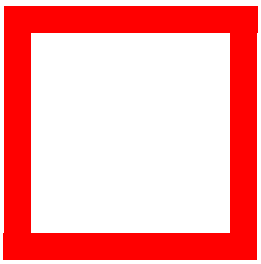
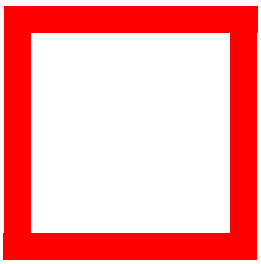
XTT Utility Guide

X-treme Testing Tool version 0.1

Created on 1 April 2009.



Contents



Introduction

This document describes the utilities classes including the parser, and the automated and manual testing methods.

It describes various utilities classes used by the XTT. You can use them during the development and testing process.

Target Audience

This document is for regular users who use XTT, for software testers who use XTT, and for developers who extend XTT.

This Guide, as well as the features it describes, is designed primarily for XTT system administrators. This Guide assumes that you are familiar with the following:

- XTT configuration
- Functions and features of XTT
- Standard aspects of operating software applications in the Window environment

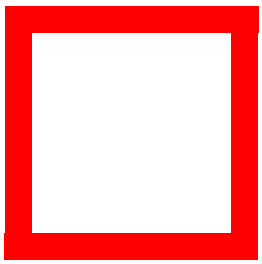
Related Documents

This Guide is part of the XTT v0.1 documentation set that includes the following documents:

- XTT Installation Guide
- XTT Function Modules
- XTT User Interface Guide

Summary of Changes

Issue	Date	Release Changes
v 0.1	1 April 2009	For XTT Release (Draft Version)



Utilities

Xtreme testing Tool uses different utilities classes. In ths chapter we will describe these utilities.

- Parser
- ConvertLib
- XTTProperties

Parser

Parser is responsible for converting a test xml file into single executable commands, which are executed from the FunctionModule classes.

Method	Details
runTest	run a single Test from a test xml document.
runTestNI	Same as runTest but with the exception that there will be no initialization of the modules which means connections stay open, variables keep their values etc
waitOnThreads	Wait for all threads created by <thread> to finish. Is called automatically at the end of the test.

ConvertLib

Method	Details
outputBytes	Convert a byte array in a String of bytes by using Integer.toHexString(). Examples: bytes={255,128,1}, start=0,len=3,vlength=2,divider=" " output="FF 80 01" bytes={0,128,0,255,0}, start=1,len=3,vlength=4,divider="" output="0080000000FF"
getStringFromByteString	Convert a byte string into a String.
getBytesFromByteString	Convert a byte string into an array of bytes by using Integer.decode(). Examples: input="FF8001" output bytes={255,128,1}
getByteArrayFromHexString	Convert any byte string into an array of bytes by using Integer.decode(). Examples: input="FF8001" output bytes={255,128,1} input="FF 80 01" output bytes={255,128,1} input="0xFF 0x80 0x01" output bytes={255,128,1}
intToHex	Convert an integer to string by using Integer.toHexString(). Examples: value="128" output="80"
longToHex	Convert an long to string by using Long.toHexString(). Examples: value="128" output="80"

Method	Details
longToHex	Convert an long to string by using Long.toHexString(). Examples: value="128",vlength=0 output="80" value="128",vlength=4 output="0080" value="65664",vlength=4 output="0080"
intToString	Convert an integer to string by using Integer.toString(). Examples: value="128" output="128"
intToString	Convert an integer to string by using Integer.toString(). Examples: value="128",vlength=0 output="128" value="128",vlength=4 output="0128" value="65664",vlength=4 output="5664"
addBytes	Add up all the bytes in the array.
getIntFromStringBytes	Get an integer from a String stored in a byte array. Returns Integer.parseInt(new String(bytes,start,len));
getStringFromCOctetByteArray	Get a String from a C-Style null(0x00) terminated byte array.;
getStringFromOctetByteArray	Get a String from a byte array.;
getCOctetByteArrayFromString	Get a C-Style null(0x00) terminated byte array from a String
getOctetByteArrayFromString	Get a byte array from a String
getIntFromByteArray	Get an integer from a byte array where the bytes contain the 4 bytes (32 bits) of the integer. bytes[0] contains the MSB, bytes[3] the LSB. BigEndian or network byte order.
getLongFromByteArray	Get an long from a byte array where the bytes contain the 8 bytes (64 bits) of the long. bytes[0] contains the MSB, bytes[7] the LSB. BigEndian.
getIntFromLittleEndianByteArray	Get an integer from a byte array where the bytes contain the 4 bytes (32 bits) of the integer. bytes[3] contains the MSB, bytes[0] the LSB. LittleEndian.
getByteArrayFromInt	Get a byte array from an integer where the bytes contain the 4 bytes (32 bits) of the integer. bytes[0] contains the MSB, bytes[3] the LSB.
getUIntVarFromInt	Create uIntVar according to WAP-230-WSP-20010705-a chapter 8.1.2 Variable Length Unsigned Integers.
getByteArrayFromInt	Get a byte array from an integer where the bytes contain the 4 bytes (32 bits) of the integer. bytes[0] contains the MSB, bytes[3] the LSB.

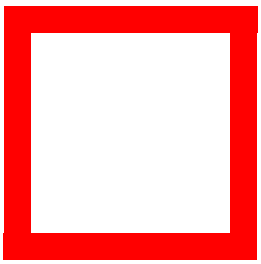
Method	Details
getByteArrayFromLong	Get a byte array from an long where the bytes contain the 8 bytes (64 bits) of the long. bytes[0] contains the MSB, bytes[3] the LSB.
queryString	Check if a String contains a certain substring via regular expression.. This stores the actuall groups under variable/matnumber/groupnumber So if you need to access group 0 of the first match it would be in variable/0/0 If you defined groups you'll find group 1 under variable/matchnumber (also variable/matchnumber/1) and if you have no groups then variable/matchnumber (also variable/matchnumber/0) will contain the full match.
queryStringNegative	Check if a String does not contain a certain substring via regular expression.
base64Encode	Converts a String into a Base64 encoded string.
createString	Creates a string of minimum length minlength filled with spaces
getStringFromGSM7bitAlphabet	Converts an array of bytes into a string with the characters taken from the GSM 7bit Aplhabet according to the byte values. See 3GPP TS 23.038 V7.0.0 (2006-03).
splitByteArray	splits the byte array around matches of the given delimiter. Examples: input : src = {0,1,2,'a','b','c',3,4,'a','b','c',5,6,7,8,9} delimiter = {'a','b','c'} output : {{0,1,2}, {3,4}, {5,6,7,8,9}} Returns: array of splited byte arrays
subByteArray	returns a byte array with the content of the given array starting from the specified start index. Examples: input : bytes = {0,1,2,3,4,5,6} start = 3 output : {3,4,5,6} Returns: a part of the byte array starting from specified start index

Method	Details
concatByteArray	Return a byte array concatenated from two or more byte arrays Examples: input : bytes1 = {0,1,2} bytes2 = {3,4,5} output : {0,1,2,3,4,5} Returns: the concatenated byte array

XTTProperties

Method	Details
getProperty	Returns a String value from an XML DOM Tree corresponding to the name argument. The name argument can point to subnodes of nodes by adding the '/' character This specifies the tree to search down, it can be as long as you want. If just a single word is specified, then the first occurance of that node will be returned. Get property first searches through the test specific configuration, before checking the global one. Returns "null" if not found
getIntProperty	Gets the property, but tries to make it into an Integer as well. Returns -1 if it's not found.
getLogFileName	Gets the path of the log file
showResults	Prints out a list of test with the corresponding results.
getNumberOfTests	Returns the number of tests loaded to run
setTestList	Sets the list of test to be run from the String[].
setTestList	Sets a single test to be run from the String.
setCurrentTestName	Sets the name of the current test.
getCurrentTestName	Gets the name of the current test. Returns the name of the file if no name is set.
setTestStatus	Sets the current test status.
readFile	Returns the contents of the file as a String. If the file is empty, or doesn't exist a blank String is returned
setTracing	Set the tracing level. Sets the tracing level based of the String interpretation of the level.

Method	Details
printExternal	Print out 'External' information. 'External' level is printed as long as tracing/disableexternal isn't added in the config. Also return true if External tracing is seen or logged
printDebug	Print out 'Debug' information. 'Debug' level is printed if the tracing level Debug Also return true if DEBUG tracing is seen or logged
printVerbose	Print out 'Verbose' information. 'Verbose' level is printed if the tracing level is Verbose, or Debug Also return true if VERBOSE tracing is seen or logged
printInfo	Print out 'Info' information. 'Info' level is printed if the tracing level is Info, Verbose, or Debug Also return true if INFO tracing is seen or logged
printWarn	Print out 'Warn' information. 'Warn' level is printed if the tracing level is Warn, Info, Verbose, or Debug Also return true if WARN tracing is seen or logged
printFail	Print out 'Fail' information. 'Fail' level is printed if the tracing level is Fail, Warn, Info, Verbose, or Debug Also return true if FAIL tracing is seen or logged
canEnd	Returns true when the program is ready to end, otherwise it returns false



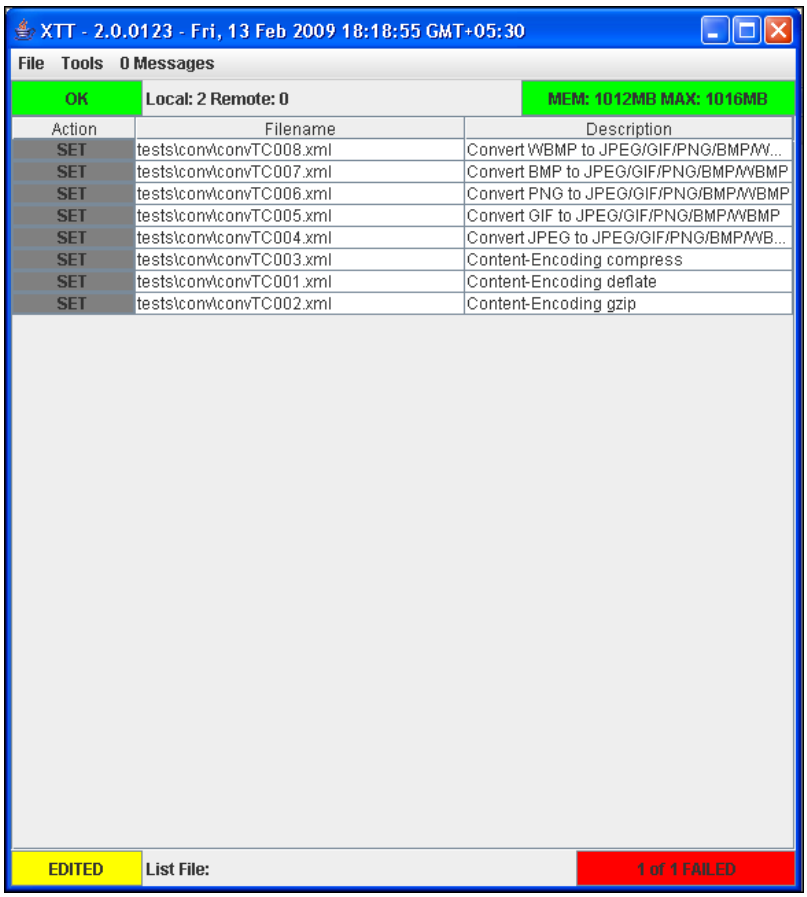
Testing

Using the Xtreme Testing Tool (XTT), test can either be executed manually using the **Execute Command** option in the **Tools** menu, or you can run the automated test scripts. The automated test scripts are imported to the XTT application console, and then executed as an individual test case or as a list file.

The automated test scripts are written using XML.

A test case is a detailed procedure that fully tests a feature or an aspect of a feature. Whereas the test plan describes what to test, a test case describes how to perform a particular test. You need to develop a test case for each test listed in the test plan.

XTT Console


XTT - 2.0.0123 - Fri, 13 Feb 2009 18:18:55 GMT+05:30
File Tools 0 Messages
OK
Local: 2 Remote: 0
MEM: 1012MB MAX: 1016MB
Action
Filename
Description
SET tests\conv\convTC008.xml Convert WBMP to JPEG/GIF/PNG/BMP/W...
SET tests\conv\convTC007.xml Convert BMP to JPEG/GIF/PNG/BMP/WBMP
SET tests\conv\convTC006.xml Convert PNG to JPEG/GIF/PNG/BMP/WBMP
SET tests\conv\convTC005.xml Convert GIF to JPEG/GIF/PNG/BMP/WBMP
SET tests\conv\convTC004.xml Convert JPEG to JPEG/GIF/PNG/BMP/WB...
SET tests\conv\convTC003.xml Content-Encoding compress
SET tests\conv\convTC001.xml Content-Encoding deflate
SET tests\conv\convTC002.xml Content-Encoding gzip
EDITED
List File:
1 of 1 FAILED

Example 1

Test Case Summary: Send HTTP get request through XMP

Prerequisites

- XMP v4.1 should be properly configured, up and running.
- XMP v4.1 workflow constants file should be properly configured and loaded into XMP.
- XMP v4.1 workflow file should be loaded into XMP.
- Other various dynamic configuration files should be properly configured and loaded into XMP.

Test Case

Test steps are as following:

1. Log on to the XMP server as radius user.
2. Start the web server.
3. Send HTTP get request through XMP.
4. Check the HTTP response code/message.
5. Stop the web server.
6. Log off as radius user from the XMP server.

Automated Test Case

```
<test>
<function name="testName" module="Basic">
   <parameter>CONNECT.TC.001</parameter>
   </function>
   <description>A gnu.org GET with radius logon/logoff</description>
<function name="radiusLogon" module="Radius">
   <parameter>xttuser</parameter>
   <parameter>12346</parameter>
   </function>
<function name="sendGetRequest" module="HTTP">
   <parameter>http://www.gnu.org/</parameter>
   </function>
<function name="checkResponseCode" module="HTTP">
   <parameter>200</parameter>
   </function>
<function name="radiusLogoff" module="Radius">
   <parameter>xttuser</parameter>
   <parameter>12346</parameter>
   </function>
   </test>
```

Example 2

Test Case Summary: HTTP GET / Virus Check is enabled / Requested web page contains virus (eicar) / XMP blocks.

Prerequisites

- XMP v4.1 should be properly configured, up and running
- XMP v4.1 workflow constants file should be properly configured and loaded into XMP
- XMP v4.1 workflow file should be loaded into XMP
- Other various dynamic configuration files should be properly configured and loaded into XMP.
- Kaspersky Virus Check is properly installed and running.

Test Case

Test steps are as following:

1. Kaspersky Virus Check is enabled.
2. HTTP Client User Agent sends HTTP GET Request to XMP.
3. XMP sends the request to the Target Web Server.
4. Target Web Server responds with a web page which contains a virus (eicar string could be used for testing purposes).
5. XMP blocks the web page responded by the Target Web Server and it might potentially send a warning page or HTTP response code to the HTTP Client User Agent.

Automated Test Case

```
<test>
<function name="testName" module="Basic">
   <parameter>KasperskyVirusScanningIntegration.TC.004</parameter>
   </function>
   <description>Requested web page contains virus (eicar) / XMP
   blocks</description>
<function name="radiusLogon" module="Radius">
   <parameter>xttuser</parameter>
   <parameter>12346</parameter>
   </function>
   <function name="startWebServer" module="HTTP" />
<function name="setHeader" module="HTTP">
   <parameter>somevalue</parameter>
   </function>
<function name="sendGetRequest" module="HTTP">
<parameter>
   <parameter>http://</parameter>
   <configuration>system/ip</configuration>
   <parameter>/viruspage.html</parameter>
   </parameter>
   </function>
<function name="checkResponseCode" module="HTTP">
   <parameter>302</parameter>
   </function>
   <function name="stopWebServer" module="HTTP" />
<function name="radiusLogoff" module="Radius">
   <parameter>xttuser</parameter>
   <parameter>12346</parameter>
   </function>
   </test>
```

Example 3

Test Case Summary: Test Case :Validate PROPFIND Request and its Response are successfully proxied by XMP (RFC-4918 / Section 9.1)

Prerequisites

- XMP v4.1 should be properly configured, up and running
- XMP v4.1 workflow constants file should be properly configured and loaded into XMP
- XMP v4.1 workflow file should be loaded into XMP
- Other various dynamic configuration files should be properly configured and loaded into XMP.

Test Case

Test steps are as following:

1. WebDav Client sends PROPFIND HTTP Proxy Request to XMP.
2. XMP forwards PROPFIND Request to the target WebDav Server.
3. The Target WebDav Server sends a success response code and response payload (see RFC-4918 / Section 9.1) to XMP.
4. XMP forwards the response to the WebDav Client.

Automated Test Case

```
<test>
   <name>WebDAVSupport.TC.001</name>
   <description>WebDAV - PROPFIND</description>
   <function name="startWebServer" module="HTTP" />
<function name="setServerHeader" module="HTTP">
   <parameter>content-type</parameter>
   <parameter>text/xml; charset="utf-8"</parameter>
   </function>
<function name="setServerReturnCode" module="HTTP">
   <parameter>207</parameter>
   </function>
<function name="setServerReturnMessage" module="HTTP">
   <parameter>Multi-Status</parameter>
   </function>
<function name="setCacheFile" module="HTTP">
   <parameter>/file</parameter>
<parameter>
   <crlf><?xml version="1.0" encoding="utf-8" ?></crlf>
   <crlf><D:multistatus xmlns:D="DAV:"></crlf>
   <crlf><D:response></crlf>
   <crlf><D:href>http://www.foo.bar/file</D:href></crlf>
   <crlf><D:propstat></crlf>
   <crlf><D:prop xmlns:R="http://www.foo.bar/boxschema/"></crlf>
   <crlf><R:bigbox></crlf>
   <crlf><R:BoxType>Box type A</R:BoxType></crlf>
   <crlf></R:bigbox></crlf>
   <crlf><R:author></crlf>
   <crlf><R:Name>J.J. Johnson</R:Name></crlf>
   <crlf></R:author></crlf>
   <crlf></D:prop></crlf>
   <crlf><D:status>HTTP/1.1 200 OK</D:status></crlf>
   <crlf></D:propstat></crlf>
   <crlf><D:propstat></crlf>
   <crlf><D:prop><R:DingALing/><R:Random/></D:prop></crlf>
   <crlf><D:status>HTTP/1.1 403 Forbidden</D:status></crlf>
   <crlf><D:responsedescription> The user does not have access to the
   DingALing property.</crlf>
```

```
   <crlf></D:responsedescription></crlf>
   <crlf></D:propstat></crlf>
   <crlf></D:response></crlf>
   <crlf><D:responsedescription> There has been an access violation
   error.</crlf>
   <crlf></D:responsedescription></crlf>
   <crlf></D:multistatus></crlf>
   </parameter>
   </function>
<function name="setTextPostData" module="HTTP">
<parameter>
   <crlf><?xml version="1.0" encoding="utf-8" ?></crlf>
   <crlf><D:propfind xmlns:D="DAV:"></crlf>
   <crlf><D:prop xmlns:R="http://www.foo.bar/boxschema/"></crlf>
   <crlf><R:bigbox/></crlf>
   <crlf><R:author/></crlf>
   <crlf><R:DingALing/></crlf>
   <crlf><R:Random/></crlf>
   <crlf></D:prop></crlf>
   <crlf></D:propfind></crlf>
   </parameter>
   </function>
<function name="setHeader" module="HTTP">
   <parameter>Content-Type</parameter>
   <parameter>text/xml; charset="utf-8"</parameter>
   </function>
<function name="sendFreeRequest" module="HTTP">
   <parameter>PROPFIND</parameter>
   <parameter>true</parameter>
<parameter>
   <parameter>http://</parameter>
   <configuration>SYSTEM/IP</configuration>
   <parameter>:</parameter>
   <configuration>WEBSERVER/PORT</configuration>
   <parameter>/file</parameter>
   </parameter>
   <parameter>response</parameter>
   </function>
```

```
<function name="sendGetRequest" module="HTTP">
<parameter>
   <parameter>http://</parameter>
   <configuration>SYSTEM/IP</configuration>
   <parameter>:</parameter>
   <configuration>WEBSERVER/PORT</configuration>
   <parameter>/file</parameter>
   </parameter>
   </function>
<function name="waitForRequests" module="HTTP">
   <parameter>1</parameter>
   </function>
   <function name="stopWebServer" module="HTTP" />
   </test>
```

Automated testing using the the test scripts

The test scripts written are saved in one of the project folders. For example, a folder named test is created within the XTT

Starting the XTT Application

To start XTT application, follow these steps:

1. Click **Start** and then click **Run**.
2. Enter `cmd` and click **OK**. The Command Prompt appears.
3. Navigate to the folder where you have extracted the XTT package.
4. Set the following paths at the command prompt if you are using a Windows computer:

```
set JAVA_HOME="<Your JDK installed folder>"
set PATH="<Your JDK installed folder>\bin";%PATH%
```

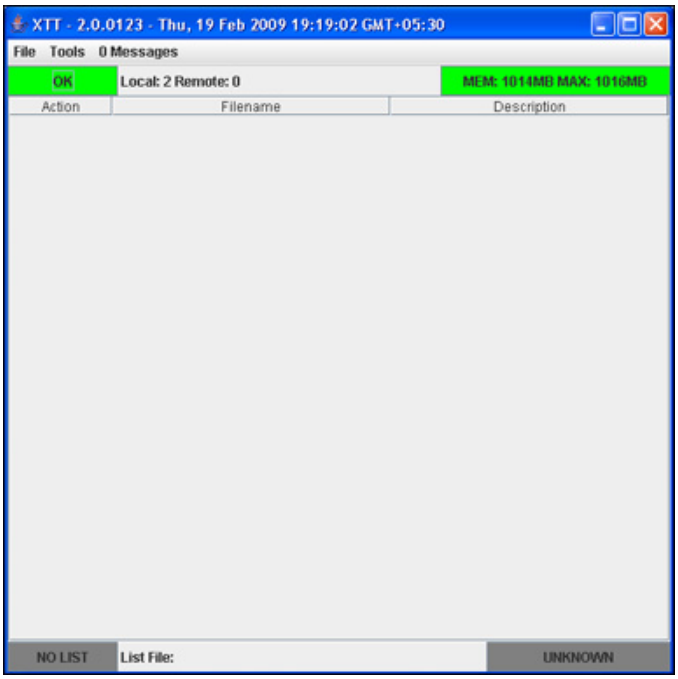
OR

Set the following paths at the command prompt if you are using a UNIX computer:

```
export JAVA_HOME="<Your JDK installed folder>"
export PATH="<Your JDK installed folder>/bin":$PATH
```

5. Enter `java -cp lib\jdom.jar -jar lib\xtt.jar -g` at the command prompt and press **Enter**.

   You see the **XTT** home screen.

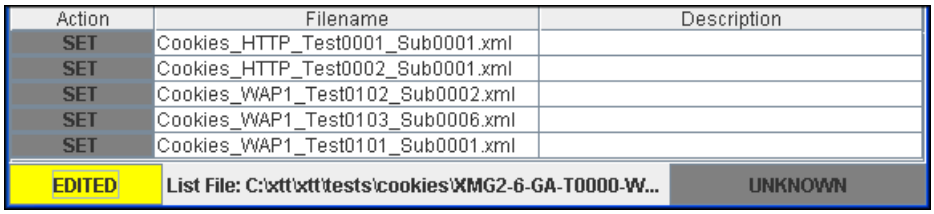


Adding XML files to the XTT console

1. Click on the **No List** button at the bottom left corner of the XTT console.
2. Click on the **Add Test** option. A dialog box appears.
3. Browse through to the appropriate folder containing the XML files
4. Select the XML file you want to load, and click **Open**. You can add one file at a time.

   The XML file can now be seen in the XTT console. The **No List** gray button now turns yellow with label changing to **Edited**. Likewise you can add more files to the XTT console.

   To run the XML file, there are several options available.



5. Click on the individual script and click **Run**. This command applies only to the selected script.
6. Alternatively, click on the **Edited** button, and click the **Run List** or **Run Selected** option.

   Run List will run all the XML files whereas **Run Selected** will run only the selected XML file(s).

Adding List files to the XTT console

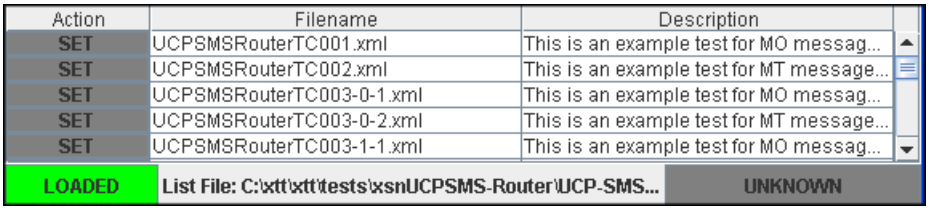
1. Click on the **File** menu.
2. Click on the **NewLoad Test List** option. A dialog box appears.
3. Browse through to the appropriate folder containing the list files.
4. Select the list file you want to load, and click **Open**.

Note: A list file contains a set of xml scripts. It contains all the test script path along with the test script name i.e. tests\XTT_AGW\MMStoHTTPRetrievalTC022.xml.

Sample test list file:

```
tests\XTT_AGW\MMStoHTTPRetrievalTC022.xml
tests\XTT_AGW\sendMM1TC003.xml
tests\XTT_AGW\sendMM1TC003.xml
```

The XML files can now be seen in the XTT console. The **No List** gray button now turns green with label changing to **Loaded**.



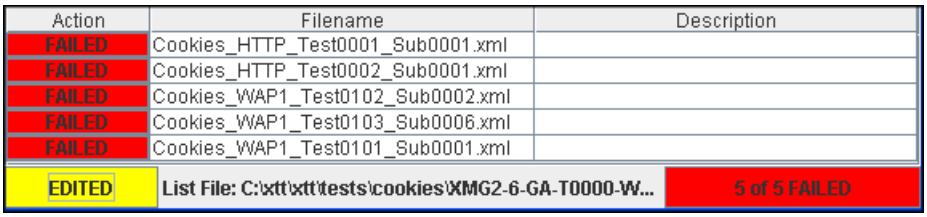
Likewise you can add more files to the XTT console.

To run the XML file, there are several options available.

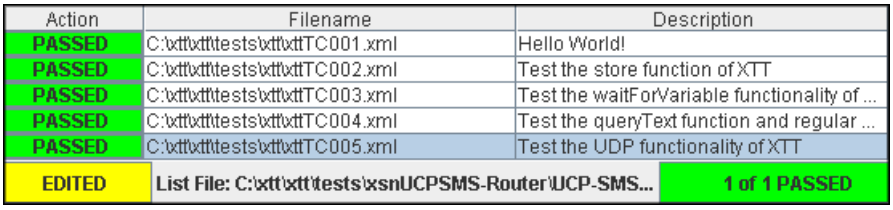
5. Click on the individual script and click **Run**. This applies only to the selected script.
6. Alternatively, you can click on the **Loaded** button, and then click on the **Run List** option. This applies to all the scripts that are loaded in the XTT.

The test scripts either Pass or Fail to execute. The status of the result can be seen on the individual script.

The **Unknown** grey colored button changes to red color and it reads **Failed** if the script failed the test.



Similarly, if the script passes the test, the **Unknown** grey colored button changes to green color and it reads **Passed.**



Viewing Logs

To view log of the test scripts, you can:

- Click on the **Passed/Failed** button

  If you have run the list, then you can see the logs of all the tested scripts in one window. If you have run an individual script, then you can see the log of the last test case that was run.

- Click on the **Edited/Loaded** button

  If you have run the list, then you can see the logs of all the tested scripts in one window. If you have run an individual script, then you can see the log of the last test case that was run.

- Click on the individual script and click Log

  It allows you to see the last log of the selected scripts only.

When you click on an individual test case, a list with the following options is displayed:

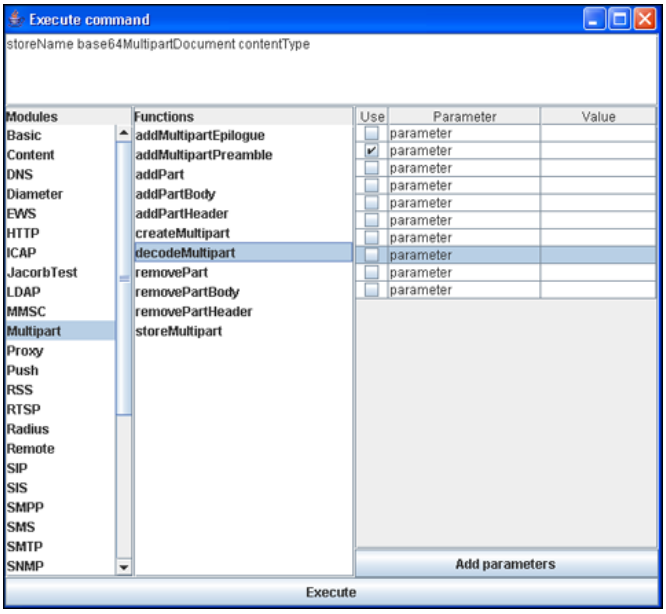
Options	Description
Run	Use this to execute the test.
Edit	This option allows you to edit the test script.
Set	Sets (or selects) the test script.
Log	This allows you to view the last log of the selected test case.
Remove	This option allows you to remove the test case from the list.

When you right-click on an individual test case, a list with the following options is displayed:

Options	Description
Run Selected	Use this to execute the selected tests.
Run List	Use this to run all the test cases available in the XTT console.
Remove Selected	This option allows you to remove the selected tests from the list.

Manual Testing using the GUI

When you run a test case, XTT executes all the functions which are available in a specified function module that is used in a particular test case. Using the Execute Command option, you can select and execute a specific function of the selected Module by providing appropriate values for the parameters.



To Execute Command, follow these steps:

1. Click on the **Tools** menu.
2. Click on the **Execute Command** option. The **Execute Command** window appears.
3. Select a **Module** in the left column. The **Functions** of the selected **Modules** appear in the right column.
4. Select a **function** in the right column. The values are displayed on the top left corner of the window.
5. Enter **Value** against the **Parameter**.

   To add additional parameter rows, click on the **Add Parameters** button.
6. Click **Execute**.

Note: If you enter the wrong argument or parameter and execute command, a failed message is displayed in red.

An illustration of how to assign appropriate values to the parameters, follow these steps:

1. Go to the **XTT** folder of your local computer and double-click the **doc** folder.
2. In the **doc** folder, open the **index.html** file in your browser.
3. In the **All Classes** column in the left side of the window, click on the **FunctionModule_UCP** link.

4. Scroll down to the **Method Summary** section of the page.
5. In the XTT application, open the **Execute Command** window.
6. Compare the **Functions** in the **Execute Command** window with those in the **Method Summary** section.
7. Enter the appropriate **Value** against the corresponding **Parameters** in the **Execute Command** window.
8. Click **Execute** in the **Execute Command** window.