Script-based Analyzer Tool V2.0

For WS-I profiles BP12, BP20, RSP10.

Authors:
Jacques Durand (Fujitsu) jdurand @ us . fujitsu .com
Tom Rutt (Fujitsu) Tom.Rutt @ us . fujitsu .com
Craig Chaney (IBM) craigcw @ us .ibm. com

Introduction:

This package contains the new version of the "Analyzer" test tool, one of the two components (Monitor and Analyzer) that compose the test tools package of the WS-I testing tools developed for Basic Profile (BP) 1.2, Basic Profile (BP) 2.0 and Reliable, Secure Profile (RSP) 1.0.

The approach in testing the new generation of profiles, is to rely [as much as possible] on an executable scripting of Test Assertion (TA), so that a generic Analyzer tool can be reused for subsequent profiles. This approach assumes in turn that all test inputs have been formatted in XML.

For this purpose, an enhanced version (but not drastically different) of the former "Monitor log file" (the test input file for the Analyzer) has been defined and used by this package. The test input file, also called the "input log file" is consolidating test artifacts as diverse as:

- service definition files (WSDL, Schemas)
- HTTP and MIME message headers
- SOAP envelopes and content

Also, the scripting of TAs in a language that is widely used - such as XPath – makes it easier for different parties (profile writers, software developers, users) to have a common understanding of what the test tools will check for, and ultimately of what concrete interpretation is given to profile requirements.

The authors have been actively involved in definition and scripting of Test Assertions for BP1.2, BP2.0 and RSP1.0. The scripting of test assertions is expected to be XPath 2.0 (many test assertions could not be expressed using XPath 1.0.)
This Analyzer tool has been designed to support XPath 2.0 test assertions, although it could as well be used for TAs written in XPath 1.0.

The Analyzer core module itself has been developed using XSLT 2.0. This version of XSLT (XSLT 2.0) had the right features for supporting not only XPath 2.0, but also some advanced TA execution functions such as TA execution ordering based on the prerequisite relationship (see definition in previous test tools). As a result, this package represents a modest amount of code. It can be run using any XSLT 2.0 processor, although some advanced features used in some TAs (like schema validation) may not be available in every XSLT product on the market.

This package works in two phases, the first phase only needed once for each new profile:

Phase 1: Generation of the Analyzer script for a particular profile.

Input: the Profile definition document with embedded TA scripts. (example: **BasicProfile-2.0.xml)**
Output: another XSLT 2.0 script that represents the actual Analyzer tool for this profile. (named here: **taplugins.xsl)**
Processor: a code generator written in XSLT 2.0. (named here: **make-plugins.xsl)**

The command line operation (using an xslt processor called here "xslt") is:

>**xslt make-plugins.xsl BasicProfile-2.0.xml taplugins.xsl**

Phase 2: Run-time analysis of Web services artifacts.

Input: a test log file. (example: **testlog.xml**)
Output: a test Report file in the same format as the previous generation of test tools V1.0, so that the same HTML rendering XSLT script can be reused. (example: **testreport.xml)**
Processor: the Analyzer XSLT 2.0 script produced by Phase 1 for this profile, used with a driver script. (name: **tadriver.xsl**)

The command line operation (using an xslt processor called here "xslt") is:

>**xslt tadriver.xsl testlog.xml testreport.xml**

Files in this package:

TA definitions:

- **(Depending on the Profile tested) BasicProfile-1.2.xml or BasicProfile-2.0.xml or ReliableSecureProfile-1.0.xml:** a sample profile document with inline Test Assertions (TAs). Only a subset of TAs have been represented in it, for testing purpose. Just for the sake of testing the TAs themselves, an XML attribute has been added to disable/enable the TAs:
  - testAssertion/@enable=”false” means the TA will not be processed.
  - testAssertion/@enable =”true” or is absent, means the TA will be processed.

- **ProfileDocToHtml-2.0.xsl:** xslt for HTML rendering of above profile definition and embedded test assertions (TAs).

Sample input files containing test targets:

- **testlog.xml:** a sample consolidated file with descriptions (WSDL with RPC binding) and messages log.
- **log.xsd:** schema for the test log file.

“Permanent” XSLT scripts (do not change from one profile to the other):

- **make-plugins.xsl:** A script that generates an xsl transform script (called “TA plugins”)that contains an executable representation of TAs in the profile definition file.

- **tadriver.xsl:** A script that drives the analysis work over some test input. Invokes (imports) the TA plugins scripts generated in Phase 1.

- **genreport.xsl:** A script imported by tadriver.xsl that generates a final report (here named testreport.xml) viewable using the existing xslt (from tools V1) for HTML browsing of report (report.xsl).

- **common.xsl:** A script reused from previous WS-I material, imported for HTML rendering.

- **xslt.sh:** a shell script for executing xsl scripts using Saxon (Unix envt). Make sure to set your environment variable $SAXON_HOME.

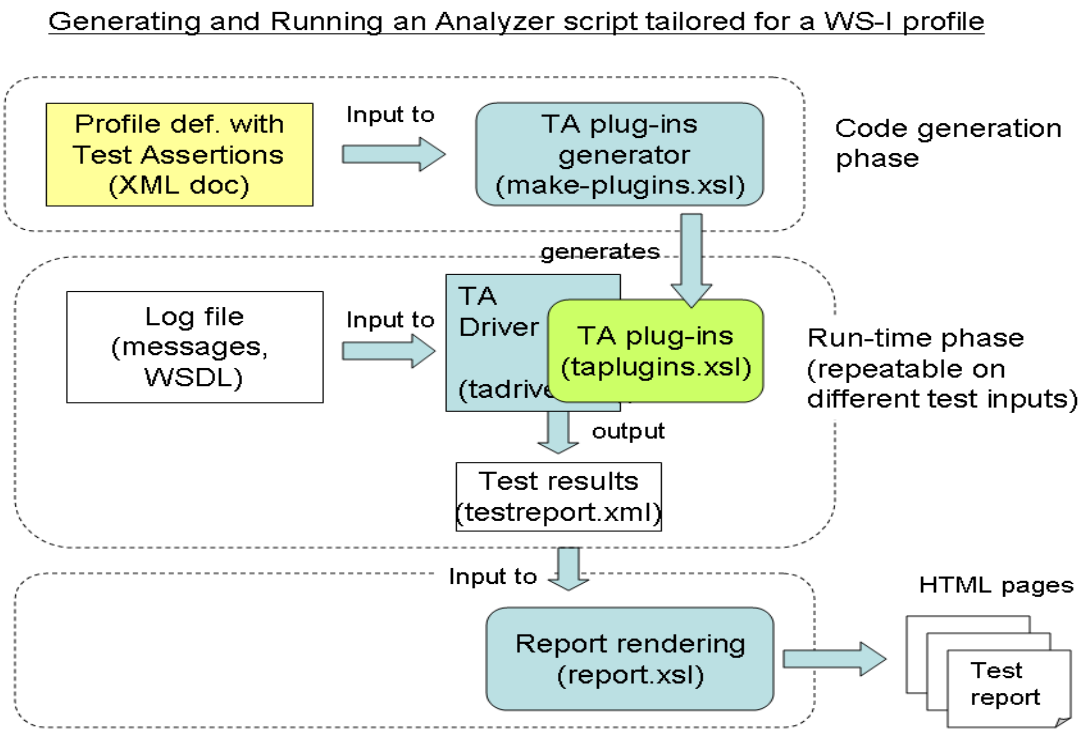
- **xslt.bat:** a batch file for executing xsl scripts using Saxon (windows envt). Please update so that (1) referred library files correspond to your implementation of Saxon (Java jars vs. .NET libraries), (2) the "home" directories for Saxon libs is yours.

Generated XSLT scripts (can be generated from above scripts):

- **taplugins.xsl:** Script generated by make-plugins.xsl. Must be called that way as the name is hard-coded in an import statement of the tadriver.xsl script. Profile-specific, does not change from one test run to the other.

- **testreport.xsl:** Test report (xml) generated by running tadriver.xsl over a test input file (here, testlog.xml).

Overall Test Process:



- The "blue boxes" in the above figure, represent code that is permanent and does not change from one profile to the other.
- The "yellow/green boxes" represent code or data that varies from one profile to the other.
- The "white boxes" represent data that changes from one test run to the other.

Usage:

A package supporting both XSLT2.0 and XPath 2.0 should be installed first.
E.g the Saxon B 9.0 package from http://saxon.sourceforge.net/ (or http://www.saxonica.com/)
The xslt command below is running the xslt.sh / xslt.bat script that is part of this package, which has been written for using the Saxon libraries.

1. Generate the TA plugins for a given profile definition:

>**xslt make-plugins.xsl BasicProfile-2.0.xml taplugins.xsl**

(in the above command line, only the profile doc name – here BasicProfile-2.0.xml – may change)

2. Run the Test driver over some test input data:

>**xslt tadriver.xsl testlog.xml testreport.xml**

(in the above command line, the input data file name may change, and the output test report name may change.)

3. Open the generated test report to browse its HTML representation.

Overview of Test Assertions:

Structure of a Test Assertion:

You will find TA samples in the profile document (e.g. BasicProfile-2.0.xml) in this package.
Main parts of a TA:

- **ID**: an assertion ID, of the kind: BPyyyy (not to be confused with the profile requirement ID of format Rxxxx, or extensibility points Ezzzz addressed by these test assertions)

- **Target**: this element is an XPath2.0 expression. Its role is to select the test targets inside the test log file.

- **co-target**: (optional) This element is an XPath2.0 expression. Its role is to select an accessory artifact related to the target, that is needed in order to evaluate the Predicate or the Prerequisite. For example, the target may be a SOAP message and the co-target a WSDL binding associated with this message.

- **Predicate**: this element is a Boolean expression (also written in XPath2.0 syntax).
  - If it evaluates to "true" for a selected test target, then the default TA outcome is "passed" in the test report.
  - If it evaluates to "false" for a selected test target, then the default TA outcome is "failed" in the test report.
  - The **reporting** element may change the way the TA outcome is reported in the test report (e.g. "warning" instead of "failed", etc.)

Other TA parts:

- **Error message**: this standard error notice will be reported in the test report when the TA fails.
- **Diagnostic**: indicates what additional content will be listed in the test report, with each failure.
- **Prerequisite**: (optional) see explanation below.

The Prerequisite element in a Test Assertion:

The execution of TAs must take into account prerequisites, when these are present. When a TA "mytestX" has another TA "mytestY" as prerequisite, then "mytestX" should be executed if and only if "mytestY" outcome was "passed" on the target.

The following rules apply:

- mytestY = "passed" → mytestX can be executed (producing an output of either pass or fail or notRelevant, in the test report)

- mytestY = "failed" → mytestX cannot be executed (producing an output of notRelevant, in the test report)

- mytestY = "notRelevant" → mytestX cannot be executed (producing an output of notRelevant, in the test report)

Several prerequisites may be present in a TA, in which case the semantics is that all of them must "passed" for the TA to be executed. For example:

- In BP20: BP2117 (BP2.0 req: R2726) (binding target) has prerequisites: BP2017 and BP2406.

NOTE: A condition for TA mytestY to be a prerequisite of TA mytestX, is that their respective target types must verify either rule:

- mytestY target type is same as mytestX target type (e.g. both are testing "WSDL bindings")

- mytestY target type is a "containing type" for mytestX target (see definitions below). E.g. mytestY is about "Message" while mytestX is about "[SOAP] Envelope".

Target types may have a "containing type" relationship:

- Target type Y is a containing type for target type X if & only if an instance of X is contained in an instance of Y.

The Test Input File:

The structure of a test input file is illustrated in **testlog.xml** and **log.xsd.**

The test input file combines description meta-data (WSDL , schemas…) and message capture. This file contains all "test targets" on which the Test Assertions must be run. It is produced by a Test Monitor not part of this package.
The TA XPath2.0 expressions must be written against the content of this file, treating it as a single document.

NOTES :
- each "message" is identified by a combination of attributes:
<wsi-log:message conversation="1" id="1" type="request" …>

@conversation : identifies an http connection (maybe common to several messages)

@id : a local ID that is automatically assigned by the Monitor to each message inside a connection.

@type : the role played by the message for the underlying protocol: request or response.

The Test Report:

General structure:

The test report provides essentially two views of a test run:

- the TA-centric view: it shows the list of TAs that were "active" for this test run, and for each one tells how many test targets passed, failed, etc. That gives an idea of whether the test run properly "exercised" all TAs.

- the Target-centric view: it shows a list of Test Targets, and for each target tells which TAs have been exercised, and if they passed, failed, etc. This view is the one useful to debug a WS instance or client. The TA ID provides a link back to the TA def (and related Rxxxx profile reqt.) inside the Profile doc.
NOTE: the test targets are classified in broad categories, or "artifact" types:

- message
- description
- discovery

Within each category, we find more fine-grained target types:

For message category: [SOAP]envelope, message, …
For description category: descriptionFile, binding, operation … (mostly, every significant sub-element in the WSDL file).

About the Identification of Test Targets:

The test report is distinguishing each test target, identified with a proper ID. This ID is obtained from the input test material. In other words, just looking in the testlog.xml file should allow you to identify the XML element that corresponds to a target ID that appears in the test report. Sometimes, this ID is composed when a single value is not sufficient to uniquely identify the target. For example, "Envelope" targets have an ID of the form: "19:1" where the first number is the @conversation ID, and the second is the @id number within this conversation.

Determining how to extract the ID of a target based on the type of this target, is automatically done by the test run-time analyzer.

For example:

- **Message target** : ID = concatenation : @wsi-log:conversation + ":" + @id (attributes of the wsi-log:message element)

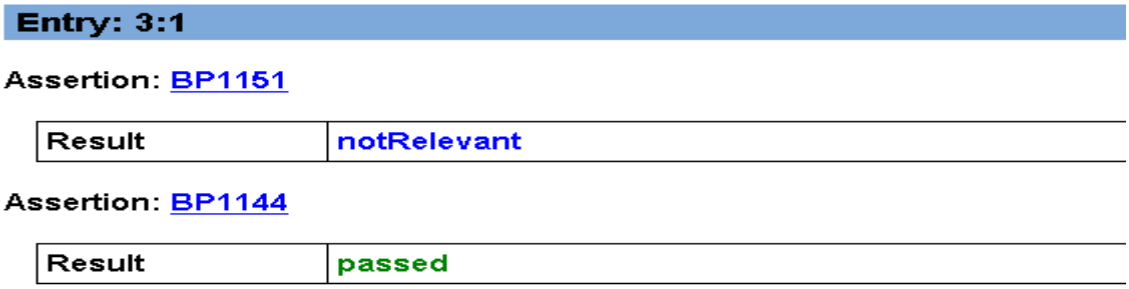
- **Envelope target** : ID = same as for Message target.

- **DescriptionFile target** : ID = wsi-log:descriptionFile/@filename

- **Binding target** : (WSDL binding) ID = concatenation : wsilog: descriptionFile/@filename + ":" + wsdl:binding/@name

- **Operation target** : (WSDL operation) ID = concatenation : wsilog: descriptionFile/@filename + ":" + wsdl:binding/@name + ":" + wsdl:operation/@name

The string that appears just after the "Entry:" label, in the test report, is the result of this ID expression computation. In the figure below, "3:1" is the ID of the Message target (@conversation = 3, @id = 1):



NOTE 1: the ID of a target with type X that is contained in a target type Y generally includes the ID of the related instance Y. For example:
Descriptionfile type is a containing type for Binding type.
The IDs for Bindings are of the form: <wsdl file name> ':' <binding name>
E.g.: ISupplier-rpc.wsdl : ISupplierSOAP

NOTE 2: In the test report, all targets are classified in two broad artifact categories:

1. Message artifacts (for Message and Envelope targets)
2. Description artifacts (for all other target types)

A more complete list of ID expressions for each target type, is given in appendix A.

Known Limitations & Bugs:

Issue #1. In order to automatically determine the ID of each test target, the script is testing the target type – e.g. envelope, WSDL binding… etc.
Not all types are supported yet. For now the supported target types are:
- For message artifacts: Message, Envelope.
- For description artifacts: DescriptionFile, Binding, Operation
The type of a target is simply the type of node selected by the TA "context" expression in the log input file.
Adding support for more target types is an easy task (a couple of simple additions in the tadriver.xsl file). More will be added in a next release.

Issue #2. Prerequisite restrictions: In the current package, a TA and its prerequisite TA(s) must use same target type (or related types, e.g. super-type / sub-type).
More diverse patterns between target types of Tas that are prerequisites of each other should be supported in a next release (this is not necessary so far for WS-I basic profile). For example, one might imagine a TA "a" testing some semantic requirement over an XML document, that has a prerequisite TA "b" that must be verified over a part of this document prior to verifying "a". In that case the "containing type" relationship is reversed: the target of the prerequisite TA is contained in the target of the main TA. It all depends on the nature of the test.
Or, the target types are disjoint, but related somehow so that the test engine can navigate from one to the other. This could be the case when testing "[SOAP]envelopes" for their conformance to the related "[WSDL]DescriptionFile". The target is still "Envelope" but the validity of the test depends on the integrity of the referred DescriptionFile, which should have been verified first. That could be done either by running the tests in two steps (one just for the Description types first, then the other for Messages), or again with an extended prerequisite mechanism.

Appendix A:

Complete list of ID expressions for each target type:

Message Artifacts:

Target type: **message** (/wsil:testLog/wsil:messageLog/wsil:message)
ID string = @conversation + ':' + @id

Target type: **envelope**
(/wsil:testLog/wsil:messageLog/wsil:message/wsil:messageContents/soap:Envelope)
ID string = message/@conversation + ':' + message/@id

Description Artifacts:

Target type: **descriptionfile**
(/wsil:testLog/wsil:descriptionFiles/wsil:descriptionFile)
ID string = 'desc:' + @filename

Target type: **wsdlfile**

(/wsil:testLog/wsil:descriptionFiles/wsil:descriptionFile/wsdl:definitions)
ID string = 'desc:' + descriptionFile/@filename + ':wsdl:' + definitions/@name

Target type: **binding**
(/wsil:testLog/wsil:descriptionFiles/wsil:descriptionFile/wsdl:definitions/wsdl:binding)
ID string = 'desc:' + descriptionFile/@filename + ':wsdl:' + definitions/@name + ':bnd:' + binding/@name

Target type: **bindingOperation**
(/wsil:testLog/wsil:descriptionFiles/wsil:descriptionFile/wsdl:definitions/wsdl:binding/wsdl:operation")
ID string = 'desc:' + descriptionFile/@filename + ':wsdl:' + definitions/@name + ':bnd:' + binding/@name + ':op:' + operation/@name

Target type: **bindingOperationFault**
(/wsil:testLog/wsil:descriptionFiles/wsil:descriptionFile/wsdl:definitions/wsdl:binding/wsdl:operation/wsdl:fault")
ID string = 'desc:' + descriptionFile/@filename + ':wsdl:' + definitions/@name + ':bnd:' + binding/@name + ':op:' + operation/@name + ':opFault:' + fault/@name

Target type: **wsdlimport**
(/wsil:testLog/wsil:descriptionFiles/wsil:descriptionFile/wsdl:definitions/wsdl:import)
ID string = 'desc:' + descriptionFile/@filename + ':wsdl:' + definitions/@name + ':import:' + import/@location

Target type: **portType**
(/wsil:testLog/wsil:descriptionFiles/wsil:descriptionFile/wsdl:definitions/wsdl:portType)
ID string = 'desc:' + descriptionFile/@filename + ':wsdl:' + definitions/@name + ':portType:' + portType/@name

Target type: **portTypeOperation**
(/wsil:testLog/wsil:descriptionFiles/wsil:descriptionFile/wsdl:definitions/wsdl:portType/wsdl:operation)
ID string = 'desc:' + descriptionFile/@filename + ':wsdl:' + definitions/@name + ':portType:' + portType/@name + ':' + operation/@name

Target type: **messagedesc**
(/wsil:testLog/wsil:descriptionFiles/wsil:descriptionFile/wsdl:definitions/wsdl:message)
ID string = 'desc:' + descriptionFile/@filename + ':wsdl:' + definitions/@name + ':msg:' + message/@name

Target type: **typedcl**
(/wsil:testLog/wsil:descriptionFiles/wsil:descriptionFile/wsdl:definitions/wsdl:types)
ID string = 'desc:' + descriptionFile/@filename + ':wsdl:' + definitions/@name + ':types:'