to any Tag value in the XML.

Advanced usage (including Ifs, includes and so on) can be found on the Jinja2 documentation on the web (http://jinja.pocoo.org/docs/).

Usage example:

```
<Var name=”test”>{{ test_item }}</Var>
<Body>
    <![CDATA[
    <Agreement>
        <Test>{{ anyvar }}</Test>
        {% if users %}
    <User>{{ user }}</User>
        {% else %}
    <User>None</User>
        {% endif %}
    </Agreement>
    ]]>
</Body>
```