



SOAPSonar Enterprise User Guide

Version 10.0

Legal Marks

No portion of this document may be reproduced or copied in any form, or by any means – graphic, electronic, or mechanical, including photocopying, taping, recording, or information retrieval system – without expressed permission from Crosscheck Networks.

SOAPSonar™ and XSD-Mutation™ are trademarks and registered trademarks of Crosscheck Networks.

All other products are trademarks or registered trademarks of their respective companies.

Copyright © 2022 Crosscheck Networks, Inc. – All Rights Reserved.

Crosscheck Networks
Needham, MA

Crosscheck Networks SOAPSonar™ Enterprise Version 8.0 User Guide, published December 2022.

ASD-D-SS-119673

Table of Contents

INTRODUCTION.....	10
SOAPSonar Editions	11
LICENSING	12
INSTANCE LICENSES	12
Internet License Activation	13
Manual License Activation.....	13
FLOATING LICENSES	15
Request New License Lease.....	16
Request a Specific License Key.....	16
Return License Back to Server.....	16
Show Licenses in Use	17
GETTING STARTED	18
Understanding the SOAPSonar GUI	19
SOAPSonar Menu Items	19
Project View	22
Project View – Project Quickstart options	22
Project View – Project Tree	22
Project View – Test Case Request	23
Project View – Test Case Response	23
Project View – Success Criteria	24
Run View.....	24
Run View – Project Tree	24
Run View – Suite Tree	24
Run View – Suite Settings	25
Run View – Test Group Settings.....	25
Run View – Run Stats	26
Report View	26
Report View – Log View Tree	26
Report View – Test Result Summary.....	26
Report View – Executed Test Request	27
Report View – Executed Test Response	27
Report View – Generate Report Options	27
Workflow Views.....	27
Run Modes.....	27
Types of Test Cases	28
OpenAPI Test Case	29
SOAP Test Case.....	29
XML Test Case	29
REST Test Case	29
JSON Test Case	29
Batch File Test Case.....	29
WS-TRUST Test Case.....	29
OpenAPI Capture and Processing	29
WSDL Capture and Processing.....	30
Working with Projects	31
Settings and Preferences	31
WSDL Parsing and SOAP Generator Settings	31
SOAP Settings	31
WSDL Capture and Parsing Optimizations	33
Auto generate Test Case for Each WSDL Operation	33
Sort WSDL Operations Alphabetically	33
Optimize Schema Optional Element Parsing Depth	33
Project Save Settings	34

SSL Security Settings	34
Test Suite Settings	35
Reporting Settings	36
Email Settings.....	36
Global Proxy Settings	36
Global Variables	37
Script Includes Variables	37
HP Quality Center.....	37
BUILDING TESTS.....	38
SOAP Test Case	39
Schema Fields Generator.....	40
Features of Schema Fields Generator	41
Data Entry: Dynamic Array Structures	41
Data Entry: Optional Elements.....	41
Data Entry: Optional Attributes	41
Data Entry: Abstract Types (XSD Polymorphism)	42
Data Entry: Auto fill	42
Data Entry: Variable Parameters	43
Data Entry: Context Functions	43
Data Entry: Entry Recall.....	44
Schema Fields View Filtering	44
OpenAPI Test Case	46
Schema Fields Generator.....	47
XML Test Case	47
REST Test Case	48
Generate HTML Form Post Request Builder	49
JSON Test Case	52
Batch File Test Case	52
WS-Trust Test Case	54
TEST CASE REQUEST	55
Test Case Protocols	55
HTTP Decompression Support.....	55
Test Case Request Definition.....	56
Request URI	56
STUB Tests	56
HTTP Headers.....	56
Authentication Credentials.....	57
HTTP Basic Authentication	58
HTTP Digest Authentication.....	58
HTTP Kerberos Authentication	58
Cookie Authentication	58
Kerberos / NTLM Authentication.....	58
Amazon AWSv4 Authentication	58
SSL X.509 Authentication	58
OAuth2 Authentication	58
Global Authentication Credentials	58
Global Cookies and HTTP Custom Headers.....	59
Automatic Cookie Handling	59
MTOM, MIME, and DIME Attachments	59
MTOM and MIME Large File Streaming.....	61
Test Case Request Tasks	62
Global Request Tasks.....	62
Request Task: Identity Tokens	63
Request Task: Identity Tokens->Add SAML 2.0Token.....	64
Request Task: Identity Tokens->Add WSS Username Token.....	64
Request Task: Identity Tokens->Add WSS X509 Token.....	64

Request Task: Identity Tokens->Add WSS Kerberos Token.....	64
Request Task: Identity Tokens->Add WSS SAML 1.1 Token	64
Request Task: Identity Tokens->Add WSS SAML 2.0 Token	64
Request Task: Identity Tokens->Add WSS Token from Runtime Variable	64
Request Task: WS-Security and WS-Addressing.....	64
Request Task: JWT Security	65
Request Task: JWT Security->JWT Sign/Encrypt	65
Request Task: JWT Security->JWT Decrypt/Verify	65
Request Task: Data Conversion	65
Request Task: Data Conversion-> Add Custom XML	66
Request Task: Data Conversion-> Omit XML Nodes	66
Request Task: Data Conversion-> Create Variable Array.....	66
Request Task: Data Conversion-> Extract XML Fragment	66
Request Task: Data Conversion-> Convert XML to Google Protocol Buffer.....	67
Request Task: Data Conversion-> Convert Google Protocol Buffer to XML.....	67
Request Task: Data Conversion-> Convert JSON to XML.....	67
Request Task: Data Conversion-> Convert XML to JSON.....	67
Request Task: Data Conversion-> Encode Values	67
Request Task: Data Conversion-> String Replacement.....	67
Request Task: Data Conversion->XSL Transform	67
Request Task: Data Conversion->XML Conditional Template.....	67
Request Task: Custom Functions.....	67
Request Task: Custom Functions->VBScript and JScript	68
Request Task: Actions->Database Query	70
Request Task: Actions->File Manipulation	71
Request Task: Actions->Add Test Delay	71
Request Task: Actions->Update Global Variable	71
Request Task: Actions->Update Memory Table	71
Request Task: Actions->Send Email	72
Request Task: Actions->Purge Message Queue.....	72
Test Case Authoring – Change Request and Response GUI Layout.....	72
TEST CASE RESPONSE.....	73
Response Data View Formats.....	73
Response Tasks.....	74
Response Large File Streaming.....	74
No Streaming Optimization (default).....	75
Streaming Statistics	75
Streaming Limit Element Content	75
ORGANIZING TESTS.....	75
Project View Test Folders.....	76
Test Suite Group Filtered View.....	76
TEST VARIABLES	77
Context Function Variables.....	78
Context Function Variable Format.....	78
Context Function: Now()	78
Context Function: GUID().....	78
Context Function: FileContents()	79
Context Function: Random()	79
Context Function: B64()	79
Context Function: MD5()	79
Context Function: SHA1()	80
Context Function: PEM().....	80
Context Function: SPACE()	80
Context Function X509Attribute().....	80
Context Function: Env()	80
Context Function: URLEncode()	80

Global Variables.....	81
Global Variable Format.....	81
Project Global Variables	82
Project Global Variable Format	82
Internal Variables	82
Internal Variable Format	82
Memory Table Variables.....	83
Memory Table Variable Format.....	83
Memory Table Variables in VBScript and JScript Code.....	84
Memory Table Variables Template Resolution	85
Memory Table Variables vs Global Variables	85
Automatic Data Function Variables	86
Automatic Data Function Library	86
Automation Data Function Project Instance Library	86
Automatic Data Function Variable Format	88
Automation Data Source Variables	89
ADS Policy Options	90
Encode XML Tags.....	90
Limit Rows.....	90
Null Value.....	90
Relative Path Root.....	91
Selecting and Associating ADS Variables.....	91
Excel vs CVS Formats	93
Automation Data Source Variable Format.....	93
Runtime Variables (Request and Response Capture Variables)	94
Update Memory Table with Runtime Variable.....	95
Capture Runtime values as Array Variable	95
Mirror Value in Global Variable	96
Include Encoded XML setting.....	96
Runtime Variable Format.....	96
Capturing Array Targets with Runtime Variables	97
Request URI Variables	99
Automatic Test Case Chaining.....	100
Steps to Create a new Test Case Chain	100
Disable Automatic Test Dependency Detection.....	102
Iteration Mode Data Sources	103
Test Flow Mapping Data Sources.....	104
HP QC Linked Data Sources	105
TEST TOOLS.....	106
Edit Using Popup Window	107
Run WSDL Schema Validator	107
Resolved Variable Test Iteration Viewer	108
NATIVE PKI MANAGEMENT	110
Windows Keystore.....	110
Java Keystores	110
PFX Files	111
SmartCard Keys	111
Key Alias Management.....	111
UDDI Explorer.....	111
Populate Schema Data Types	113
Map Fields to Automation Data Source.....	113
LINKED TEST CASES.....	115
Importing a Linked Test Case into the Project.....	115
Detach a Test Case as a File Link.....	116
Remove a Test Case File Link.....	117
Change a Test Case File Link	117

Read-Only Test Case File Links	117
TEST SUITE MANAGEMENT	118
Create a Test Suite	119
Add Test Cases to a Test Suite	119
Test Suite Pre-Run Tasks	120
Test Suite Post-Run Tasks	121
Conditional Testing	121
Run View Toolbar Options	122
Run Suite	122
Create and Schedule Command-Line Script	122
Create Baseline Regression Data Set	122
Test Suite Properties	122
Test Suite Properties: QA Mode	123
Test Suite Properties: Performance Mode	124
Test Suite Properties: Compliance Mode	126
Test Suite Properties: Vulnerability Mode	127
PROJECT MANAGEMENT	128
Import a Project	129
Load as new Project	129
Merge Project	129
Append Project	129
Save a Project	129
Save Full Project	129
Export Project	130
WSDL Project SOAP Version	130
Upgrade a WSDL	130
Global Find/Replace Options	130
Convert SSP Project to Raw XML	131
RUN MODES	132
QA MODE	133
QA Test Cases	134
Success Criteria Rules	134
Success Criteria Rules: XPath Match	135
XPath Success Criteria Rule Evaluation Modes	137
Using XPath Match Rule To Validate File and JMS, MQ, WLS, and EMS Message Queues	138
XPath Runtime XML Source: Response	138
XPath Runtime XML/JSON Source: FILE	138
XPath Runtime XML/JSON Source: MESSAGE QUEUE (JMS, MQ, WLS, EMS)	139
Success Criteria Rules: XPath Array Handling Options	140
XPath Array Handling: DEFAULT	140
XPath Array Handling: IGNORE ORDER	140
XPath Array Handling: MATCH ALL	141
Success Criteria Rules: XPath Compare Node Fragment Array to Database Query	142
XPath Rule Evaluation Mode - Conditional Analysis Scripting Rules	142
Using XPath Database Matching Rules	144
Success Criteria Rules: HTTP Code Range	145
Success Criteria Rules: HTTP Header String Match	145
Success Criteria Rules: String Match	146
Success Criteria Rules: SOAP Fault Match	146
Success Criteria Rules: Attachment MD5	146
Success Criteria Rules: Response Time	147
Success Criteria Rules: VBScript Module	147
Success Criteria Rules: XSD Schema Validation	148
Success Criteria Rules: Schematron Validation	148
Success Criteria Rules: Invoke DLL Plugin	149
Success Criteria Rules: Database Query	151

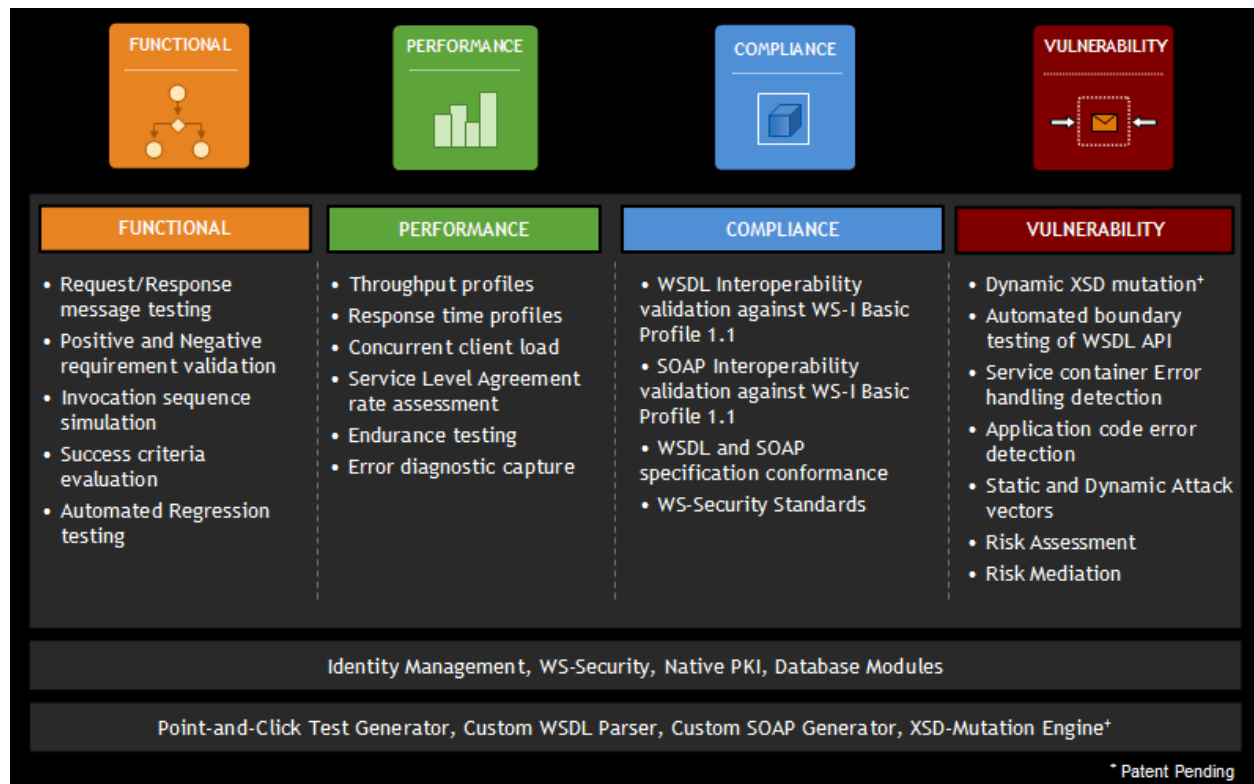
Success Criteria Rules: File Analysis	151
Success Criteria Dynamic Variables	152
Baseline Regression Testing	154
Vulnerability Scan of QA Test Results	159
PERFORMANCE MODE	161
Performance Test Cases	162
Performance Test Suite Property Settings	162
Override URI	163
Detailed Transaction Analysis (Error Tracing)	163
Tracing Analysis Options:	163
Storage Options:	163
Disable Dependency Checking (Run Primary Tests Only)	164
Segment Data Source Rows Uniquely Among Virtual Clients	164
Run all Test Groups Asynchronously at the Same Time	164
Performance Test Group Property Settings	165
Virtual Clients	165
Test Duration	165
Iteration Mode	166
Duration Mode	166
Data Source Iteration Mode	167
SLA Rate Throttling	167
Ramping Virtual Clients	167
Ramp Up	168
Ramp Down	168
Ramp Strategy	168
Think Time	168
Concurrent Load	168
Multiple Source Client IPs	168
Performance Mode Test Variables	169
How Statistics are Calculated	169
Distributed Loading Using SOAPSonar Agents	170
VULNERABILITY MODE	172
Static Attack Vectors	173
Dynamic Attack Vectors	174
Automatic Vulnerability Discovery Libraries	176
AVD Criteria	177
Scanning Responses using AVD Definitions	177
Risk Score	177
COMPLIANCE MODE	178
Design Time WS-I Basic Profile 1.1 WSDL Analysis	179
Active WS-I Basic Profile 1.1 SOAP Analysis	180
LOGGING AND REPORTING	182
Generating Reports	183
Report Export Formats	183
Exporting Raw Request and Response Data to File	184
Exporting Raw Log Results to Normalized XML	184
Exporting Log Data to CSV, Excel, or HTML Format	184
HTTP Response Error Codes	184
IBM MQ, Weblogic JMS, Tibco EMS, and Native JMS MESSAGE PROVIDERS	185
MQ Message Provider	185
Oracle Weblogic JMS Message Provider (WLS)	188
Tibco EMS Message Provider	190
JMS Message Provider	192
COMMAND-LINE INTERFACE	194
Running Test Suites From The Command-Line	194
Generating Reports from the Command-Line	195

SMARTCARD INTEGRATION.....	197
Setup for SmartCard Integration Support.....	197
Using a SmartCard Key for Signing, Encryption, or Decryption.....	197
Using a SmartCard Key for SSL X509 Mutual Authentication.....	197
HP QUALITY CENTER INTEGRATION : EMBEDDED.....	199
SOAPSonar Embedded Integration Features	199
HP Quality Center Console	199
Login Screen.....	199
Project Viewer	200
Uploading and Downloading Files Directly To HP Quality Center Server	200
Automatically Publish Test Results to HP Quality Center	201
Publish Option #1: Link Results to Test instance	201
Publish Option #2: Link Groups to Test Set.....	202
Manually Publish Test Results to HP Quality Center	202
Publish Option #1: Link Results to Test instance	203
Publish Option #2: Link Groups to Test Set.....	204
Dynamically Link to Quality Center Attachments for SOAPSonar Data Sources	205
HP QUALITY CENTER INTEGRATION : VAPI-XP-TEST	207
SOAPSonar Generated HP Quality Center Test Script.....	207
Integrate the SOAPSonar Test Script into HP Quality Center.....	208
JUNIT INTEGRATION	214
SOAPSonar Junit Class Wrapper.....	214
SERVICE SIMULATION USING CLOUDPORT	215
Launching CloudPort Runtime Simulation Player.....	215
WSDL SCORING AND GOVERNANCE.....	216
WSDL Scoring	217
WSDL Scoring Rules	218
Rule Categories	218
WS-I Basic Profile Analysis	219
WSDL and Schema Document Review	220
EXTENSIBLE PLUGIN API.....	221
Request Event	221
Response Event	221
Evaluate Success Event	222
Enterprise License Component – Automation Edition	224
SOAPSonar Enterprise Automation Module License Component (APC)	224
Enterprise License Component Upgrade – Performance Agent Pack	225
SOAPSonar Enterprise Performance Agent Pack (PAP).....	225
Appendix A – Now() Context Function Date Formats.....	226
Appendix B – SOAPSonar Product Edition Feature Matrix	228

INTRODUCTION

Welcome to the Crosscheck Networks SOAPSonar API testing product. Crosscheck Networks created the SOAPSonar product to address the full range of API and app services testing across the entire deployment lifecycle. SOAPSonar API Testing provides integrated extensible features that allow you to accomplish testing for functional, performance, interoperability compliance, and security testing across diverse API message formats (such as JSON, XML, SOAP, etc) and across different API protocols .

Building robust and reliable APIs requires the ability to validate functional requirements of the API and application behavior, measure and validate performance characteristics, maximize interoperability, and understand the vulnerability gaps in the API error handling. Each of these areas is addressed through the SOAPSonar testing modes. Each testing mode is meant to address each of the 4 pillars of API testing, Functional, Performance, Compliance, and Vulnerability. These pillars are broken down into the logical diagnostics areas as shown the illustration below.



SOAPSonar also has comprehensive support for identity and security standards to provide the means to validate other key components of your API architecture such as IDAM and security controls.

SOAPSonar Editions

The features within the product depend on the license and edition of SOAPSonar you are running. SOAPSonar Personal Edition and SOAPSonar Enterprise Edition represent the installation platforms for the product line . Within the Enterprise Edition platform, component licenses can be applied to enable different levels of functionality. A summary of the product editions are provided below. A matrix of features per product can be seen in [Appendix B](#). If you attempt to access an unlicensed feature, SOAPSonar will present a dialog indicating that the install type or the existing license does not provide access to the feature.

SOAPSonar Personal Edition

Personal Edition contains a subset of functionality from the Enterprise Edition and is meant for lightweight testing and diagnostics. This edition is provided free to the community.

SOAPSonar Enterprise Standard Edition

Standard edition provides the core features of the product such as OpenAPI 2.0 and 3.0 parsing, complex WSDL loading, all 4 pillars of API testing, test suite management, and reporting. This edition can be upgraded to Professional or Platinum edition via a new license key.

SOAPSonar Enterprise Professional Edition

Automation edition extends Standard Edition features providing access to data source variable substitution, command-line interface, test scheduling, and baseline regression testing. This edition can be upgraded to Platinum Edition via a new license key.

SOAPSonar Enterprise Platinum Edition

Platinum edition extends Automation Edition features with 50 additional performance testing virtual clients and rate-based throttling. This edition can add additional virtual users for more performance testing horsepower by purchasing performance agent packs which provide sets of additional 50 virtual users. Platinum Edition also adds support for Large File Streaming, and ESB protocols such as IBM MQ, Tibco EMS, WebLogic JMS, and native JMS.

For more information about the edition comparisons and features within each, please visit our web site at: <http://www.crosschecknet.com/products/soapsonar/>

LICENSING

SOAPSonar can be licensed per installed instance, or by using a License Server which allows for floating licenses to be used on-demand.

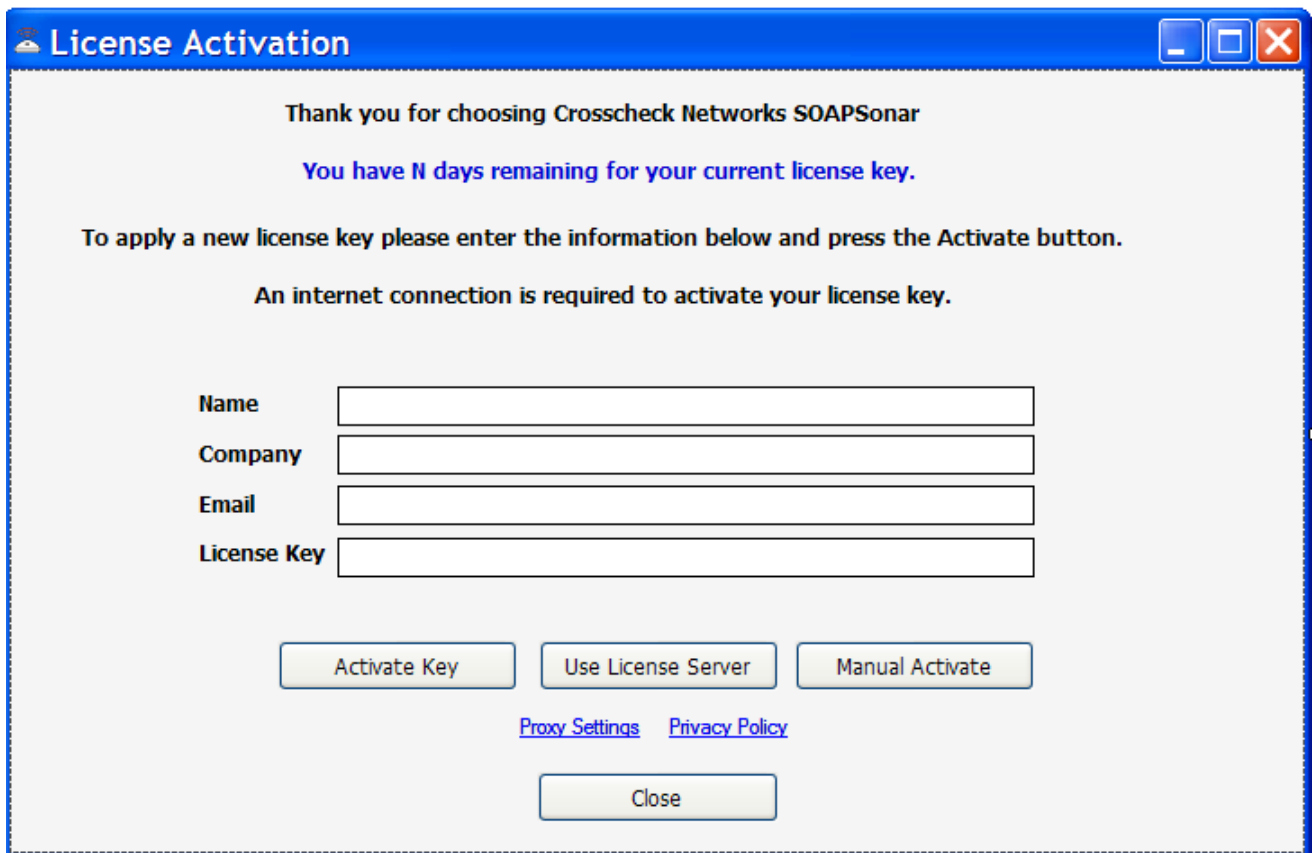
Licenses that are installed directly on the machine where SOAPSonar is running are called **Instance Licenses**. Instance licenses are locked to the machine in which SOAPSonar is installed via a machine identification mechanism built into the licensing.

Licenses that are shared among SOAPSonar installations are called **Floating Licenses**. Floating licenses are leased for a designated period of time as needed, and returned to the pool of available licenses once the lease has expired or when the product is closed and no longer in use.

INSTANCE LICENSES

Instance based licensing uses license keys which are activated on the specific instance of the application. An instance license is enabled based on activation of the license key.

To apply an instance license, go to the **Registration->Install Instance License** menu and enter your user information as well as the license key provided to you.



License Activation

Thank you for choosing Crosscheck Networks SOAPSonar

You have N days remaining for your current license key.

To apply a new license key please enter the information below and press the Activate button.

An internet connection is required to activate your license key.

Name

Company

Email

License Key

Activate Key Use License Server Manual Activate

[Proxy Settings](#) [Privacy Policy](#)

Close

Internet License Activation

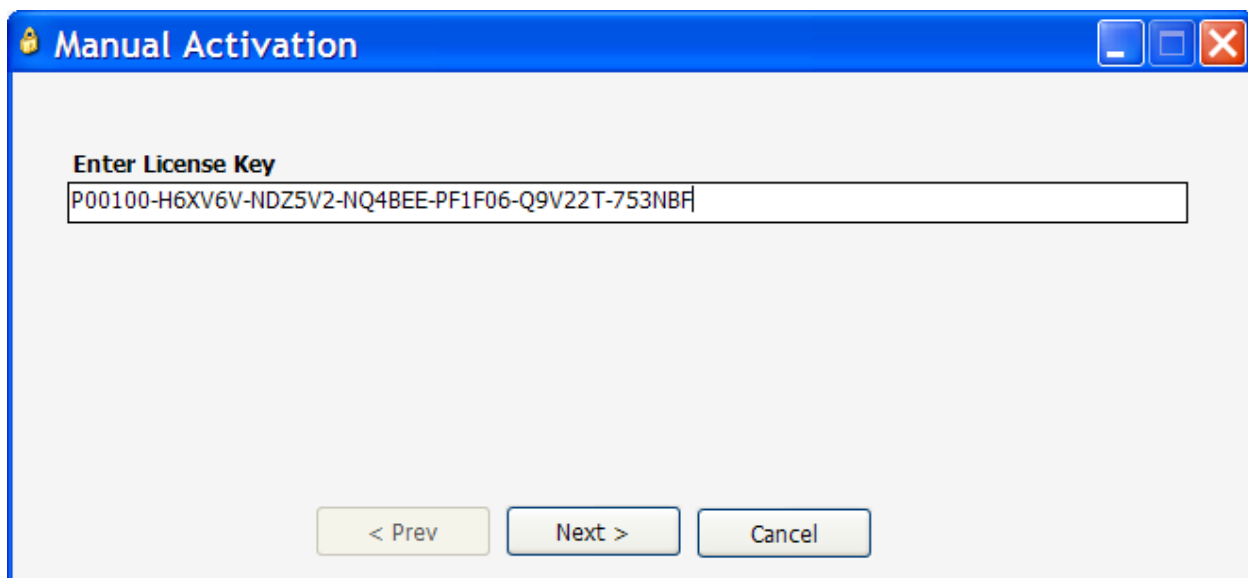
Once the licensing information has been entered, press the Activate button to activate this license against the Crosscheck Networks Activation server. You will require access to the internet for this activation to succeed. If the license key has valid activations remaining, your license will be enabled for this instance.

If you do not have internet access, refer to the Manual Activation steps shown in the Manual License Activation section below.

Manual License Activation

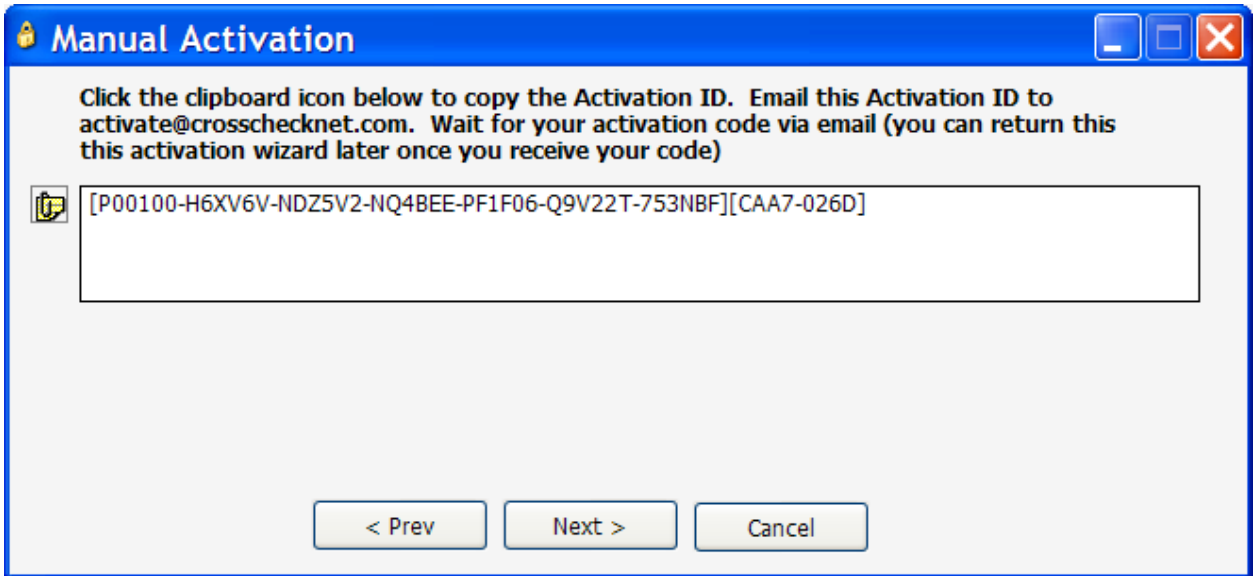
If you do not have internet access to activate your license using the Crosscheck Networks Activation server, then follow the steps below the manually activate your license instance.

1. Enter your name, company name, email, and key code in the fields provided.
2. Click on the "Manual Activate" button below the key code field. The manual activation dialog will appear with your target license key to be activated. Please check this key to be sure it is the key you intend to activate. Press the Next button

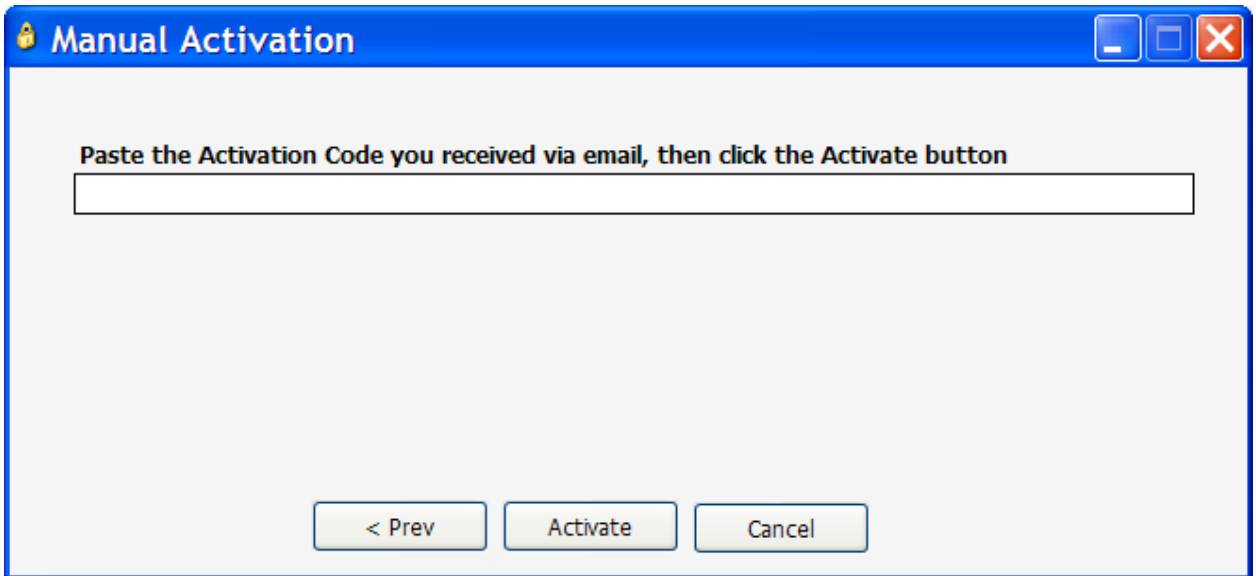


The screenshot shows a dialog box titled "Manual Activation" with a blue header bar. Inside the dialog, there is a label "Enter License Key" above a text input field. The input field contains the license key: "P00100-H6XV6V-NDZ5V2-NQ4BEE-PF1F06-Q9V22T-753NBF". At the bottom of the dialog, there are three buttons: "< Prev", "Next >", and "Cancel".

3. The Manual activation code will be generated on the next screen. Copy the activation ID shown (click the clipboard button) and email this ID to activate@crosschecknet.com



4. Crosscheck Networks will return the Activation Code in an email. Enter the provided Activation Code in the "Enter Activation Code" box



5. Close and restart the product.

FLOATING LICENSES

Floating Licenses use a license server to obtain a lease for a license key for a specified duration. The Crosscheck Networks License Server is installed on some location within the network that can be accessed by the individual SOAPSonar instances. The Crosscheck Networks License Server uses a leasing model which does not require persistent network access to the licensing server, but rather only access at the time the lease is granted. Once a lease is granted, the product instance will not need to have a connection to the license server to use the leased license.

To request a new license lease, go to the **Registration->Use License Server** menu.

License Server Options

Thank you for choosing Crosscheck Networks SOAPSonar

Status: No Active License Server Lease

License Server Location

Server: 10.5.1.50 Port: 9550 [Proxy Settings](#)

License Actions

- Request New License Lease
- Return License Back to Server
- Show Licenses In Use

New License Lease

Key Type: Standard Edition Lease Duration (minutes):

Request License

Automatically attempt to return license to server when SOAPSonar closes

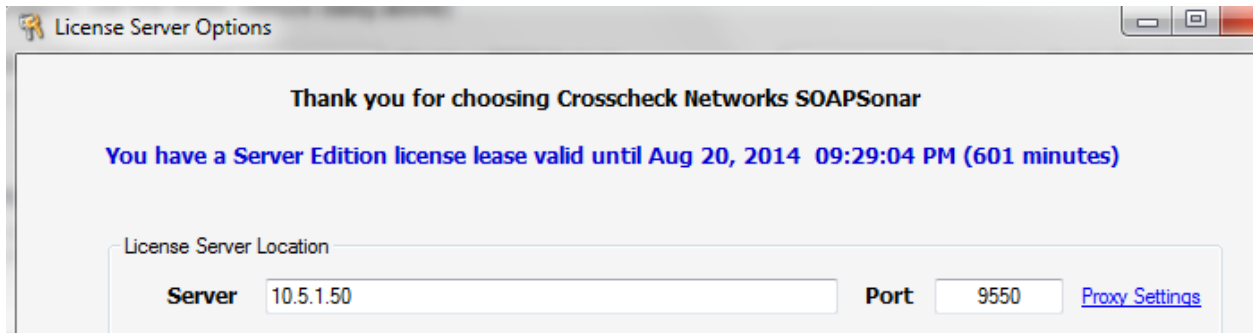
Close Change To Instance License

Request New License Lease

The floating license model works on a leasing system that only requires access to the license server at the time of the lease request. Licenses can thus be obtained and used for the duration of the lease interval without requiring access to the license server itself.

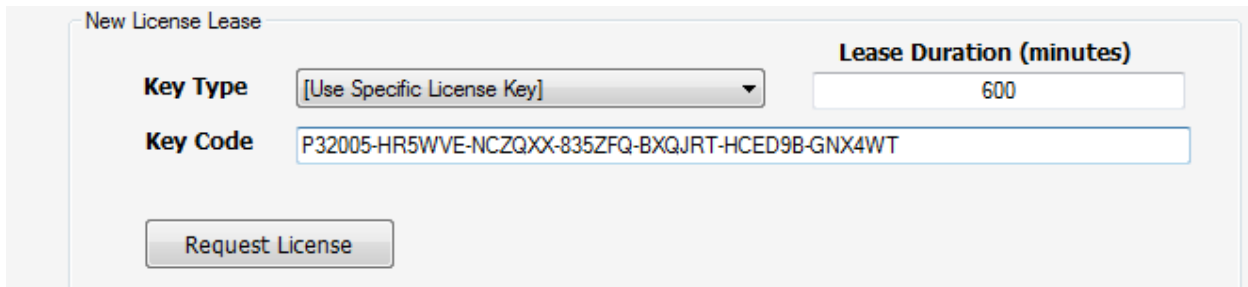
To request a new license lease, enter the information about the server location either as the domain name, or the IP address, and the port the server is configured to listen on (default is 9550). Then select the action “Request New License Lease” and choose the type of license and the duration of the lease and click on the Request License button to obtain a license lease from the server.

If successful, you will see the summary at the top of the screen that shows the duration of the newly obtained license lease and the type of license which is now active for use.



Request a Specific License Key

To check out a license key from the server with specific functionality, or when you have multiple license server keys of the same type and want to use a lease of a particular key, use the “use specific license key” option from the key type and enter the license key that you want to use from the license server.



Return License Back to Server

You can choose to return a leased license back to the license server at any time by going to the “Return License Back to Server” option. If you do not have network access to the server you will be prompted with a release code that can be sent to the License Administrator for manually removing the lease from the server.

Show Licenses in Use

To see the current leases in effect for the selected license type and the duration of each, go to the “Show Licenses in Use” option and query the license server to see all active leases and the times remaining for each.

GETTING STARTED

This section will help explain the various aspects of the SOAPSonar work flow. Subtopics within this section include

- [Understanding the SOAPSonar GUI](#)
- [Workflow Views](#)
- [Run Modes](#)
- [Types of Test Cases](#)
- [Capture a WSDL](#)
- [Auto-populate Settings for Test Case Creation](#)
- [Working with Projects](#)

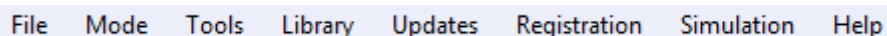
Understanding the SOAPSonar GUI

The SOAPSonar interface is broken down into 3 views: Project View, Run View, and Report View. Menu options are available for accessing various tools and configuration options of the product. The SOAPSonar interface includes:

- [Menu Items](#)
- [Project View](#)
- [Run View](#)
- [Report View](#)

SOAPSonar Menu Items

The SOAPSonar menu includes items allowing you to perform configuration setup, perform actions, and license the application.



File Mode Tools Library Updates Registration Simulation Help

File Menu: New WSDL Test Project

This option will open the WSDL capture panel and allow you to choose a network or file location where the WSDL resides to capture and parse into a SOAPSonar project for SOAP testing.

File Menu: New OpenAPI Test Project

This option will open the OpenAPI capture panel and allow you to choose a network or file location where the OpenAPI 2.0 or 3.0 document resides to capture and parse into a SOAPSonar project for JSON/XML testing.

File Menu: New Custom Test Project

Creates a new test group which can contain any type of customized tests, including REST, JSON, and XML testing.

File Menu: Import Proxy Server Traffic Capture

Creates a project from traffic captured using the Crosscheck Networks Traffic Capture tool (available from the **Tools->Proxy Server Traffic Capture Tool** menu)

File Menu: Load Project

Loads a previously stored SOAPSonar project.

File Menu: Save Project (As)

Stores current Project and Run settings for into a SOAPSonar project file.

File Menu: Clear Project

Removes all items in the Project and Test Suites areas.

File Menu: Recent Projects

Load project by selecting from the provided list of recently loaded or saved projects.

File Menu: Recent WSDLs

Load WSDL by selecting from the provided list of recently captured WSDLs.

File Menu: Recent OpenAPIs

Load OpenAPI definitions by selecting from the provided list of recently captured OpenAPI 2.0 or 3.0 documents.

File Menu: Settings and Preferences

This option brings up the screen that allows the configuration of various defaults and settings for the running instance of SOAPSonar on that machine.

Mode Menu: QA

The run mode for functional testing, baseline regression testing, and success evaluation.

Mode Menu: Performance

The run mode for performance testing, concurrent clients, and service level agreement rate throttling.

Mode Menu: Compliance

The run mode for interoperability assessment for WS-I Basic Profile including both passive WSDL and active SOAP interoperability compliance adherence. This setting is only applicable to WSDL projects

Mode Menu: Vulnerability

The run mode for automated boundary condition error testing with patent-pending dynamic XSD mutation test generation and risk mitigation and risk assessment definitions.

Tools Menu: PKI Management

Launch native PKI editor providing direct access to public-private keys.

Tools Menu: Key Alias Management

Allows the definition of aliases that map dynamically to the PKI keys when the test cases are run. This option allows testers to dynamically select/map different keys at runtime.

Tools Menu: HP Quality Center Console

Launches the interface to define integration with HP Quality Center.

Tools Menu: UDDI Browser

Launches UDDI browser to search for services in a UDDI registry.

Tools Menu: XML Viewer

Launches the XML viewer for inspecting XML files in formatted graphical format

Tools Menu: Active Cookie Viewer

Launches the dialog that shows any active cookies currently in the cache from running tests. These cookies are the ones used by the “Use Cookies” setting on the authentication tab of test cases. When “Use Cookies” is enabled, SOAPSonar will automatically store cookies based on API responses that have the Set-Cookie headers.

Tools Menu: SFTP Client

Launches the embedded sftp client to transfer files using sftp protocol

Tools Menu: Database SQL Tool

Launches the SQL tool allowing queries to be executed on ODBC database targets

Library Menu: Automatic Data Functions

Opens the ADF configuration panel to create, modify, and delete Automatic Data Functions for test input parameterization.

Library Menu: Automatic Vulnerability Detection

Opens the AVD configuration panel to create risk assessment and risk mitigation filter definitions to be used in Vulnerability test mode when running a vulnerability assessment test.

Updates Menu: Check for Product Updates

Checks for updates to your existing installed SOAPSonar version

Registration Menu: Install Instance License

Used for licensing by the instance method where the installation of SOAPSonar is directly licensed on the machine where it is installed.

Registration Menu: Use License Server

Used to open the Floating License configuration panel to checkout a license from the license server, check-in a license to the license server, or query the license server for the current floating licenses in use by SOAPSonar instances on the network.

Registration Menu: Deactivation

For instance based licensing, this option deactivates a license and provides a deactivation ID which can be provided to activate@crosschecknet.com for moving the license to another machine.

Simulation Menu: Launch CloudPort Service Simulation

Starts the CloudPort service simulation product enabling creation of simulated back-end services that SOAPSonar can send transactions to.

Help Menu: Contents

Opens the online help.

Help Menu: Feedback

Send email to Crosscheck Networks regarding product feature request, bug reports, or general comments.

Help Menu: About

Shows license details and product version.

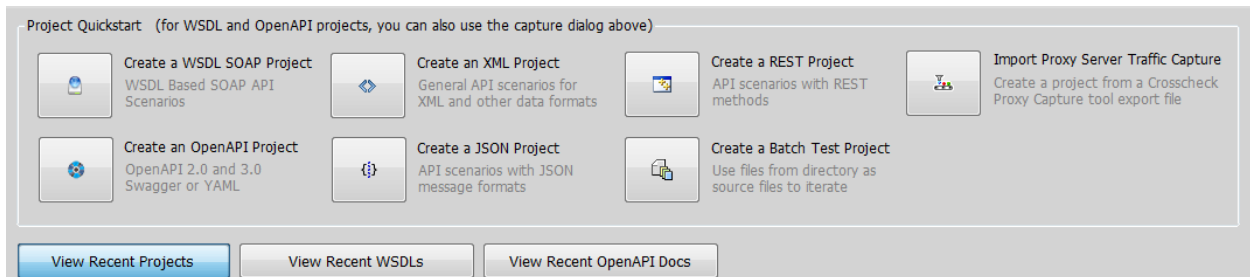
Project View

Opens the panel views used to create new tests, define inputs, analyze documents, build success criteria, and perform dynamic task configuration for run-time message policy and enrichment.



Project View – Project Quickstart options

To begin a new project, or to load a previous project such as an OpenAPI, WSDL, XML, JSON, or REST project you can use the Project Quickstart buttons. These are shortcuts that work the same as the File->New menu options.

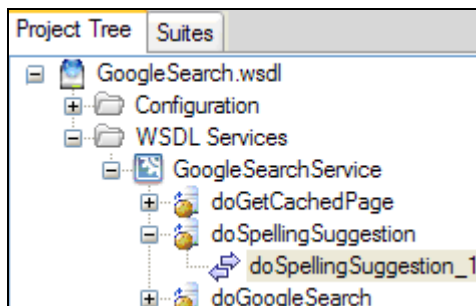


Additionally, recent projects are visible under the “View Recent Project” button, as well as recently loaded OpenAPI and WSDL documents. Previously loaded projects, OpenAPI documents, and WSDLs can be annotated with comments by clicking on the document icon and entering in comments relating to the Project, OpenAPI document, or WSDL.

To remove entries, click the checkbox for those entries to remove, then click the “Delete Selected Items” button that will appear.

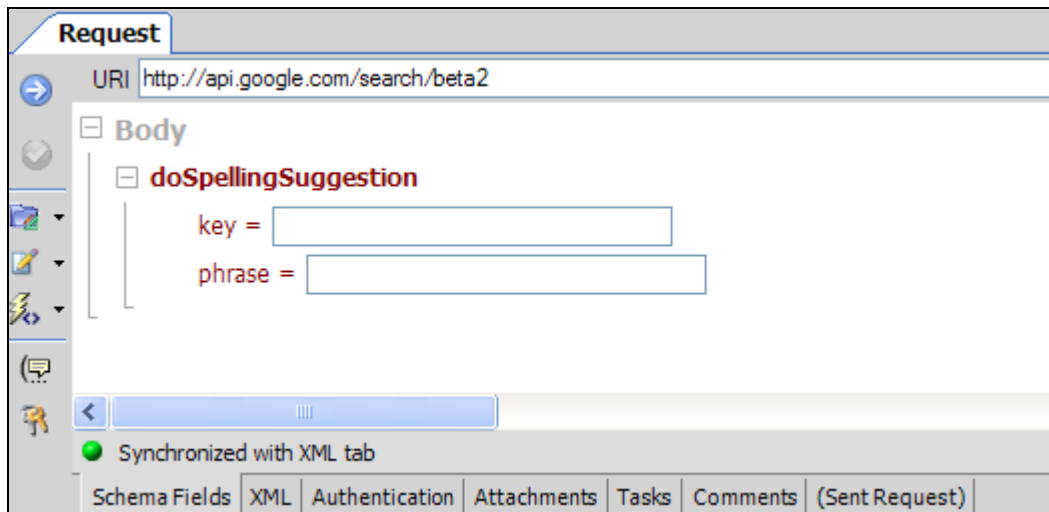
Project View – Project Tree

The Project tree displays the loaded project(s) and enables navigation to the configuration aspects of setting up tests.



Project View – Test Case Request

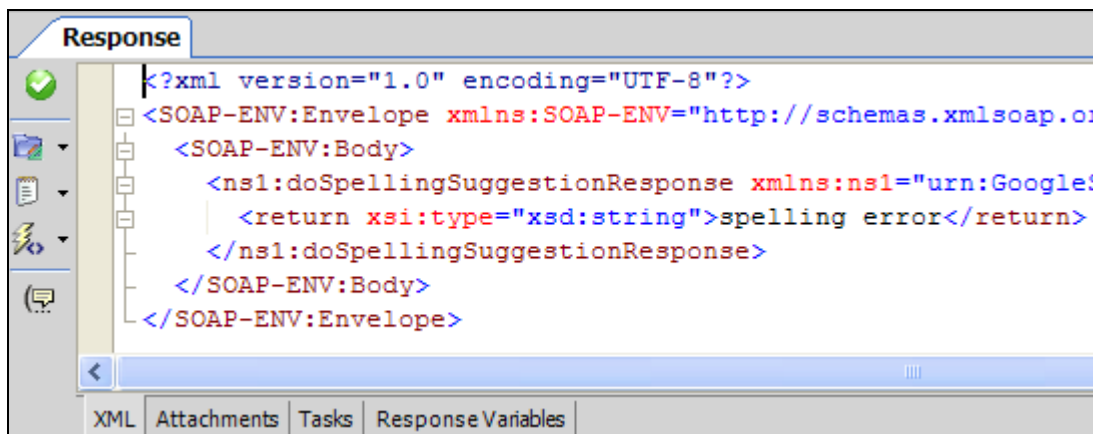
Displays request panel for defining test input for API testing including authentication settings, security settings, workflow processing tasks, and custom plugin extensions.



The Schema Fields tab is displayed for OpenAPI and WSDL projects where the XML/JSON tab can be autopopulated using the Form view of Schema Fields. Navigating between the Schema Fields view and the XML/JSON view will show the request in Form view and Raw view. You can edit the request in either place. The Authentication tab includes options for user credentials. Attachments are available for MIME/MTOM. Tasks provide the ability to define workflow rules to apply to the request before it is sent to the API. The comments tab allows you to provide comments to about the Test Case. The (Sent Request) tab will show the actual live test case request that was sent to the API after all the processing defined in the other tabs was applied.

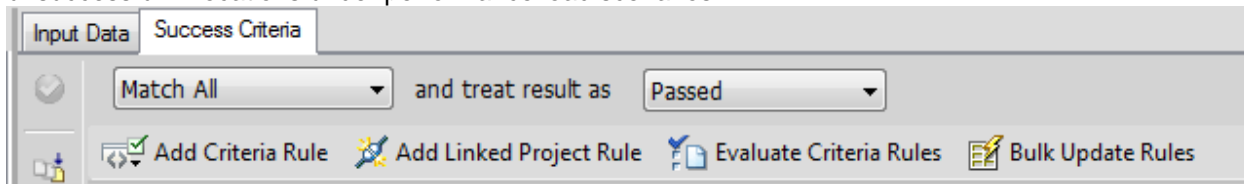
Project View – Test Case Response

Displays panel for viewing the test endpoint response when the current test request configuration is submitted to the API for preliminary result assessment prior to running the test in run mode. The attachments tab is used when the response is MIME or MTOM to show the attachments. The Tasks tab is where response workflow tasks can be defined to manipulate and inspect the response information. Response Variables tab allow the capture of response meta data within nodes, elements, attributes to preserve for use in success criteria or within other test cases.



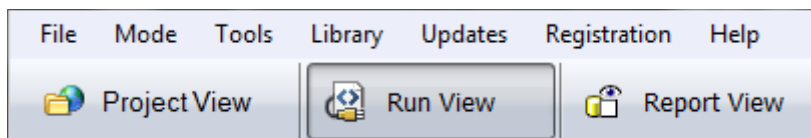
Project View – Success Criteria

Displays the panels used for defining the evaluation criteria rules to assess success or failure for tests. When running in QA mode, this results in the results panel and generated reports to determine a Pass or a Fail for the Test Case. In Performance mode, these rules can be used to determine successful or unsuccessful invocations under performance load scenarios.



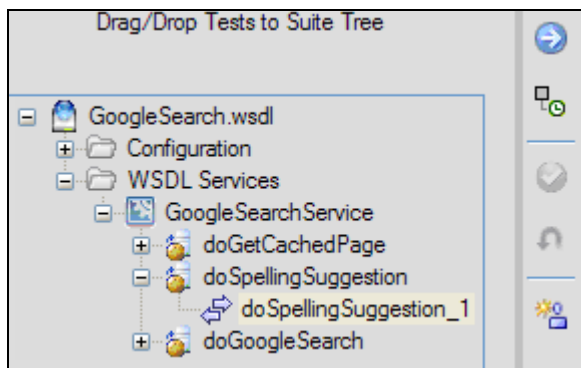
Run View

Opens the panel views for combining tests created in project view into logical sequences and groupings and provides actions to run the tests interactively or via the command-line interface, define test output and other test settings specific to the current run mode.



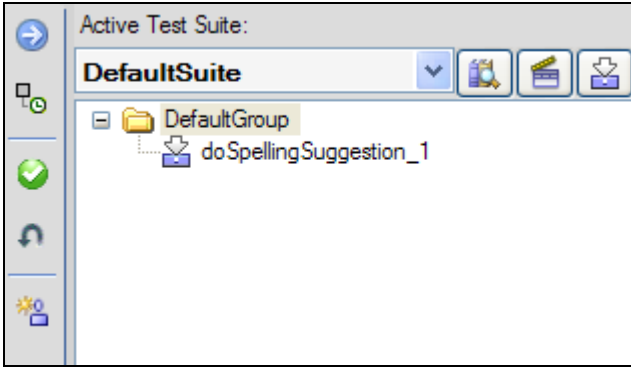
Run View – Project Tree

Displays the same tree as shown in Project view to be used as the source for dragging and dropping test cases to the Suite tree to the right.



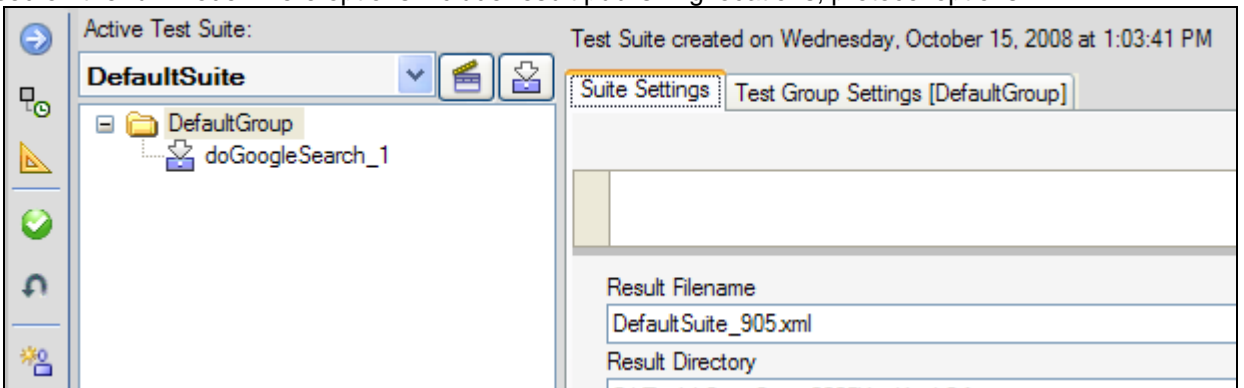
Run View – Suite Tree

Shows the test suites where the project view test cases are logically combined into the entries to run together. Groups can be created for logical test groupings and the test suite and test suite group settings are defined on the right hand screen. The toolbar to the left of the Suite tree is used to interactively run the selected test suite using the real-time monitor, or create the command-line script which can be used to run the selected test suite with the selected settings via the SOAPSonar command-line. In QA mode, an option is presented in the toolbar to generate a baseline regression set for the test suite which can be used for iterative runs of the test suite to measure success based on variance from the stored baseline responses.



Run View – Suite Settings

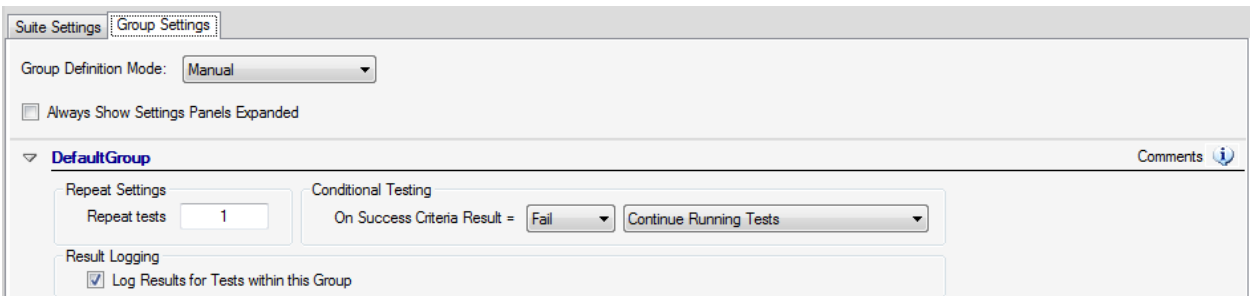
The suite settings panel shows the configuration options for the current selected test suite. Options will vary based on the run mode where options include result publishing locations, protocol options.



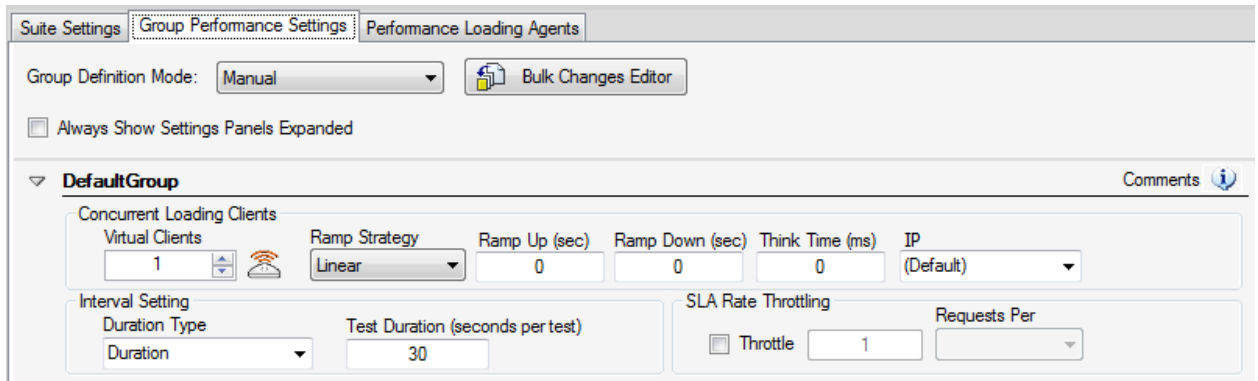
Run View – Test Group Settings

The test group settings panel shows the configuration options for the current selected test group within the active test suite. Options will vary based on the run mode where options presented include conditional test settings and concurrent virtual clients.

For example, in QA mode, the test group settings will provide setting for how many times to repeat the tests in the specified group and conditional testing settings that allow moving to the next test group or aborting the test based on the success criteria results of the test(s) in the group.



In Performance mode, the test group settings provide the settings per test group with regard to virtual clients, ramp time, think time, etc. These settings are discussed in details in the Performance section of this document.



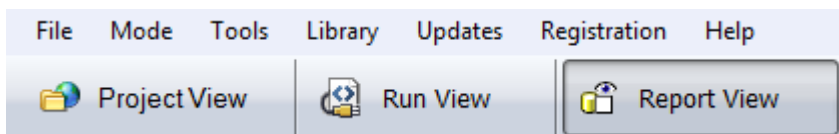
Run View – Run Stats

The run stats panel shows a summary of the test suite run and allows direct link to the detailed reporting assessment by double-clicking the entry in the list.

Run Stats										
Log Filename	Duration	Requests	Transfer Errors	Network Errors	HTTP Errors	Min Bytes	Max Bytes	Min Time (ms)	Max Time (ms)	TPS

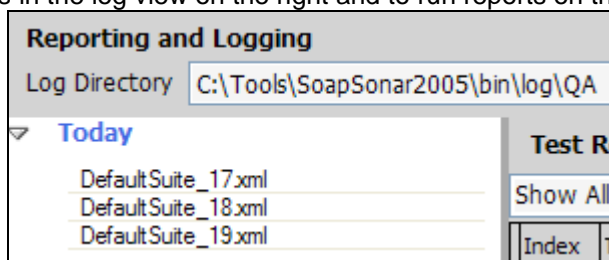
Report View

Opens the panel views for analyzing test results, viewing actual request and response messages, statistics of the test runs, and generating reports.



Report View – Log View Tree

Shows the log files in the current active directory for the selected run mode. Click on the file to see the contents in the log view on the right and to run reports on the results.



Report View – Test Result Summary

Shows a summary view of the test results. Summary view will vary depending on the run mode. In QA mode, success results of the test are shown, in performance mode this will show performance statistics

for each virtual client and the aggregate statistics, for interoperability and vulnerability modes this will show each dynamic mutated request function. Selecting an item shows the details in the panels below.

Report View – Executed Test Request

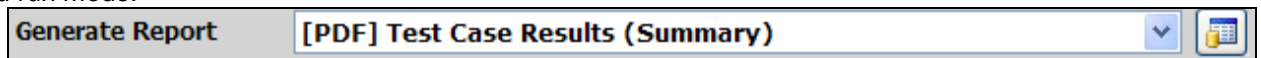
Panel which shows the actual request that was submitted to the endpoint during the test suite run for the selected entry in the test result summary list above.

Report View – Executed Test Response

Panel which shows the actual response that was received from the endpoint during the test suite run for the selected entry in the test result summary list above.

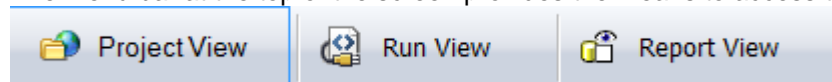
Report View – Generate Report Options

Options for generating various report types. Options for report types will vary according to the currently selected run mode.



Workflow Views

The menu bar at the top of the screen provides the means to access the 3 main workflow areas.



The Project view is where test cases are defined, including automation, success criteria, OpenAPI and WSDL document loading and inspection, building success criteria, and various other tasks related to the test case workflows.

The Run View is where the run-time testing flows are defined. Test Suites represent the ability to combine test cases for running the testing scenarios. Run View is where these are defined via drag/drop tests from the Project view tree to the Run View tree and the applicable runtime settings based on which run mode (QA, Performance, etc) the current view is set to.

The Report View displays the logging and reporting panel where you can choose previous test suite results and analyze the data specific to the run mode under which the tests were executed. Report view provides data analysis along with reporting and export options.

Run Modes

The run modes provide solutions for each of the 4 pillars of API testing. To change run modes, use the Mode menu item, or select the run mode from the right hand portion of the top toolbar.

QA Mode

The QA run mode allows you to configure and run tests for functional testing. In QA run mode, test suites are run to verify the success or failure of the tests according to the specified success criteria for each test. There are a variety of success criteria functional for response analysis including Baseline Regression criteria which can be configured per test suite to run the test cases in the test suite and preserve the responses as the regression baseline by which to compare subsequent runs of the test suite.

Performance Mode

In Performance mode, test suites are run with the specified number of concurrent virtual clients for a duration of time or a specified number of iterations to determine performance statistics of the service such as transactions (request/response pairs) per second, throughput, and response time stats. Performance reports provide metrics for capacity planning purposes.

Compliance Mode

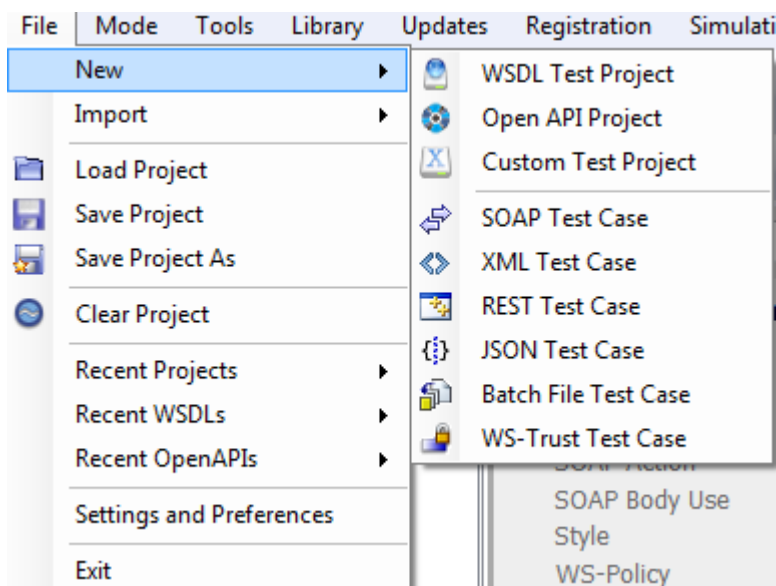
Compliance mode is specific to WSDL-based SOAP testing. Each WSDL service operation can be tested for the actual runtime SOAP WS-I Basic Profile 1.1 web service interoperability compliance. Using patent-pending dynamic mutation technology, the SOAP requests are intelligently mutated to break each requirement in section 3 of the WS-I Basic Profile 1.1 specification to assess the behavior of the back-end server when consuming these errant messages. Compliance reports provide an aggregate view of the active compliance posture of each service operation.

Vulnerability Mode

In Vulnerability mode automatic error boundary condition tests can be auto-generated for each web service operation. Using patent-pending dynamic XSD mutation technology, each SOAP message is intelligently mutated from its original form based on the WSDL XSD which defines the boundary expectations for the request. Using this technology, each web service operation has a unique dynamically generated vulnerability profile to assess the error handling logic of the application server and application service container. Risk assessment and risk mitigation is provided via AVD libraries to assess and analyze responses to report governance and security adherence.

Types of Test Cases

SOAPSonar allows you to test virtually any type of API or service. The different test types include SOAP test case, XML test case, REST test case, JSON Test Case, Batch File test case, and WS-Trust Test Case. Depending on what type of test project is currently selected will dictate what options appear under the File->New menu. These are the same options that appear when right-clicking in the project tree test nodes.



OpenAPI Test Case

A OpenAPI Test Case is automatically created based on an OpenAPI 2.0 or 3.0 document. Each OpenAPI operation is parsed and provides the structure of expected format for the request. A graphical JSON generator is provided to allow the JSON to be defined in form view or in raw view.

SOAP Test Case

A SOAP Test Case is automatically created based on a WSDL and the parsed schema. Each WSDL operation provides the structure of expected format for the request. A graphical SOAP generator is provided to allow the SOAP to be derived from the schema items enabled and populated for the test case.

XML Test Case

An XML test case provides the means to import or create any XML request to send to an endpoint. This test case type can be created under any of the project types and allows a fully customizable framework to defined the test inputs. Despite its name, the XML Test Case type can be used to send XML and non-XML data. Whatever is provided in the input will be sent to the server.

REST Test Case

A REST Test Cases provides a set of input screens that align with HTTP URI, Header, and Body building across various types of REST content-types. It has similar functionality as the XML and JSON test case types, with the focus on automating building the URI and query string parameters.

JSON Test Case

An JSON test case provides the means to import or create any JSON request to send to an endpoint. This test case type is similar to the XML Test Case type, but uses underlying JSON parsing instead of XML parsing.

Batch File Test Case

The Batch File test case type points to a specified directory and using the file filter provided will extract the named files in the directory to run as the data request. A single batch file test case can point to a directory with potentially thousands of files. A test iteration will be generated for each referenced file. Each file will have the authentication, policy, encryption, signatures, and other configured task enrichment applied per the existing configuration allowing the files to each be submitted.

WS-TRUST Test Case

A WS-Trust test case is used to query an STS service for a WS-Trust security token.

OpenAPI Capture and Processing

When testing APIs, an OpenAPI 2.0 Swagger or OpenAPI 3.0 YAML document may be available that provides the contract of information about the expected structure of each request and response for the service to allow clients to build requests to consume the API responses. SOAPSonar parses OpenAPI 2.0 and 3.0 documents for API testing and builds the request templates for all defined operations. When an OpenAPI document is captured, it becomes a new Project item in the project tree with various configuration options for the test environment and creating tests for each API operation.



You can capture an OpenAPI document using several supported methods, including

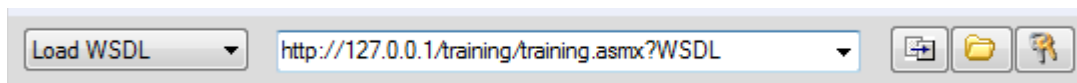
- Drag-Drop – Drag and drop OpenAPI files from a file explorer directly into the project window
- File – Browse your file system for the OpenAPI document to load
- HTTP – Fetch the OpenAPI document using HTTP
- HTTPS – Fetch the OpenAPI document using HTTPS
- HTTP Basic Auth – Fetch the OpenAPI document using HTTP with Basic Authentication Credentials
- HTTPS Basic Auth - Fetch the OpenAPI document using HTTPS with Basic Authentication Credentials
- HTTPS X.509 Auth - Fetch the OpenAPI document using HTTPS and X.509 SSL Authentication
- Project – Load OpenAPI document previously saved to a project file

For http or https access, authentication options are provided using the “Authentication Options” button at the right of the OpenAPI Location field. Both HTTP Basic Authentication and SSL X.509 client authentication are supported. SSL X.509 client authentication is supported through a fully integrated native PKI management interface that allows you direct access to your windows certificates and private keys. Simply select the certificate you want to use for SSL client cert from the list of X509s loaded on your local machine.

Once the OpenAPI document is captured and parsed, the project tree will show a graphical representation of the OpenAPI contents. Each node in the project tree will show properties specific to the type of object (i.e. Service Tag, OpenAPI Operation, Test Case). When you select an item in the tree, the available options for that item will appear on the right portion of the screen.

WSDL Capture and Processing

When testing web services, a WSDL document provides the contract of information about the expected structure of each request and response for the service to allow clients to build requests to consume the services responses. SOAPSonar parses WSDL documents for client emulation testing and diagnostics of the service consumption and response behavior. When a WSDL is captured, it becomes a new Project item in the project tree with various configuration options for the test environment and creating tests for each service operation.



Capturing a WSDL document will automatically build SOAP test cases and populate the requests based on the settings enabled in the File->Settings panel. You can capture a WSDL document using several supported methods, including

- Drag-Drop – Drag and drop WSDL files from a file explorer directly into the project window
- File – Browse your file system for the WSDL to load
- HTTP – Fetch the WSDL using HTTP
- HTTPS – Fetch the WSDL using HTTPS

- HTTP Basic Auth – Fetch the WSDL using HTTP with Basic Authentication Credentials
- HTTPS Basic Auth - Fetch the WSDLs using HTTPS with Basic Authentication Credentials
- HTTPS X.509 Auth - Fetch the WSDLs using HTTPS and X.509 SSL Authentication
- Project – Load WSDLs previously saved to a project file

For http or https access, authentication options are provided using the “Authentication Options” button at the right of the WSDL Location field. SOAPSonar supports both HTTP Basic Authentication and SSL X.509 client authentication. SSL X.509 client authentication is supported through a fully integrated native PKI management interface that allows you direct access to your windows certificates and private keys. Simply select the certificate you want to use for SSL client cert from the list of X509s loaded on your local machine.

Once the WSDL is captured and parsed, the WSDL project tree will show a graphical representation of the WSDL contents. Each node in the WSDL project tree will show properties specific to the type of object (i.e. Service, Operation, Test Case). When you select an item in the tree, the available options for that item will appear on the right portion of the screen.

Working with Projects

All the settings configured in the Project view and Run view are preserved when you save a project file. Project files are stored as .ssp files and can be shared among team members, checked into version control to synchronize with service release versions, and are used by the command-line interface to access the test suites and test data to execute. Project management also allows subprojects to be combined into master projects and merging awareness will allow merging of projects and subprojects.

Settings and Preferences

Various configuration settings are provided to allow customization of the test cases, such as SOAP generation for WSDL project parsing behavior, default SSL/TLS settings, Global Variables, etc. To configure these settings, go to the File->Settings and Preferences dialog and configure the settings. For more information on each setting, go to the project settings section.

WSDL Parsing and SOAP Generator Settings

SOAP Settings

SOAP Settings

Add xsi:nil="true" attribute rather than omit element when checkbox is enabled

Define namespace prefix mapping in SOAP Body child (instead of SOAP Envelope)

Preserve whitespace when formatting SOAP

Ignore SOAP header for XSD Validation (when invoked in Project View Test Menu)

Default WSDL SOAP Binding

Add xsi:nil="true" attribute rather than omit element when checkbox is enabled

With this option enabled, a checkbox placed in front of an element in the fields view will result in the generated SOAP message having an xsi:nil="true" attribute added to the element node (i.e. <test xsi:nil="true"/>). When this option is disabled (default), a checkbox in front of the element in Schema

Fields view will result in the element being omitted from the generated SOAP request. The `xsi:nil="true"` setting is required by some servers which expect SOAP encoded requests to preserve the XML structure with nil contents explicitly specified.

Define namespace prefix mapping in SOAP Body child (instead of SOAP Envelope)

This option allows you to choose where the document prefix namespace mappings are defined. While both options produce semantically equivalent documents, some non XML-compliant back-end servers do not properly recognize namespaces defined in the SOAP Envelope and expect them rather to be defined within the first SOAP Body child element. Checking this option will result in the generated SOAP messages to define the namespaces in the SOAP Body child. The default unchecked behavior will create the prefix namespace references in the SOAP element.

Preserve whitespace when formatting

This option allows you to choose to preserve whitespace characters when formatting the message to be sent. This option would be used if you want to send blank spaces within XML tags.

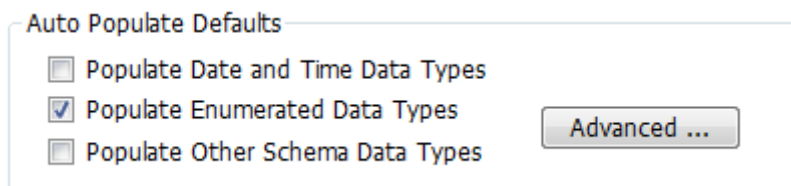
Ignore SOAP header for XSD Validation

This option allows you to ignore the contents of the SOAP header when validating SOAP message contents.

Default WSDL SOAP Binding

This option determines which SOAP version to attempt to parse from the target WSDL, SOAP 1.0, or SOAP 1.1.

Auto populate Defaults



Auto Populate Defaults

- Populate Date and Time Data Types
- Populate Enumerated Data Types
- Populate Other Schema Data Types

Advanced ...

Populate Date and Time Data Types

Auto-detect parameter items which have a date or time data type and populate the value with a date/time value representing the current date and time.

Populate Enumerated Data Types

Auto-detect parameter items which have enumeration facets and populate the values with the first enumerated item in the referenced items.

Populate Other Data

Auto-detect the type based on the XSD schema and provides a schema compliant value representative of the referenced data type.

Advanced Settings

The advanced settings provide some additional automatic data populate options. For optimal performance when parsing WSDLs and working with test cases, it is recommended to keep these settings as their defaults.

Allocate Arrays

Auto-detect array items represented as schema objects of complex or simple type with attribute of `maxOccurs > 0` and include these object in the default build SOAP request. When dealing with

complex schemas, it is recommended to keep this setting unchecked and manually from the Schema Fields view enable each instance of an array to be included in the request.

Enable Attributes

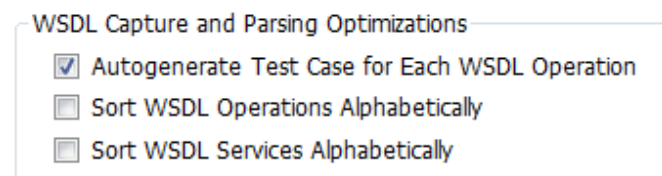
Auto-detect attribute values and enable all of them. When dealing with complex schemas, it is recommended to keep this setting unchecked and manually from the Schema Fields view enable each attribute to be included in the request.

Enable Optional Elements

Auto-detect SOAP elements defined with XSD schema attribute minOccurs="0" mean that they are optional elements and may not need to be present in the SOAP request. This setting allows you to enable all instances of optional elements such that they all appear in the generated SOAP. When dealing with complex schemas, it is recommended to keep this setting unchecked and manually from the Schema Fields view enable each SOAP element to be included in the request.

WSDL Capture and Parsing Optimizations

The settings within this section need only be altered if you are experiencing excessively long delays in parsing the WSDL and rendering the graphical items in the Schema Fields view. Delays usually indicate that there are complex recursive schema definitions causing the number of permutations for test configurations becomes CPU intensive for SOAPSonar to represent graphically. The optimization settings will detect these hierarchical schema declarations and limit the depth of recursive processing.



Auto generate Test Case for Each WSDL Operation

To improve speed in creating new WSDL projects, disable the automatic test case generation to allow you to manually create each test case from the selected WSDL operation

Sort WSDL Operations Alphabetically

This setting applies to newly parsed WSDLs. To display the WSDL operations in alphabetical order, check this setting. The default unchecked behavior will display the WSDL operations in the order they are listed within the WSDL.

Advanced Settings

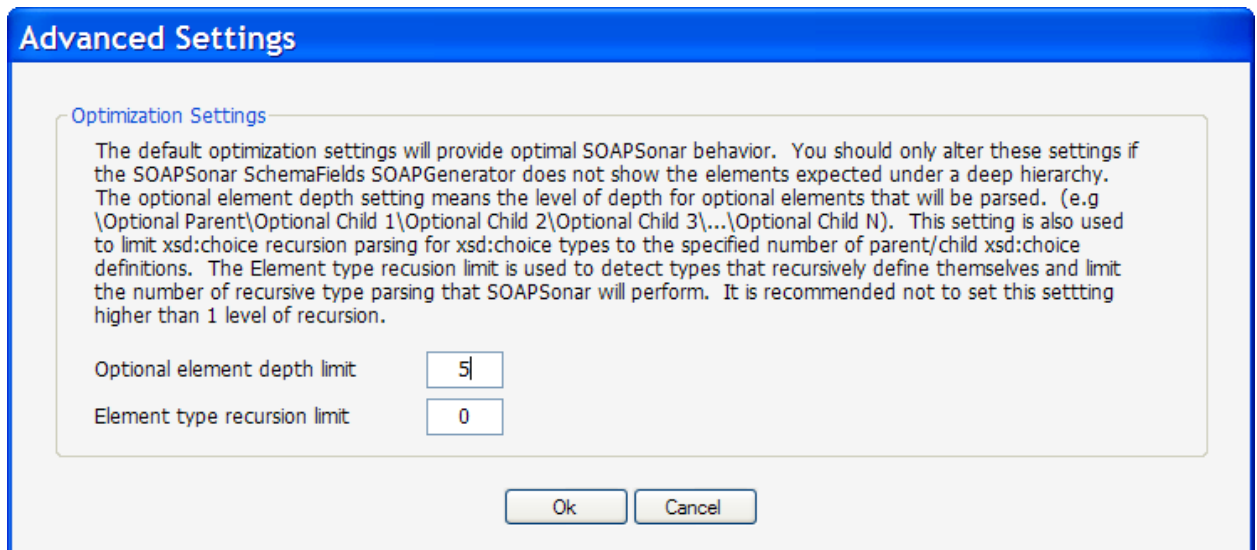
The advanced settings allow you to change the default optimizations for SOAPSonar. For optimal performance when parsing WSDLs and working with test cases, it is recommended to keep these settings as their defaults.

Optimize Schema Optional Element Parsing Depth

Optional Elements are elements defined as minOccurs=0, or having a nillable=true attribute. These elements do not need to be included in the SOAP request. This optimization depth parsing setting will limit the recursive depth of optional elements with optional children elements to a limited depth of recursion (default = 5). Alter this setting if you do not see the elements you expect to see in Schema Fields view.

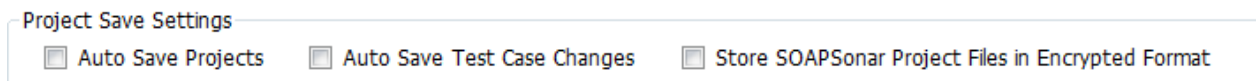
Optimize Recursive Schema Definition Parsing Depth

Sometime schema elements are defined recursively to include themselves. This is poor practice for schema authoring and can cause significant overhead in repeatedly parsing the same type declarations over and over. This setting allows recursive parsing to the specified depth. The default setting is 0 to prevent recursive type parsing. Alter this setting if you see the elements you expect to see in Schema Fields view, but it is only a text field, and not a complex type.



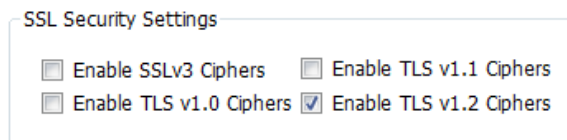
Project Save Settings

These settings allow you to define whether project are auto-saved periodically, after each test case change, and whether to store the projects in encrypted format.



SSL Security Settings

These settings define the default cypher suites to use when using HTTPs in test cases.



Test Suite Settings

The Suite Settings tab sets the default locations where test suite results are to be written. Choose the folders which all suites will write results to by default, and also lock these settings for all existing and newly created or imported suites. The lock feature is useful when working in environments where you are sharing SOAPSonar project files as the lock behavior will ensure that imported projects have the locations for test results be initialized with these settings such that they are applicable to each installed SOAPSonar instance.

Project Test Settings | **Suite Settings** | Email Settings | Global Proxy Settings

Suite Result Directories

Suite Log Destination Setting: Lock New and Existing Suites to these Settings

QA Mode Result Directory: C:\Tools\SoapSonar2005\bin\log\QA

Performance Mode Result Directory: C:\Tools\SoapSonar2005\bin\log\Performance

Compliance Mode Result Directory: C:\Tools\SoapSonar2005\bin\log\Compliance

Vulnerability Mode Result Directory: C:\Tools\SoapSonar2005\bin\log\Vulnerability

Suite Protocol Settings

Update Existing Suite Values

Protocol: HTTP 1.1 | Response Timeout (sec): 10 | Wait Time (ms): 200

Suite Result Directories

Specify the result directories to publish test results for running suites within each run mode. Lock the settings so that any imported projects will adhere to these local machine settings.

Suite Protocol Settings

Specify the default protocol settings for created test suites. Check the box for updating existing values to globally replace all test suite protocol values with the values specified.

Performance Mode Transaction Tracing Settings

If using performance mode log tracing, the log size can grow rapidly. This setting provides the limit of how large the tracing file can grow to.

Reporting Settings

The graphical diff parsing can be set to treat whitespace as a difference, or to ignore whitespace based on the Ignore Whitespace checkbox.

Graphical Diff Report Generation

Ignore Whitespace

Email Settings

Settings for the SMTP server and from email address that are used by the command-line interface when the email report option has been enabled.

SMTP Settings

SMTP Mail Server Port

Use Windows Credentials

Specify Username / Password Credentials

User

Pwd

Use SSL/TLS

From Email Address

Send Test Email

Global Proxy Settings

Enables the use of a proxy server for test case execution and when capturing documents such as OpenAPI or WSDL documents from a target HTTP/s location that requires proxy access.

Global Proxy Settings

Use HTTP Proxy for All Requests

Proxy URL Example (http://myproxy.example.com:port)

Credentials

Default Security Credentials (NTLM / Kerberos)

Specify

Username

Password

Global Variables

Defines the global variables for current product instance. Manual definitions can be entered as name=value pairs, one per line. File definitions allow global variables to be defined in external files and referenced. View Current Variables will show all loaded global variables.

Manual Definitions File Definitions View Current Variables

Global Variables (name=value)

Script Includes Variables

When using the VBScript or JScript task functions, this screen allow the definition of the include files that can be used to include VBScript or JScript libraries.

HP Quality Center

Defines the integration settings for project instances licensed to support HP Quality Center integration. For more information about this feature, refer to the HP Quality Center integration guide.

BUILDING TESTS

Project view is where the various types of API tests can be created based on the target API, service or application's message and protocol formats. Each test case type is highly extensible allowing you to create virtually any type of API testing paradigm.

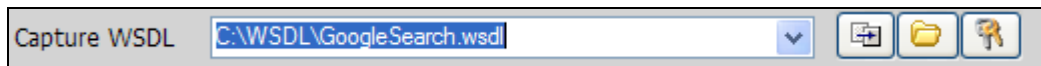
Subtopics in this section include

- OpenAPI Test Case
- [SOAP Test Case](#)
- [Schema Fields Input Generator](#)
- [XML Test Case](#)
- [Batch File Test Case](#)
- [REST Test Case](#)
- [JSON Test Case](#)
- [Test Case Request - Configuration Parameter Tabs](#)

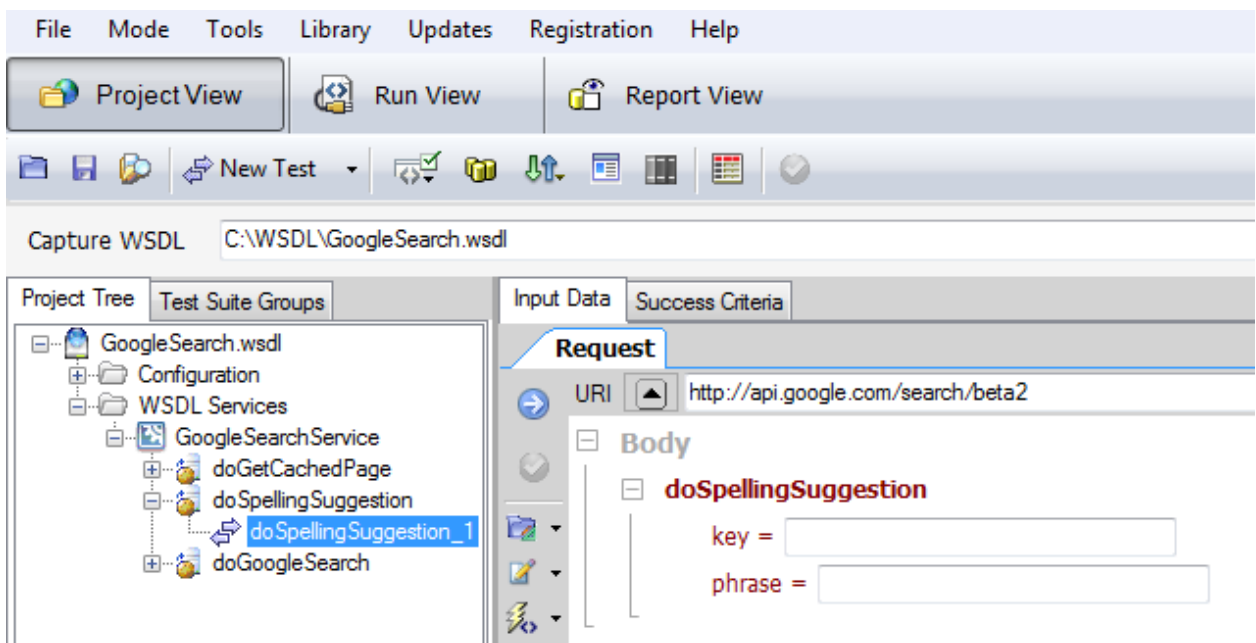
SOAP Test Case

A SOAP Test Case is automatically created based on a WSDL and the parsed schema. Each WSDL operation provides the structure of expected format for the request. A graphical SOAP generator is provided to allow the SOAP to be derived from the schema items enabled and populated for the test case.

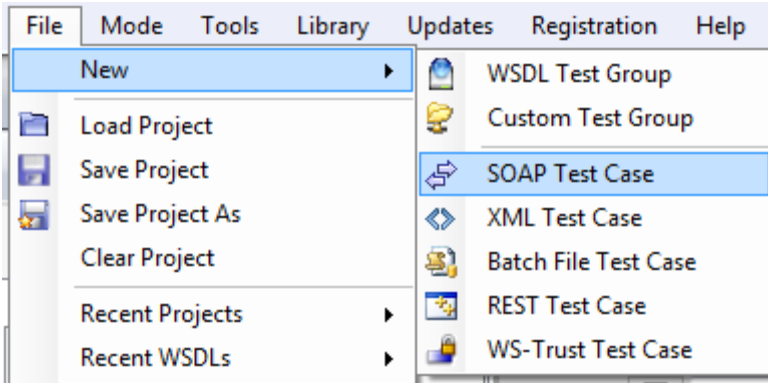
To create a SOAP test cases from a WSDL document, first go to the Capture WSDL area, then choose a file or network location where the WSDL resides. Once a WSDL document is captured and parsed, the project tree will show a set of nodes representing the parsed WSDL and providing access to global test settings and test case generation.



By default a test case will be automatically created for each WSDL operation. The test case node can be accessed by navigating to the WSDL operation node and expanding the contents. Click on the test case node to access the test case configuration panels.

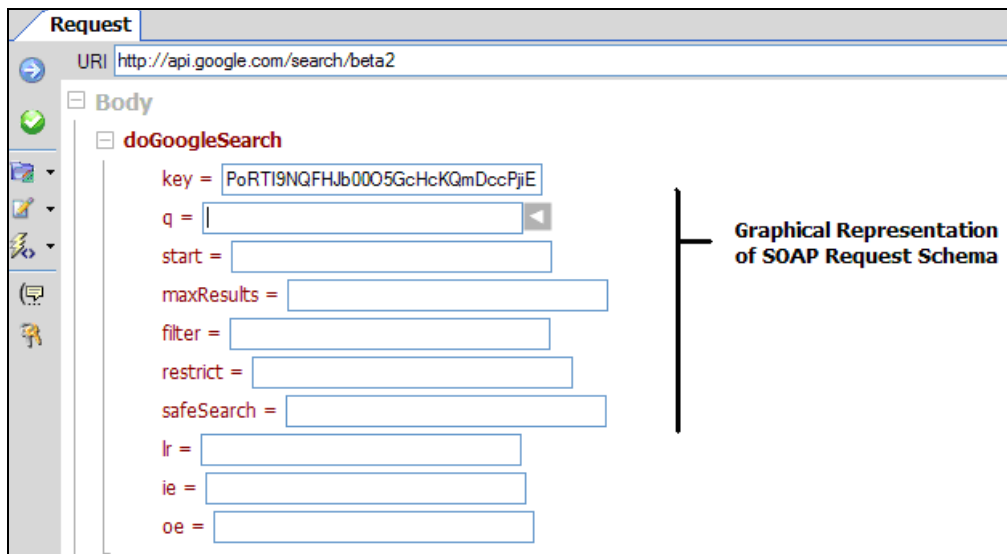


WSDL SOAP test cases have a graphical SOAP generator called Schema Fields. The Schema Fields graphical objects provide a means to define schema contents to generate the resulting SOAP request structure. To create new SOAP test cases, right-click the WSDL operation node, or use the File->New->SOAP Test Case menu item. In order for the Schema Fields SOAP generator to know the appropriate schema to use for the request generation, the WSDL operation to invoke must be selected when creating the SOAP test case type.



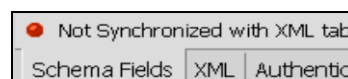
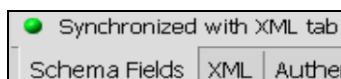
Schema Fields Generator

The Schema Fields tab provides a graphical set of objects representing each schema element applicable for the SOAP request for that WSDL operation. Each type of schema declaration has a corresponding GUI widget to configure it. The data provided in the fields will be used to generate the SOAP message for the test case.



GUI widgets that dynamically appear will be based on the data type of the schema element that is being represented. Each type of [data entry widget](#) is explained below.

For custom XML or custom SOAP, the green icon labeled **Synchronized with XML tab** just above the Schema View tab allows the Schema Field SOAP generator to be disabled. When this icon is green, the XML tab will show the resulting generated SOAP message based on the schema mappings defined. When this icon is red, the SOAP generator is disabled and data can be directly imported or defined on the XML tab.



Features of Schema Fields Generator

In addition to providing easy parameter entry for SOAP values, the Schema Fields view also provides a rich set of features that simplify creation of test cases and test data entry editing. The types of GUI items that appear based on the underlying schema type are described below.

Data Entry: Dynamic Array Structures

For XSD elements defined with `maxOccurs="N"` and `N > 0`, a dynamic array boundary graphic will appear for this array object to allow you to choose the array boundary (i.e. the number of occurrences to generate for this element and its child elements)

The image shows two examples of dynamic array structures in the Schema Fields Generator. The top example shows a field labeled 'field' with a value of 1. To its right is a text box containing the XML snippet: `<dns:field>sample field value</dns:field>`. The bottom example shows a field labeled 'field' with a value of 5. To its right is a text box containing five XML snippets, each with a unique sample value: `<dns:field>sample field value 1</dns:field>`, `<dns:field>sample field value 2</dns:field>`, `<dns:field>sample field value 3</dns:field>`, `<dns:field>sample field value 4</dns:field>`, and `<dns:field>sample field value 5</dns:field>`.

For example, the above setting would generate 1 instance of the detail level element. Setting this value to 5 would generate 5 instances of the detail level element, each new instance having a separately defined parameter.

The image shows a field labeled 'field' with a value of 5. To its right is a dynamic array boundary icon. A callout box points to the icon with the text: **Click icon to dynamically allocate array, or to map array items to a data source**.

Data Entry: Optional Elements

For XSD elements defined with `minOccurs="0"` or with a nillable attribute value of `true` this means that the element may be omitted from the message. For these elements, a checkbox will be visible to include or omit the element when generating the SOAP request.

The image shows an optional element labeled 'Echo'. Below it is a checkbox for 'Buf =' which is checked.

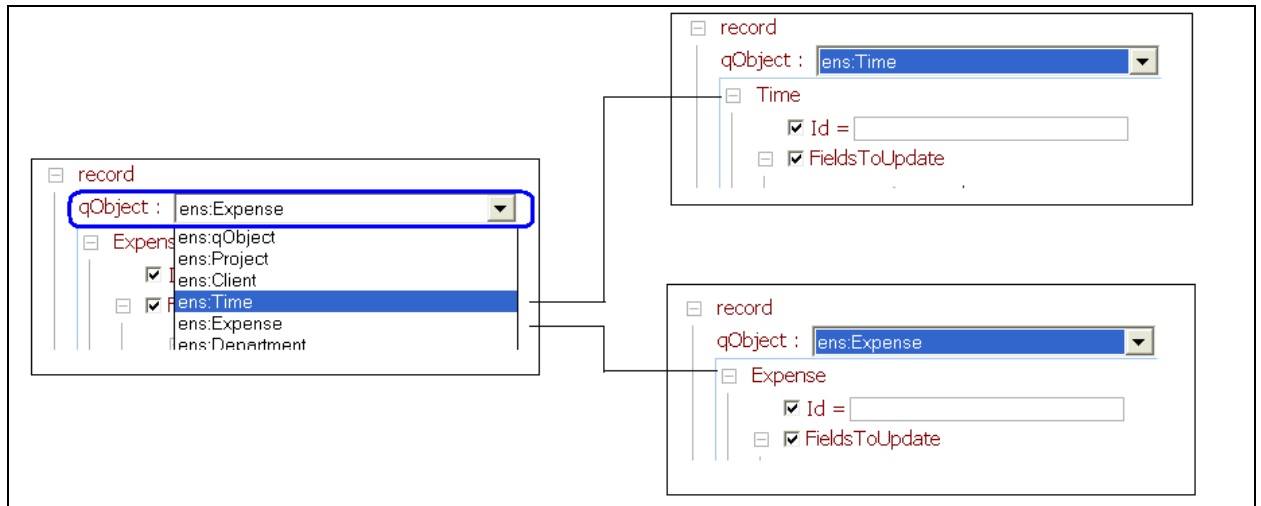
Data Entry: Optional Attributes

Attributes appear with checkboxes so that you can omit or include the attribute when generating the SOAP request.

The image shows three optional attributes: 'language =' (checked), 'id =' (unchecked), and 'href =' (unchecked).

Data Entry: Abstract Types (XSD Polymorphism)

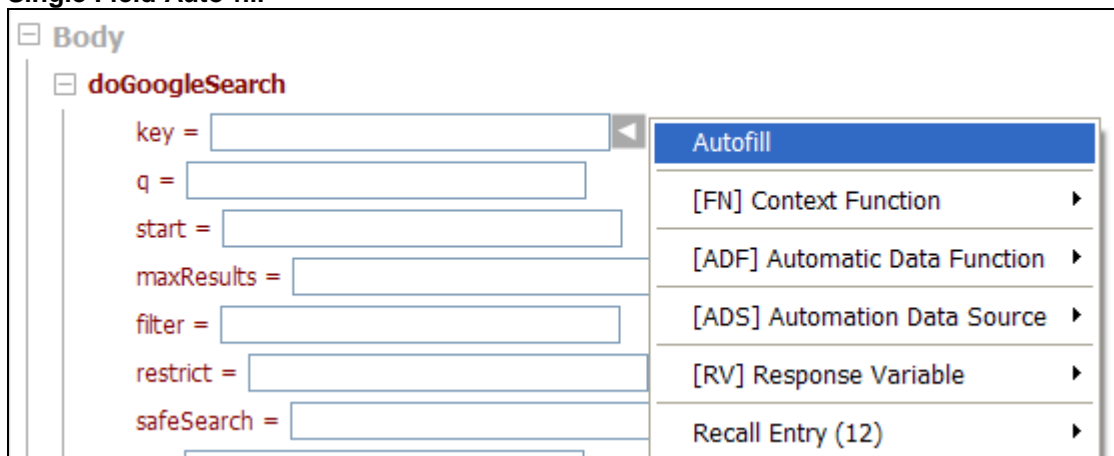
For XSD elements defined as abstract=true, where other XSD elements extend the base abstract type, this correlates to the notion of late-binding (the binding of a data-type to an object after the object has been instantiated). Each extended type based on the abstract type will appear in a pull-down list to dynamically alter the binding of the extended type to use when generating the SOAP request:



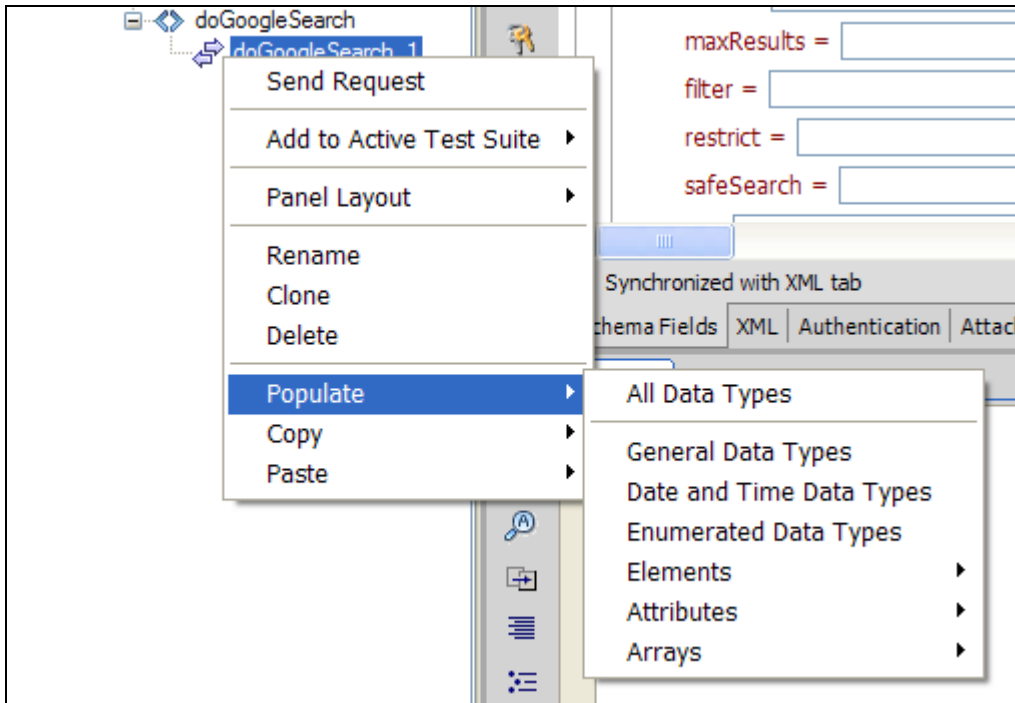
Data Entry: Auto fill

To generate a data value automatically which is schema compliant based on the XSD data type, select Auto fill from the right field option menu. Each generated value auto-detects the schema type. To automatically fill all data type values, right-click on the test case node in the project tree and select the Populate option.

Single Field Auto-fill

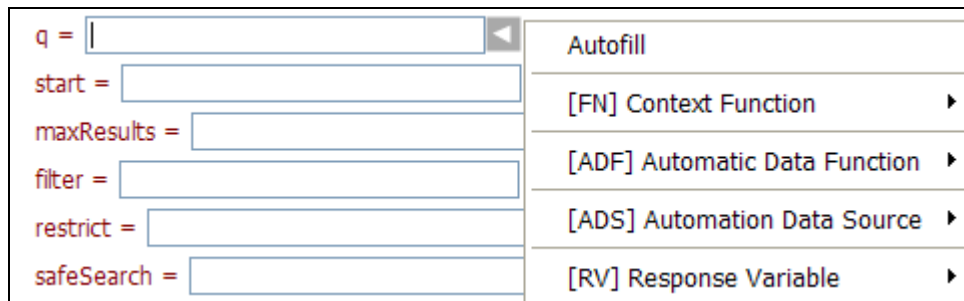


Multiple Field Auto-fill



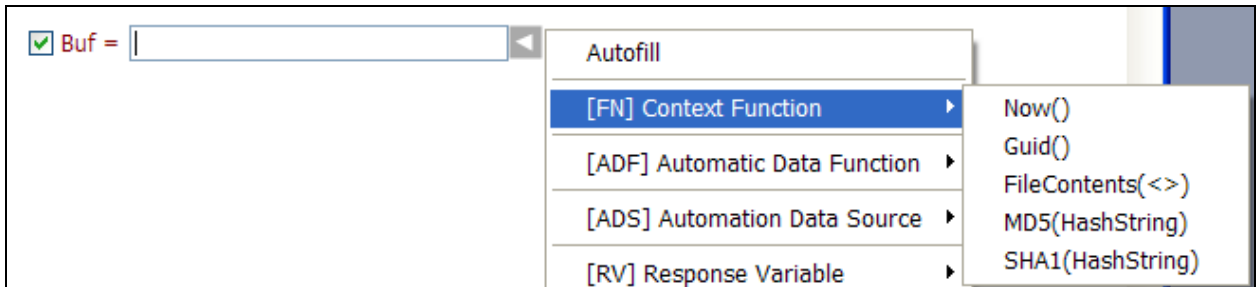
Data Entry: Variable Parameters

Request values can be defined statically, or defined as variables. Variable options include data source driven testing, data inputs created via ADF libraries, or response values from dependent test cases. To choose a variable parameter, right-click on the test case node in the project tree and select the type of variable to create.



Data Entry: Context Functions

Context Functions provide inline data processing with value replacement occurring each time the request is sent. (Context functions and parameters are discussed in more detail in the [Context Function](#) section).



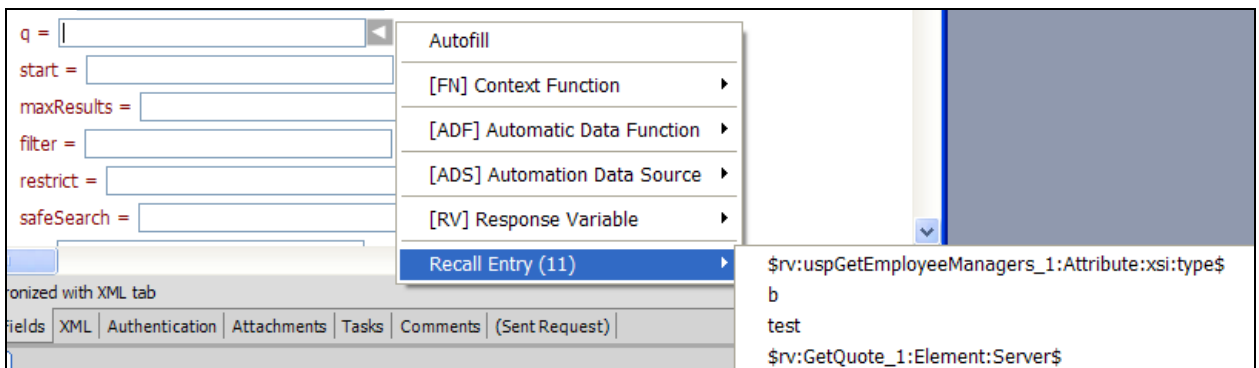
Data Entry: Entry Recall

A handy editing feature which is commonly seen in web browsers is also provided by SOAPSonar. The entry recall tracking will recall all values previously entered into this parameter field. Values are stored and recalled based on each WSDL project loaded.

This feature can be invoked by pressing the <down-arrow> key while the cursor is in the parameter field



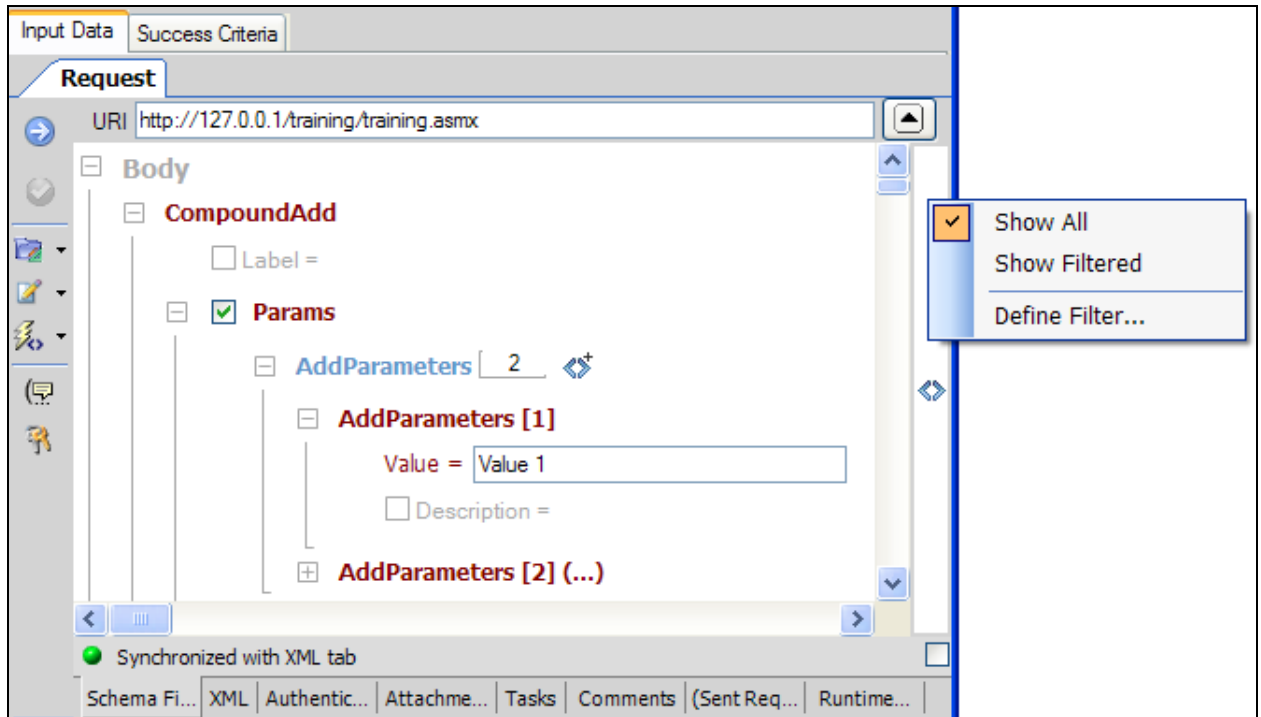
You can also use the context menu shown at the right of the control to select the “Recall Entry” option and select from the list of previously entered items.



Schema Fields View Filtering

By default all schema items related to the selected WSDL operation will be displayed for allocation when using the SOAP generator. This allows you to build any variation of SOAP as defined by the schema of the WSDL. However, often in practice only a subset of these schema items are of interest to view and allocate.

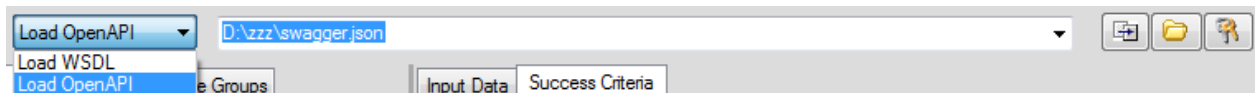
You can choose to hide fields that you do not want to allocate and filter the display to only show the fields of interest to allocate values for the generated SOAP. To access the filtering options, click on the menu bar to the right of the schema fields area. Options include “Show All”, “Show Filtered”, and “Define Filter...”. Choose Show All to view all schema field items, choose Show Filtered to filter the schema items to those defined the last time the “Define Filter” option was selected and the filter was defined.



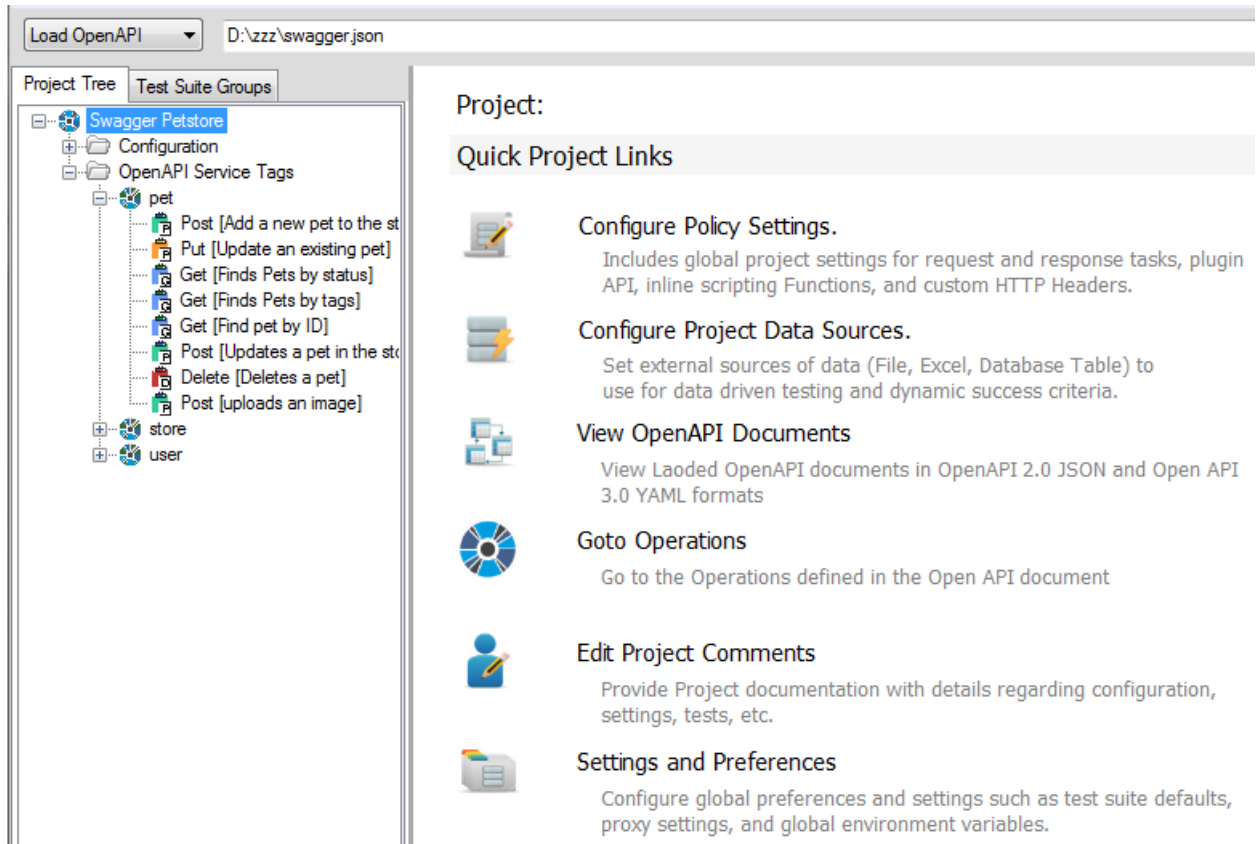
OpenAPI Test Case

An OpenAPI Test Case is automatically created based on a loaded OpenAPI 2.0 or 3.0 Swagger or YAML document. The OpenAPI operations are parsed and extracted from the document and the project will show all defined operations and enable OpenAPI test cases to be generated using the underlying OpenAPI definitions as the template. A graphical schema fields view enables a Form based generator to build the JSON request, or the JSON request can be edited in Raw format from the JSON tab.

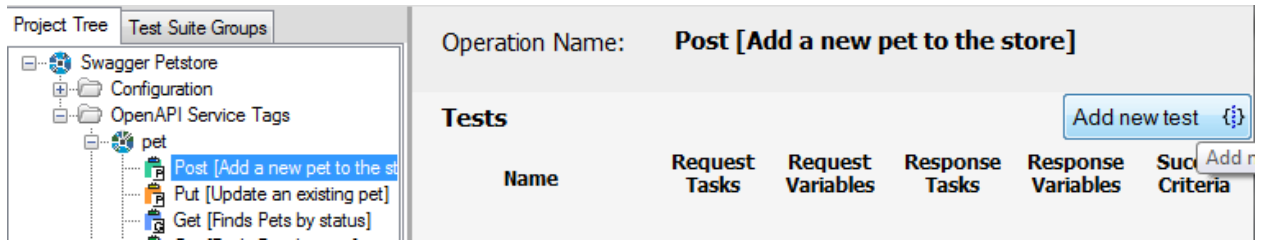
To create OpenAPI test cases from an OpenAPI document, first go to the top left of the Project view screen and select “Load OpenAPI” from the Load selection and then choose a file or network location where the OpenAPI document resides. Once an OpenAPI document is captured and parsed, the project tree will show a set of nodes representing the parsed OpenAPI document and providing access to global test settings and test case generation.



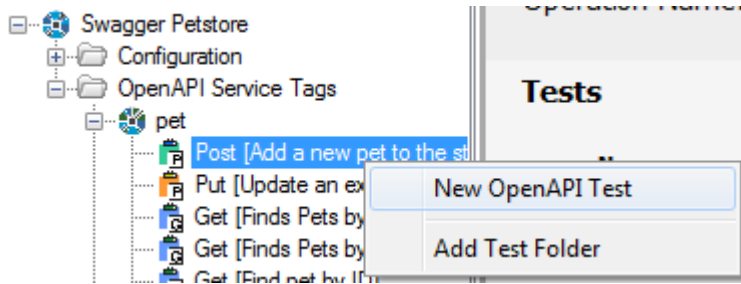
Each Service Tag and Operation defined within the OpenAPI document will be parsed and shown in tree format.



To create a new OpenAPI test case, simply click on the operation and choose “add New Test Case” button



or right-click on the OpenAPI operation and choose



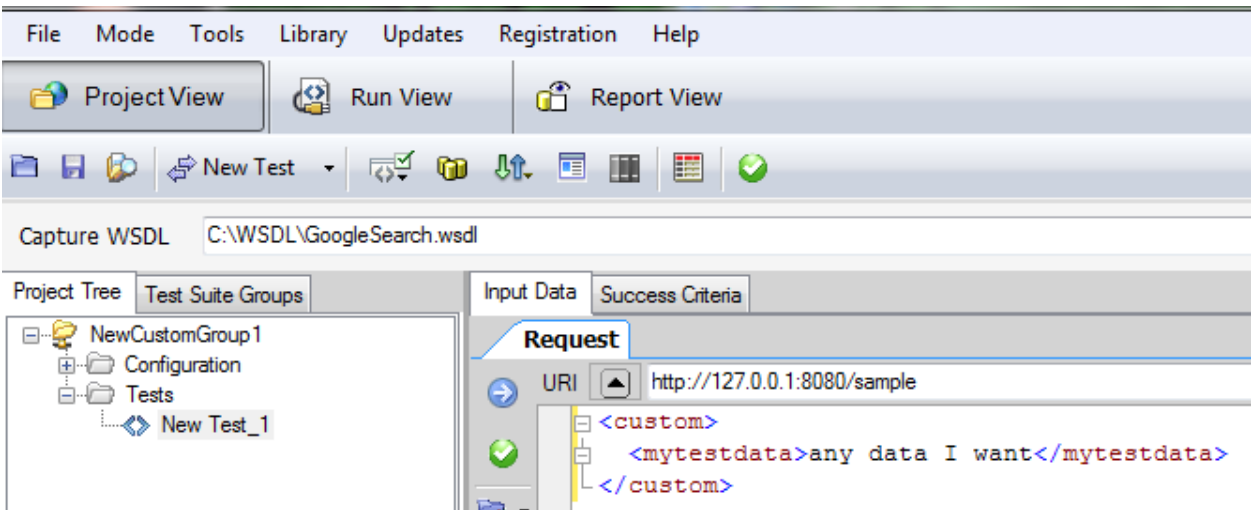
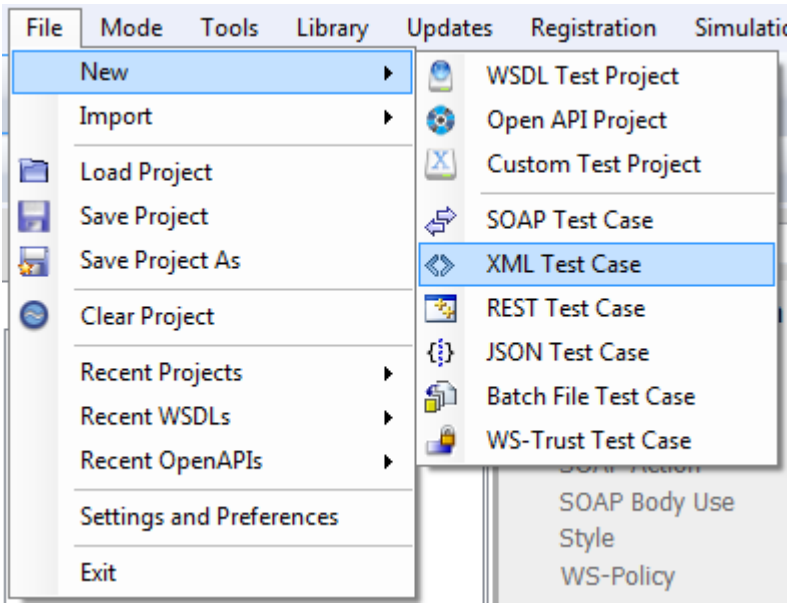
Schema Fields Generator

The JSON message format can be generated using the Schema Fields view or via the JSON tab to see the raw message view. The schema fields generator works the same for OpenAPI JSON test case messages as it does for WSDL-based SOAP messages. Refer to the [Schema Fields Generator](#) section under SOAP test case type for more details about the Schema Fields message generator.

XML Test Case

An XML test case provides the means to import or create any XML request to send to an endpoint. This test case type can be created under a WSDL Operation or under a Custom Test Group. The XML Test Case type can be used to send XML and non-XML data. Whatever is provided in the input will be sent to the server.

To create an XML test case for a WSDL project, navigate to an existing WSDL project operation node and select File->New->XML Test Case. To create an XML test case for any endpoint, create or select a Custom Test Group, then select File->New->XML Test Case. The XML test case type allows you to load the request data from file, paste from clipboard, or type in the request data manually. Variable parameters are supported anywhere within the XML data section and the HTTP Header for ADS, ADS, RV, and Context variables.

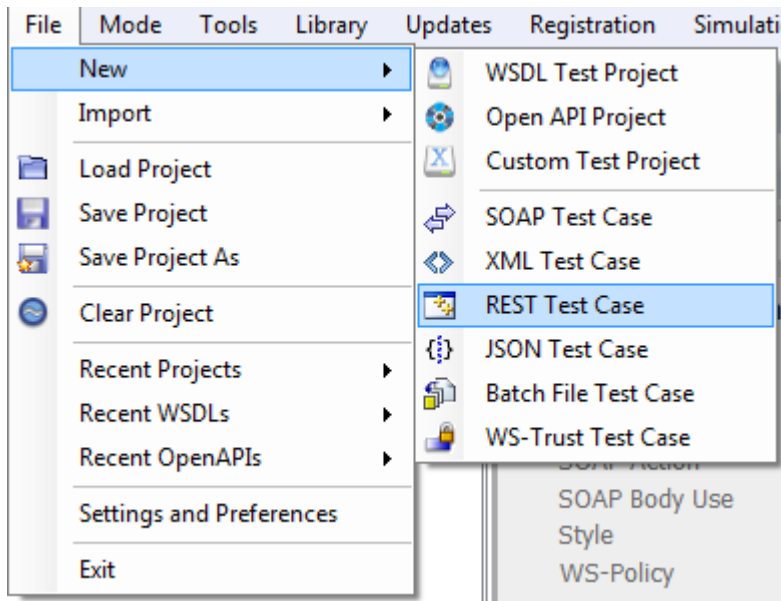


After defining the request data, the additional tabs on the request panel can be used for additional authentication and enrichment settings.

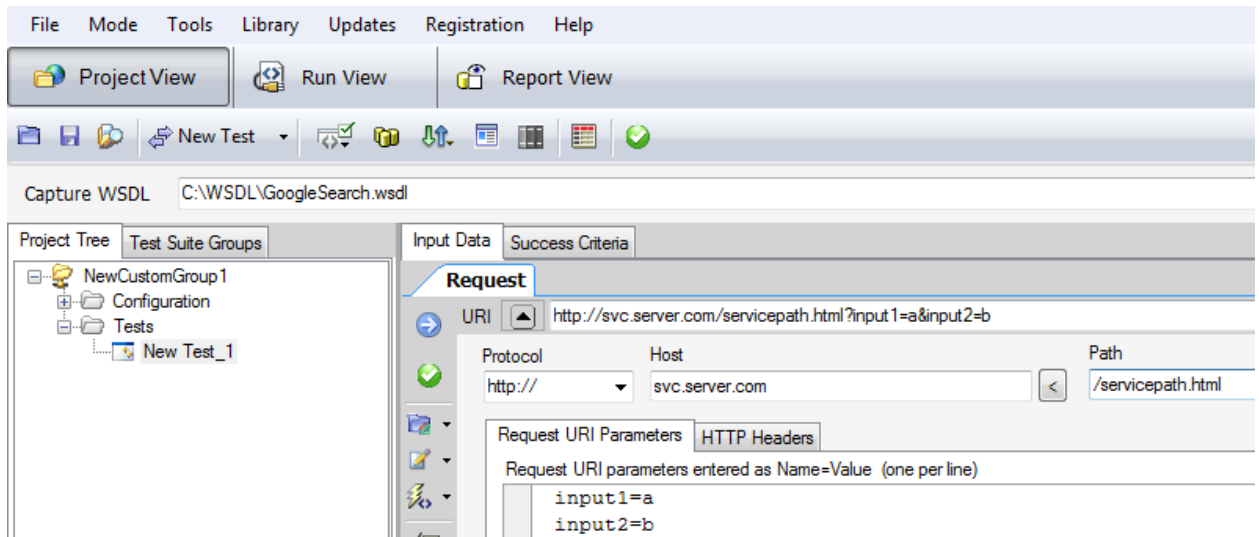
REST Test Case

To test REST services, the REST test case type provides test configuration settings specific for the GET request.

You can create a REST test case to send an HTTP GET query with custom parameters. You can create a REST test case by going to the File menu and first creating a Custom Test Group, then creating a REST Test case.



For REST test cases, selecting the test case node in the project tree will display the REST Query tab on the request panel. The REST Query tab can be used to enter name/value pairs for the URI. Variables are available for dynamic URI value replacement by selection the option menu items on the right of the data entry field. Press the Update URI button to build the resulting URI that will be used to retrieve the data from the REST request.



Generate HTML Form Post Request Builder

In a REST Test Case, if you want to do testing for application/x-www-form-urlencoded or multipart/form-data requests, SOAPSonar provides a graphical tool to build these requests and provides a script format to accomplish web form posts and URL encoded HTTP form posts with socket streaming technology that enables large file transfers to be accomplished with low memory overhead.



When in the REST Test Case, click on the icon that appears on the toolbar which will bring up the graphical editor. The HTML Form Data Format Generator enables selecting the content-type, and if you are using form-data, entering in the file parameter and the target location.

To use the HTML Form Data generator, choose the content-type, and then use the table below to enter name/value pairs or if using multipart/form-data content-type, you can use the "@" Symbol in the Name Column with the name of the field, and use the Value column to reference the file.

i.e.

Name	Value
@Name	Filename

Content-Type: **application/x-www-form-urlencoded**

	Name	Value
1	parameter	value
2	@file	c:\temp\myfile.bin
3		

When you press OK on this dialog, the resulting REST Body tab on the request will be populated with the special scripting that identifies the request as a form post, and the data to be posted.

Input Data | Success Criteria

Request

URI: http://127.0.0.1/

```
$fn:HTMLFormPost (START) $
Content-Type: multipart/form-data; boundary=c5c6d816-79bc-491f-886e-37ecf63a6483
parameter,value
@file,c:\temp\myfile.bin
$fn:HTMLFormPost (END) $
```

REST URI | REST Body | Authentication | Tasks | Comments | (Sent Request) | Runtime Variables

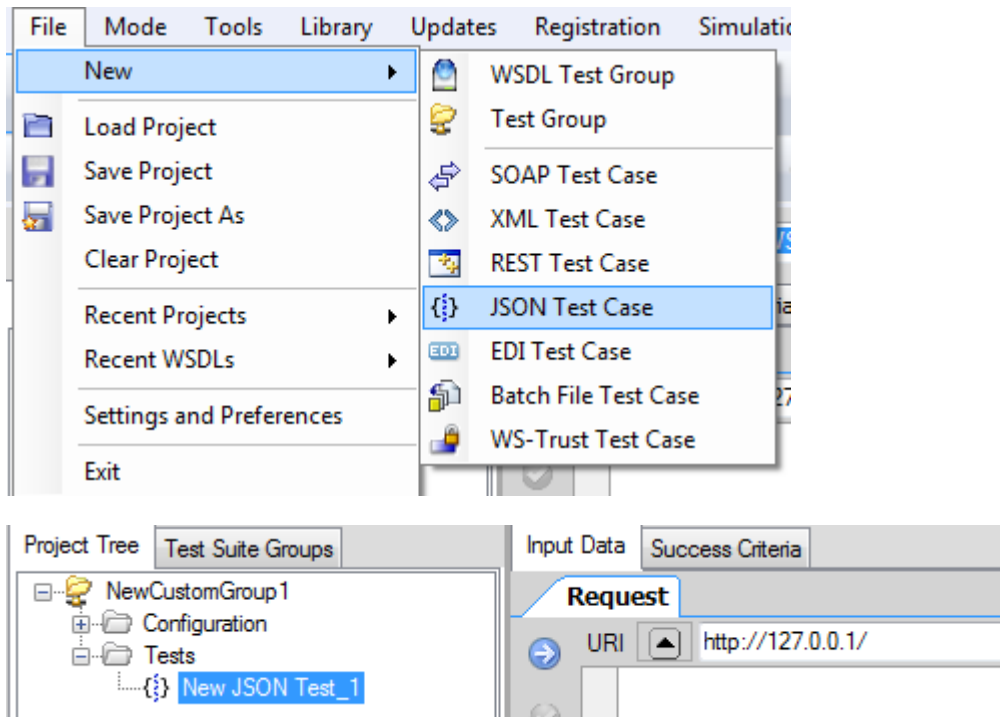
You can edit this script directly as well if preferred:

```
$fn:HTMLFormPost (START) $
Content-Type: multipart/form-data; boundary=c5c6d816-79bc-491f-886e-37ecf63a6483
parameter,value
@file,c:\temp\myfile.bin
$fn:HTMLFormPost (END) $
```


JSON Test Case

A JSON test case provides the means to import or create any JSON request to send to an endpoint. This test case type can be created under a Custom Test Group.

To create an JSON test case for any endpoint, create or select a Custom Test Group, then select File->New->JSON Test Case. The JSON test case type allows you to load the request data from file, paste from clipboard, or type in the request data manually. Variable parameters are supported anywhere within the JSON data section and the HTTP Header. Simply right-click to see variable substitution options.

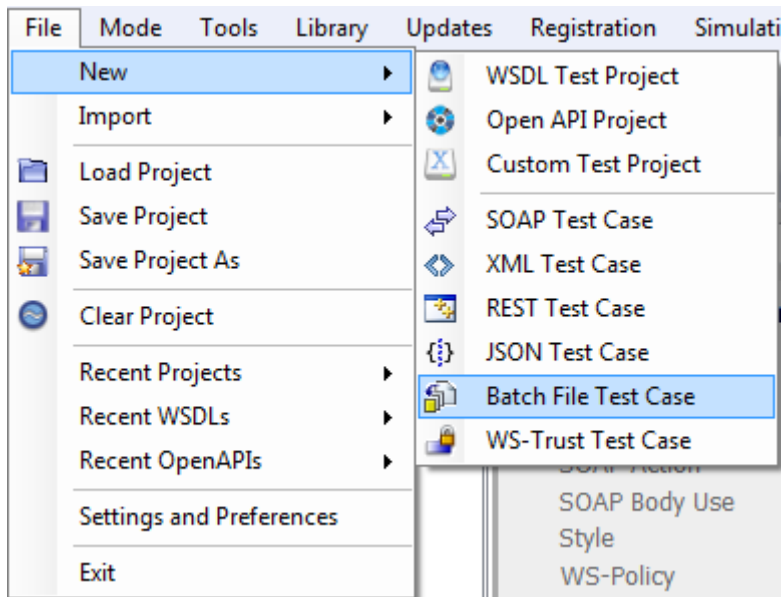


After defining the request data, the additional tabs on the request panel can be used for additional authentication and enrichment settings.

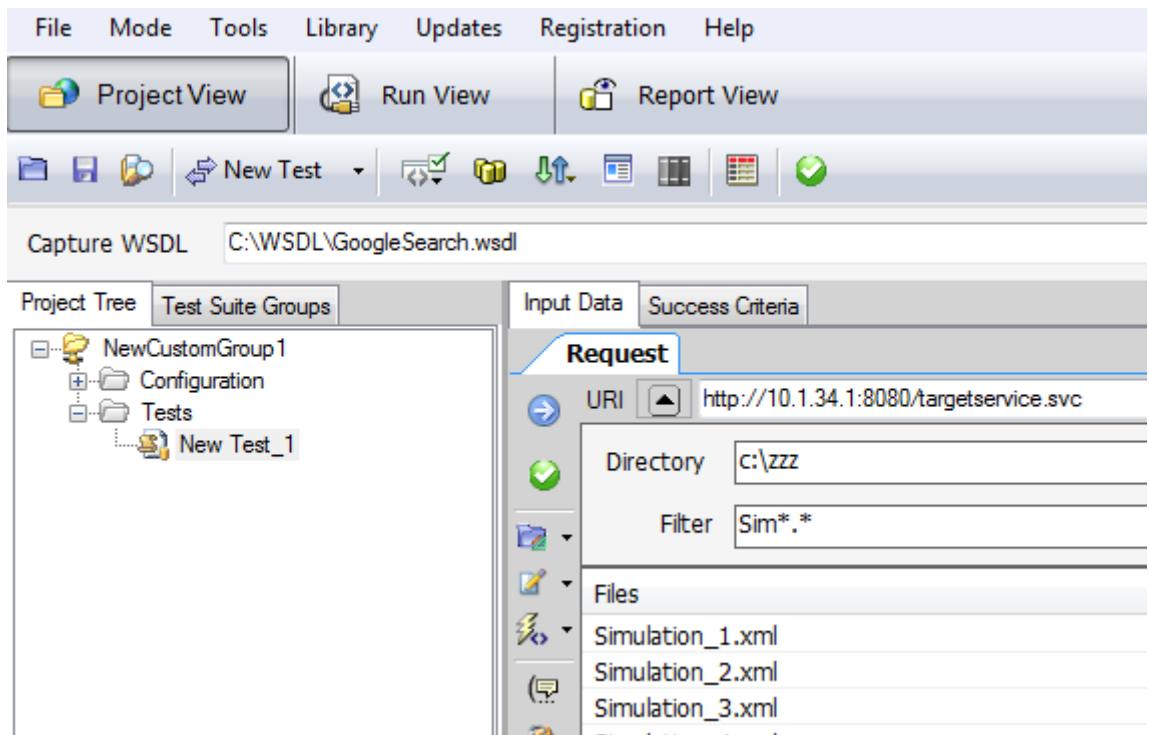
Batch File Test Case

A Batch File test case maps files in a directory to test iterations. Each file found in the specified directory will be iterated in the order shown on the dialog and the contents of each file will be extracted and used as the request data.

This test case type allows you to test using any type of input files, and still take advantage of the authentication and enrichment framework to apply to each files contents when the test is run.

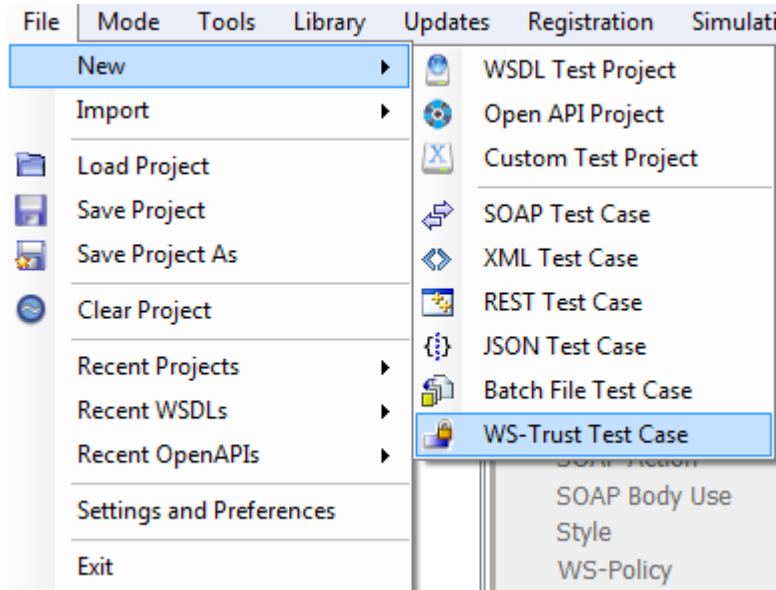


For batch file test cases, selecting the test case node in the project tree will display the Batch Files setting tab on the Request panel for configuring the batch file location and file filter information. A test case iteration will be created for each file referenced. The number of test iterations will appear in parenthesis on the Batch Files tab.

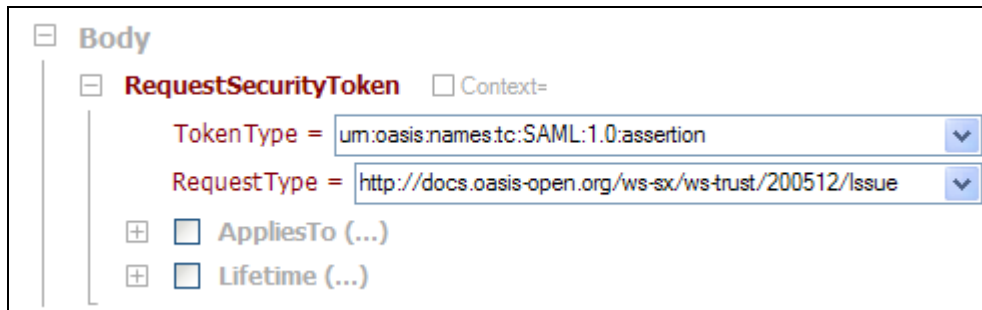


WS-Trust Test Case

A WS-Trust test case is used to query an STS service for a WS-Trust security token. You can create a WS-Trust test case by going to the File menu and first creating a Custom Test Group, then creating a WS-Trust Test case.



The resulting test case will provide the settings for requesting a WS-Trust security token from an STS server. The standard provisions of protocol (HTTP Basic Auth, SSL X509) and document (WS-Token) authentication can be used to authenticate to the STS service and retrieve the security token for subsequent use.

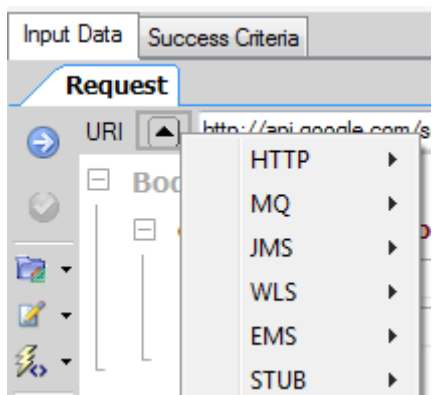


TEST CASE REQUEST

For each type of test case there are options to provide the input data, protocol authentication criteria, attachments, dynamic tasks, and scripting extensibility.

Test Case Protocols

For each type of test case, you can choose among HTTP, HTTPS, FTP, SFTP, IBM MQ, Tibco EMS, Weblogic JMS (WLS), and JMS as the protocol to send the messages on. For HTTP, HTTPS, and FTP, and SFTP the URI can be specified directly in the URI field. For MQ, EMS, WLS, or JMS protocols, the policy under the Message Providers under the Configuration area can be selected or entered directly.



The format for MQ policies is the prefix "MQ::" followed by the policy name. The format for JMS policies will show the prefix "JMS::" and then add the policy name, and so on for the other types of message provider policies. For HTTP or HTTPS policies, simply type the URI directly into the field, or select from previously entered entries. The selector to the left of the URI field allow you to choose from the configured message provider policies for the current project.

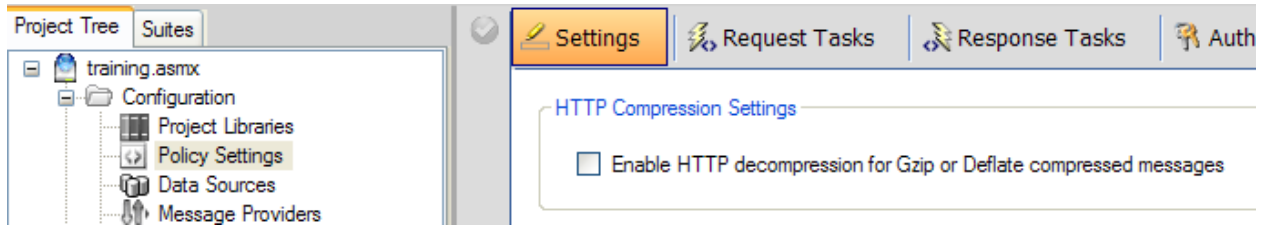
For FTP and SFTP policy types, when **FTP://** or **SFTP://** is detected in the URI, a field will appear in the response area for designating the FTP (or SFTP) response location if the test case is intended on reading data back from the server. Leaving this field blank will result in a one-way test case where the data is simply posted to the target FTP/SFTP server. The authentication credentials for both FTP and SFTP protocols can be set from within the Authentication tab.

STUB is used to indicate that no request is to be sent to the back-end server, but rather treat the test case as a STUB and use the saved response associated with the case case as the actual response.

For more information about configuring message provider policies, please refer to the Message Providers section of the help.

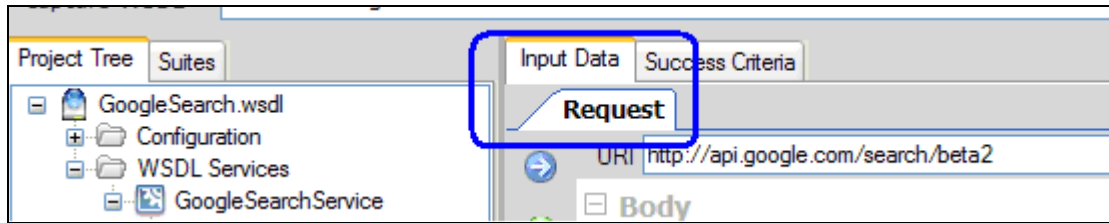
HTTP Decompression Support

If you are running tests against an endpoint which support HTTP compression of either GZip or Deflate, you can enable automatic decompression of this from within the Policy Settings tab under the Configuration area.



Test Case Request Definition

The Request tab provides access to the test message authoring features.

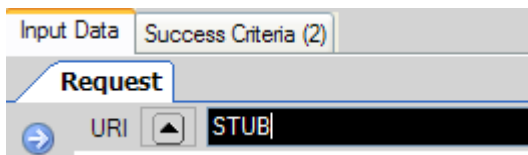


Request URI

This is the API, service, or application endpoint to send the data to. If the API is unavailable, you can choose to STUB a test and use the current response value of the test case (see STUB Tests below). The Request URI can also be dynamically parameterized using a test variable, for more information about variables in the request URI, go to the [Request URI Variables](#) section.

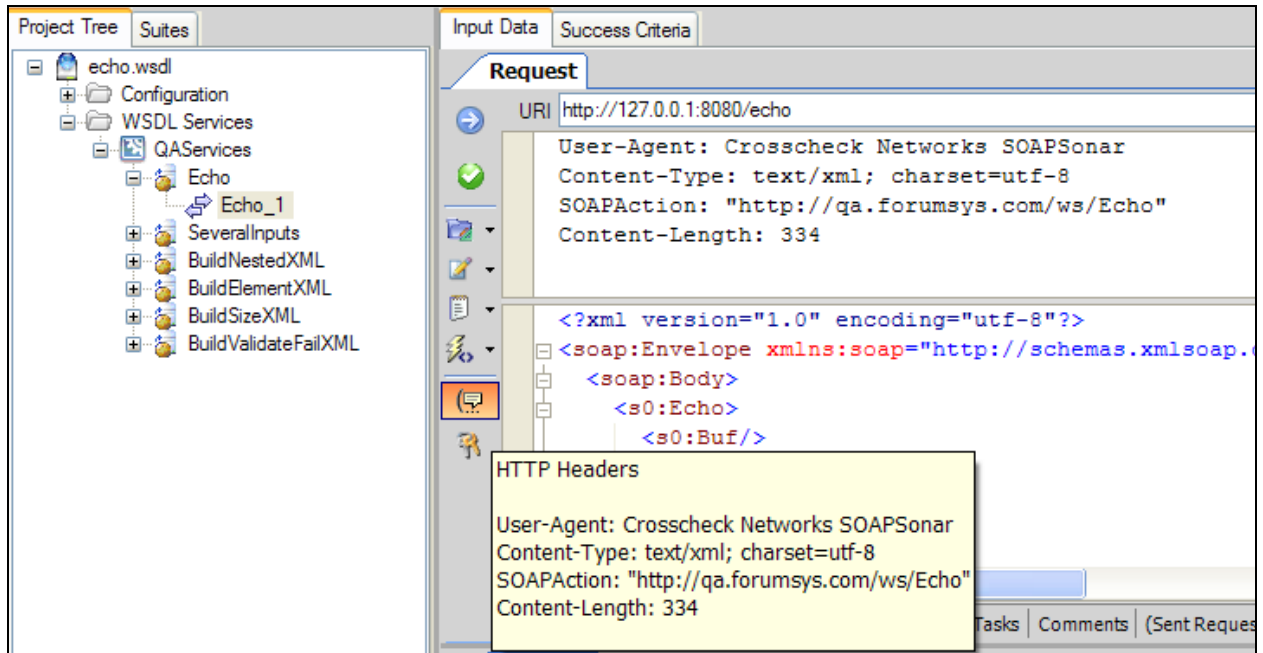
STUB Tests

There may be situations where the service is unavailable but the test is important to run for sequence testing. The test can be configured to not send the request to the back-end server but instead simply use the stored response message of the test. To configure STUB tests that use the current response stored with the test and not send a message to the back-end service, simply enter "STUB" in the URI field of the test.



HTTP Headers

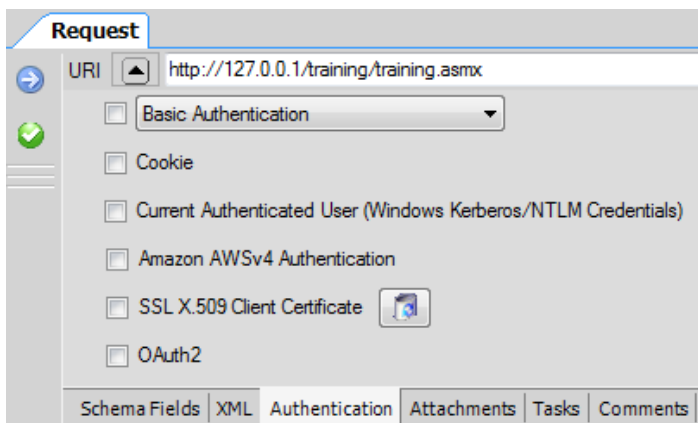
If you want to add any custom HTTP headers, click on the HTTP headers button on the menu bar next to the request and the screen will split to provide access to define the HTTP Headers. If you are working on the REST Test Case type, the HTTP Headers dialog will show by default.



Authentication Credentials

To define authentication credentials required by the API, the Authentication tab provides the various options for credentials on the HTTP protocol request (note, additional message-based authentication credentials are available under the Tasks tab under the “Identity Tokens” menu).

- SOAPSonar supports various types of authentication credentials. One the authentication tab, the credential types include:
- HTTP Basic Authentication,
- HTTP Kerberos
- HTTP Digest Authentication
- Cookie Authentication
- Kerberos/NTLM Authentication
- Amazon AWSv4 Authentication
- SSL X.509 PKI Authentication
- OAuth2 client_credentials and password grant-type Authentication



HTTP Basic Authentication

Choose the checkbox in front of the HTTP protocol authentication selection and select “Basic Authentication”. Enter the username, password, and optionally the domain

HTTP Digest Authentication

Choose the checkbox in front of the HTTP protocol authentication selection and select “Digest Authentication”. Enter the username, password, and optionally the domain

HTTP Kerberos Authentication

Choose the checkbox in front of the HTTP protocol authentication selection and select “Kerberos Authentication”. Enter the username, password, and optionally the domain

Cookie Authentication

Choose the checkbox in front of the Cookie authentication selection which will enable set-cookie header responses to cache cookies for subsequent tests.

Kerberos / NTLM Authentication

Choose the checkbox in front of the Current Authenticated User to use Kerberos or NTLM

Amazon AWSv4 Authentication

Choose the checkbox in front of the Amazon AWSv4 authentication. This will require an AWS Access Key, and AWS Secret Key, the AWS region name, and the AWS service name.

SSL X.509 Authentication

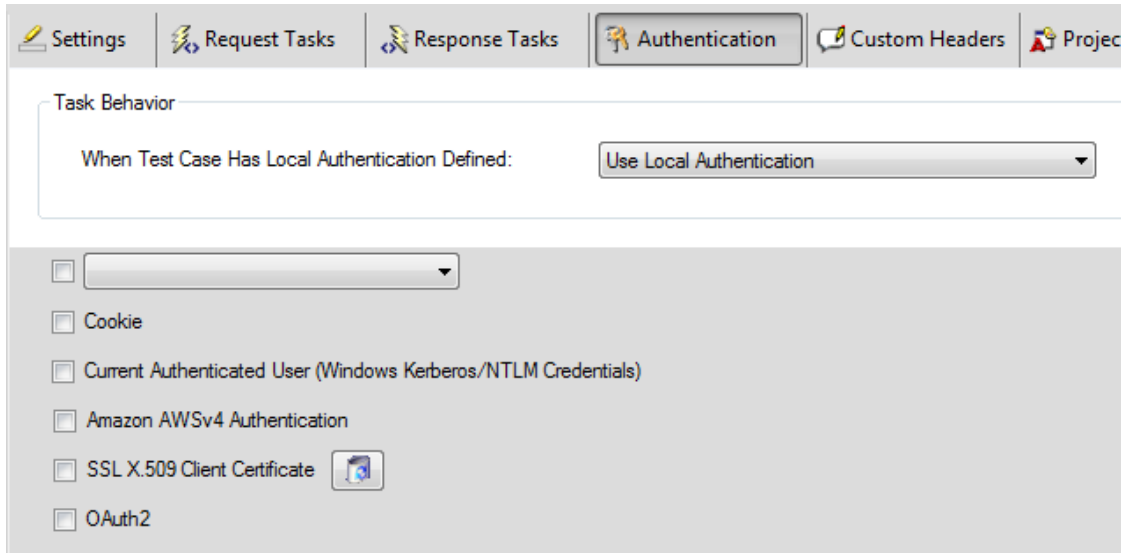
Choose the checkbox in front of the SSL x.509 authentication and select either a certificate alias, or an actual certificate.

OAuth2 Authentication

Choose the checkbox in front of the OAuth2 authentication. OAuth2 grant types supported include client_credentials and password grant types. ClientId and Client secret are required for client_credentials grant type. Those properties and the username/password is required for the password grant type.

Global Authentication Credentials

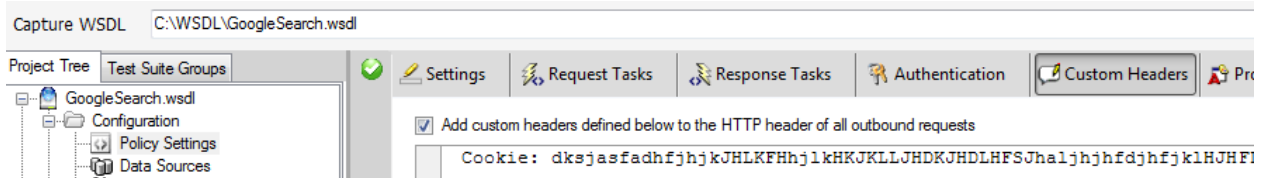
You can optionally choose to define the authentication settings globally for the Test Project to have the settings apply to each test in the group. To access the global authentication, go to the Policy Settings node, then click on the Authentication tab.



The Task Behavior section provides override options on this screen to dictate how to apply the credentials when the Test Case has locally defined authentication.

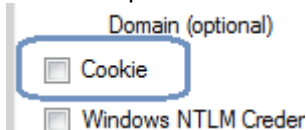
Global Cookies and HTTP Custom Headers

You can optionally choose to define custom HTTP headers, including cookies, which will be added to the test case HTTP header when the test is run. To access the global HTTP Header settings, go to the Policy Settings node and click on the Custom Headers tab. Click on the “Custom HTTP Headers” checkbox and enter the global header(s) one per line.



Automatic Cookie Handling

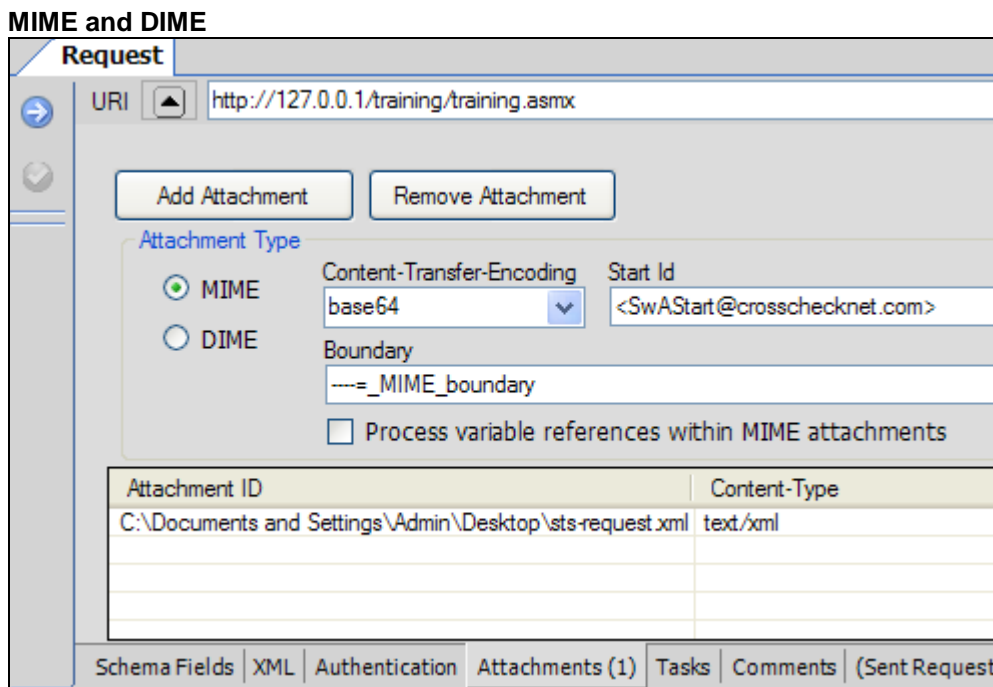
Automatic cookie handling can be enabled from the Global Authentication Settings tab, or from the Authentication tab within the test case. Click on the checkbox for “Cookies” and if SET-COOKIE responses are detected, the cookie will be automatically stored and used in subsequent requests per the URI and Domain specified in the original SET-COOKIE response.



MTOM, MIME, and DIME Attachments

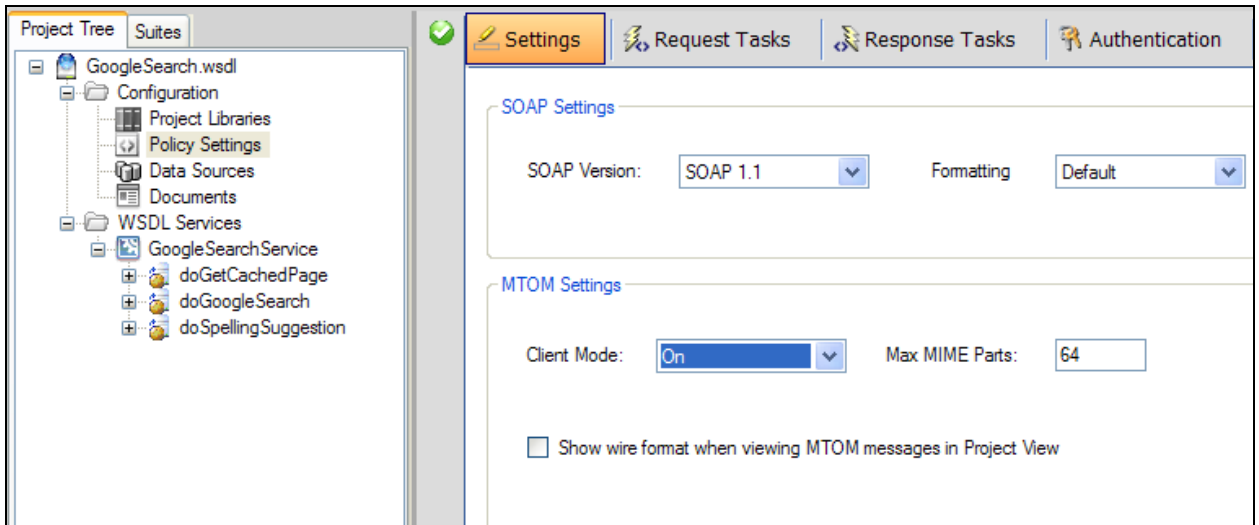
Your back-end service may be enabled for SOAP attachments and require a SwA MIME, DIME, or MTOM formatted input message. SOAPSonar support all three of these specifications for transmitting data. The MIME and DIME specifications provide for the ability to browse to the locations of the files to attach to the message.

For MIME or DIME, you can configure each test case with the specified attachments by navigating to the Attachments tab and selecting the file(s) to be attached to the request. If you choose MIME mode, you also have the option to enabled processing variable dependencies within the MIME attachments. This means that you can instrument the attachments with SOAPSonar variable references and the variables will be resolved dynamically within the attachments when the tests are run.



MTOM

MTOM stands for “Message Transmission Optimization Mechanism”. This means that the SOAP message may be optimized in transit to the target machine. Most applications will not provide the ability to see the representative MTOM messages since the serialization and conversion is done when the messages are send and received from the network. SOAPSonar provide the ability to generate and consume MTOM messages, and further allows you to diagnose and view the raw MTOM format. MTOM support can be enabled from the Policy settings node under the Configuration folder. When sending the requests from Project View, the Sent Request tab will show the MTOM message in it’s raw format. To see Raw MTOM responses in Project View, check the option shown and SOAPSonar will show the raw MTOM response, rather than the de-serialized SOAP message after MTOM conversion



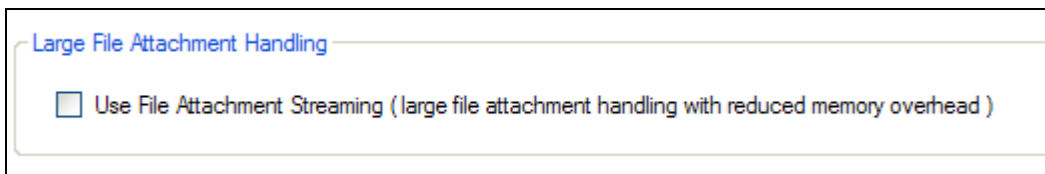
MTOM and MIME Large File Streaming

Enabled via a special component license, SOAPSonar supports large file streaming of MTOM or MIME based attachments. Large file streaming allows files up to 4GB in size to be sent or consumed via MTOM or MIME content types.

Large file streaming will take the outbound message, or inbound message, and process the data in chunks such that the analysis portion of the transaction takes place, but the message itself is not stored. Rather, only the statistics of the message is stored.

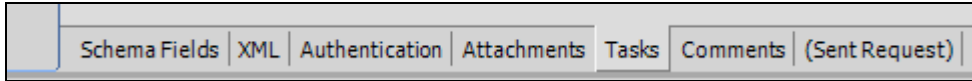
Traditionally it would require client or server machines with significant memory resources to be able to send or consume messages of large sizes. With streaming technology, the same file transfers can be tested with dramatically lower resource overhead. With SOAPSonar streaming technology, a 4GB file can be transferred over the network and only consume a few MB of memory.

The large file handling support can be found at the global policy level under Configuration->Policy Settings, or on individual test cases on the Attachments tab. The option when checked will enable the streaming processing module to handling inbound and outbound transactions of type MTOM or MIME.



Test Case Request Tasks

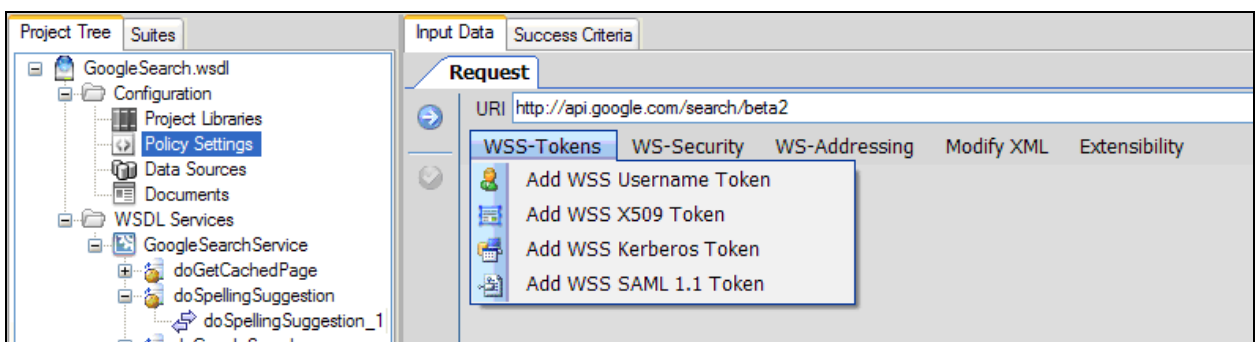
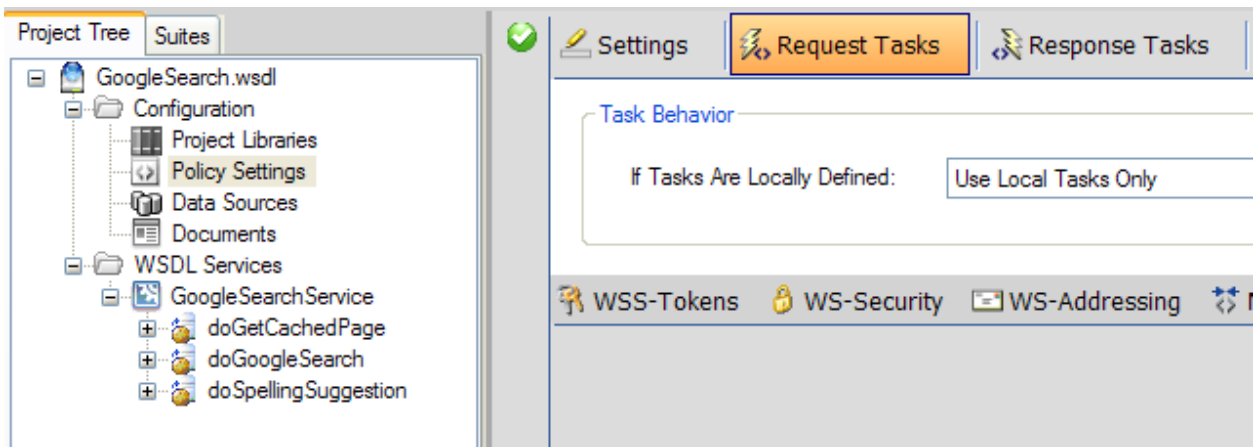
The tasks tab provides request document enrichment tasks that will modify the base test case data each time the test is run. Task enrichment functions include WS-Header authentication, WS-Security, WS-Addressing, XSLT, string replacement, external DLL plug-in invocation, and inline scripting.



Each time the test case is run, the corresponding task list is run against the request dynamically and thus any applicable timestamp values, nonce attributes, signatures, and other replay preventative measures required by the WS-* specifications are met.

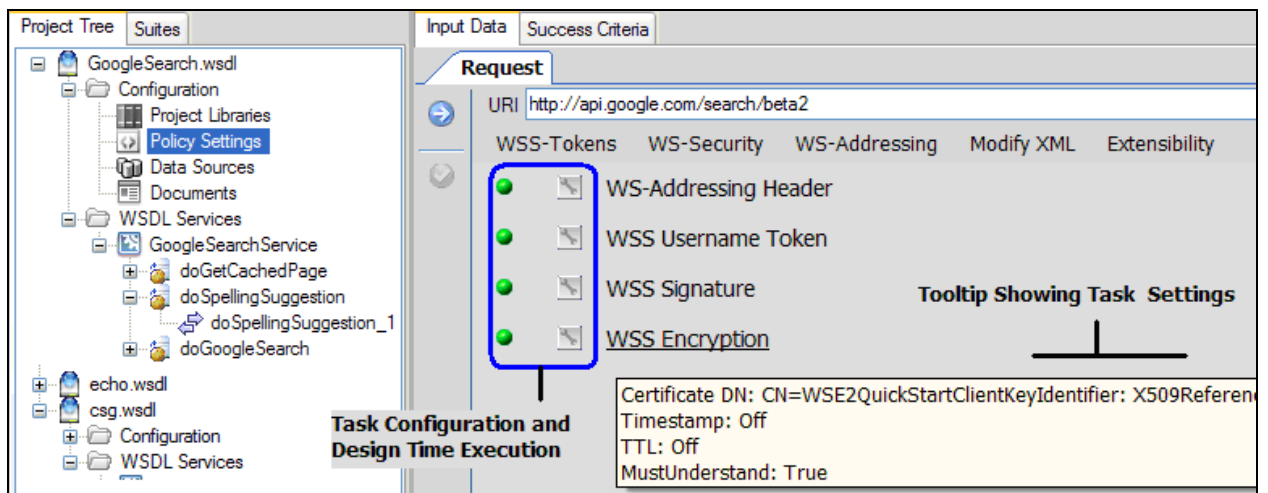
Global Request Tasks

Tasks can be configured globally for each WSDL project group, or individually for each test case, or both. The global task configuration page is found on the Policy Settings node under the Configuration folder, while the local test case dynamic tasks can be seen from the Tasks tab of the Test Case Request area.



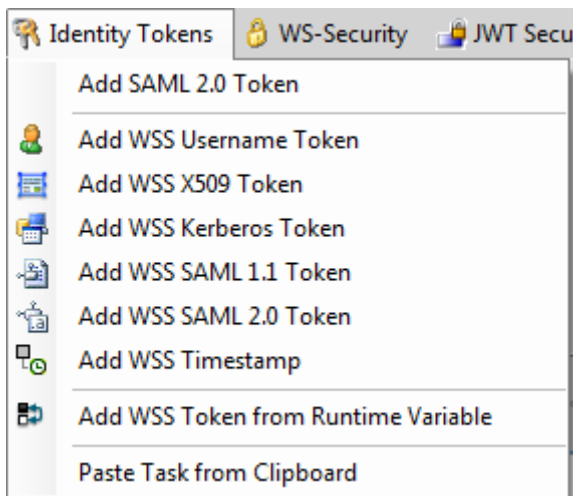
When building lists of tasks, you can configure each one individually by clicking on the left icon and choosing **Configure**. You can view the resulting document with the result of invoking all tasks up to and including the currently selected one by clicking the left icon and selecting view. To change the order of tasks, click on the left icon and choose move up or move down. To remove a task, click on the left icon

and choose Delete. If you hover over the labels for the tasks, you will see the settings summary for each displayed in the tooltip text.



Request Task: Identity Tokens

This menu item provides access to the tasks for adding tokens to the message. This includes standard SAML 2.0 and WS-Security Token options, including WSS SAML 1.1 and WSS SAML 2.0.



To add a token identification task to the test case document, click on the Identity Tokens menu and then choose the type of token task to create. Click on the task name to open the task properties screen for configuring the task. Once a task is configured, you can view the result of applying the task by clicking on the wrench icon and choosing the View option. This will result in invoking each task in the list up to the task associated with the wrench icon to preview the processing results that will occur each time the test case is executed. A document viewer dialog is presented to review the resulting processed document.

Request Task: Identity Tokens->Add SAML 2.0Token

Use this option to generate a standard (non WS-Security based) SAML 2.0 Token

Request Task: Identity Tokens->Add WSS Username Token

Use this option to generate a WSS-Username Token

Request Task: Identity Tokens->Add WSS X509 Token

Use this option to generate a WSS-X509 Token

Request Task: Identity Tokens->Add WSS Kerberos Token

Use this option to generate a WSS-Kerberos Token

Request Task: Identity Tokens->Add WSS SAML 1.1 Token

Use this option to generate a WSS-SAML 1.1 Token

Request Task: Identity Tokens->Add WSS SAML 2.0 Token

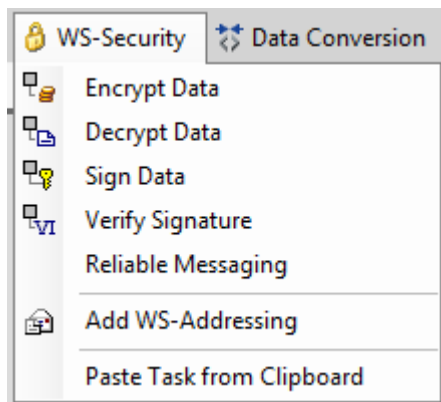
Use this option to generate a WSS-SAML 2.0 Token

Request Task: Identity Tokens->Add WSS Token from Runtime Variable

You can also choose to extract a token from another test via Runtime Variable reference. To do this, go to the test where the token is to be referenced from and create a Node capture Runtime Variable targeting the identity token. Then, go back to the current test case and select the WSS Token from Runtime Variable task. Within that task, select the Runtime Variable identity token.

Request Task: WS-Security and WS-Addressing

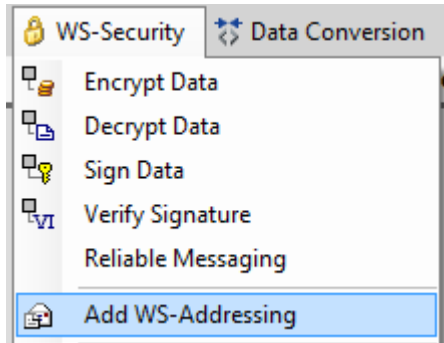
The WS-Security Menu is used to define tasks for Signature and Encryption settings. The security tasks support the WS-Security Standards for generating compliant WSS-2004 Signatures and WSS 2004 Encryption.



To add a Signature or Encryption task, click on the WS-Security menu. The Signature and Encryption tasks can be applied to the header, body, or any selected node. Click on the task name to open the task properties screen for configuring the task. Once a task is configured, you can view the result of applying the task by clicking on the wrench icon and choosing the View option. This will result in invoking each task in the list up to the task associated with the wrench icon to preview the processing results that will

occur each time the test case is executed. A document viewer dialog is presented to review the resulting processed document.

To add a WS-Addressing header to the request, choose the Add WS-Addressing option.



To add a WS-Addressing task, click on the WS-Addressing menu and then choose the Add WS-Addressing option. Click on the task name to open the task properties screen for configuring the task. Once a task is configured, you can view the result of applying the task by clicking on the wrench icon and choosing the View option. This will result in invoking each task in the list up to the task associated with the wrench icon to preview the processing results that will occur each time the test case is executed.

Request Task: JWT Security

This menu item provides access to the JWT (Java Web Tokens) Security functions



Request Task: JWT Security->JWT Sign/Encrypt

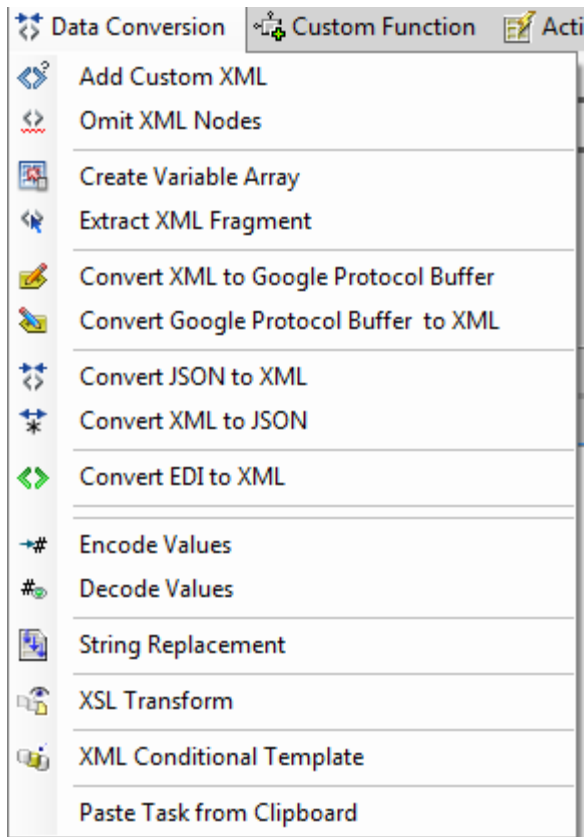
Use this option to sign and/or encrypt via JWT.

Request Task: JWT Security->JWT Decrypt/Verify

Use this option to decrypt and/or verify signature via JWT.

Request Task: Data Conversion

Enable the processing of the request or response via various data conversion tasks, including XSLT transformation, direct string replacement, and adding custom XML fragments to the request.



Click on the task name to open the task properties screen for configuring the task. Once a task is configured, you can view the result of applying the task by clicking on the wrench icon and choosing the View option. This will result in invoking each task in the list up to the task associated with the wrench icon to preview the processing results that will occur each time the test case is executed.

Request Task: Data Conversion-> Add Custom XML

Use this option to inject Custom XML fragments into the message.

Request Task: Data Conversion-> Omit XML Nodes

Use this option to define a NULL value that when detected will remove the entire element, and optionally the parent element when the remove parent checkbox is enabled.

Request Task: Data Conversion-> Create Variable Array

Use this option to define an XML fragment array using runtime variables.

Request Task: Data Conversion-> Extract XML Fragment

Use this option to extract values from an XML or JSON document either inline for the current request, or targeting a file where the XML/JSON information resides.

Request Task: Data Conversion-> Convert XML to Google Protocol Buffer

Use this option to translate the XML document to Google Protocol Buffer format. This option requires the Google Protocol buffer libraries to be installed.

Request Task: Data Conversion-> Convert Google Protocol Buffer to XML

Use this option to translate the Google Protocol Buffer format to XML format. This option requires the Google Protocol buffer libraries to be installed on the instance.

Request Task: Data Conversion-> Convert JSON to XML

Use this option to convert from JSON to XML

Request Task: Data Conversion-> Convert XML to JSON

Use this option to convert from XML to JSON

Request Task: Data Conversion-> Encode Values

Use this option to encode selected XML or JSON values, or the entire document. Supported encodings include URL Encode, Base64 Encode, UTF8 Encode, MD5SUM, SHA1, SHA256, and JWT (Java Web Token)

Request Task: Data Conversion-> String Replacement

Use this option to perform find/replace string replacement.

Request Task: Data Conversion->XSL Transform

Use this option to perform XSLT transformations.

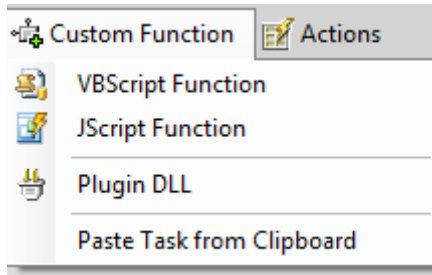
Request Task: Data Conversion->XML Conditional Template

Use this option to perform conditional replacement of XML structured content based on criteria that determines which template to substitute. To build new conditions, click on the “+” button to create a new custom XML template rule. The condition rule can then be created. Both the source and target of the condition can be variable values. The custom data template is the data that will be used to store into a variable, or to insert directly into the document at the specified location.

Request Task: Custom Functions

SOAPSonar supports scripting and custom DLL extensibility with a request event that can invoke DLL functions. This task allows you to build your own DLL function to be invoked dynamically for each test case iteration being invoked. The DLL function can modify the request and HTTP header in any manner and the results of the modifications will be submitted as the test case request data.

The ISOAPSonarPlugin API interface for the DLL plug-in can be found in the plugins/ subdirectory under the installation directory. .

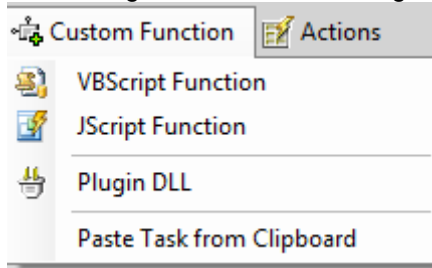


Inline scripting is available in VBScript or JavaScript format. Inline scripting provides full ability to alter the request and request headers within the scripting language of choice.

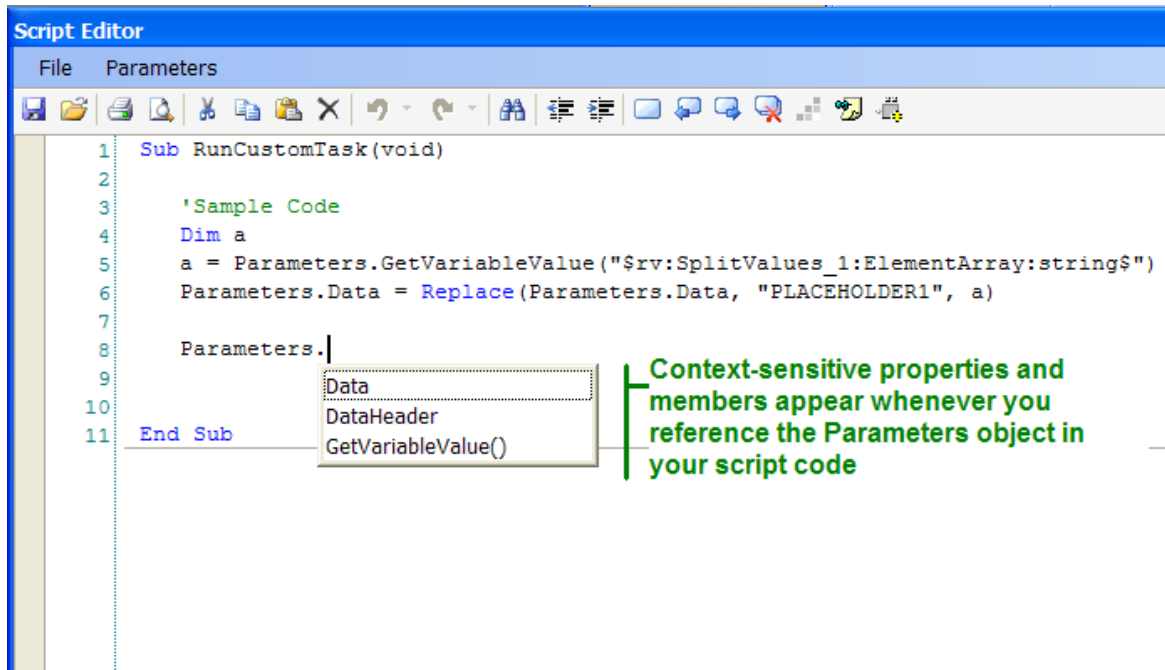
Click on the task name to open the task properties screen for configuring the task. Once a task is configured, you can view the result of applying the task by clicking on the wrench icon and choosing the View option. This will result in invoking each task in the list up to the task associated with the wrench icon to preview the processing results that will occur each time the test case is executed. An XML viewer dialog is presented to review the resulting processed document.

Request Task: Custom Functions->VBScript and JScript

This option allows you to create your own custom script function to process data or manipulate the environment during the test run. The languages supported are VBScript and JScript



SOAPSonar provides context-sensitive script editing which allows you to see highlighting as well as use the Global class object "Parameters" to resolve SOAPSonar variables, or view the current data for the transaction occurring at runtime. When the script editor is launched, the menu item Parameters shows the global class properties and methods that are accessible.



The **Parameters.Data** property provides access to either the request data or the response data of the current transaction depending on whether the task is a Request Task or a Response Task. If you set this value in your code, it will replace the Request/Response in SOAPSonar.

The **Parameters.DataHeader** property provides access to either the request header or the response header of the current transaction depending on whether the task is a Request Task or a Response Task. If you set this value in your code, it will replace the RequestHeader/ResponseHeader in SOAPSonar.

The **Parameters.GetVariableValue(VarName)** member function allows you to get the value for a SOAPSonar variable name

The **Parameters.SetVariableValue(VarName,NewValue)** member function allows you to set the value for a SOAPSonar variable directly within the script function.

Memory Table variables are fully integrated into the VB Script environment allowing you to read and write Memory Table variables as well as enable direct manipulation of the collection objects of the memory table values.

Some example code showing memory table function in VB Script is below

```

Sub RunCustomTask(void)

' Get the index of the memory table from an ADS Variable (shows the optional
' ability to specify memory table variables, or portions of memory table
' variables dynamically from a data source such as CSV or Excel

Dim SIndex
SIndex = Parameters.GetVariableValue("$ads:STEP:012345$")

'Set up the Memory Table reference

```

```

Dim MemoryTableVar
MemoryTableVar = "test[" & SIndex & "].AAA"

'Add some values to the "AAA" member variable

MemoryTable.SetValue MemoryTableVar , "value1"
MemoryTable.AppendValue MemoryTableVar , "value2"
MemoryTable.AppendValue MemoryTableVar , "jason3"

'Enumerate the value collection of a memory table variable
'and show the values in a popup Message Box
Set memTableValues = MemoryTable.GetValueCollection("test[" & SIndex & "].AAA")

For Each strItem in memTableValues
    MsgBox(strItem)
Next

'Replace the string PLACEHOLDER1 with the Memory Table string value
Parameters.Data = Replace(Parameters.Data, "PLACEHOLDER1",
MemoryTable.GetValue(MemoryTableVar))

End Sub

```

Once a task is configured, you can view the result of applying the task by clicking on the wrench icon and choosing the View option. This will result in invoking each task in the list up to the task associated with the wrench icon to preview the processing results that will occur each time the test case is executed. An XML viewer dialog is presented to review the resulting processed document.

Request Task: Actions->Database Query

Use this task to run a database query using ODBC, or native SQL Server or Oracle database drivers. This action task fires prior to running the request. With this feature database tables can be created, deleted, or modified based on the SQL query before the test data is sent.

Database query response information can also be preserved into several different types of variables. The most flexible is the Memory Table variable which can be used to preserve all columns for the entire query, or just select values can be stored to a runtime variable or a global variable.

To set the column values to the Memory Table, there is no need to have already defined the Memory Table member variables to match the columns. If the Memory Table target does not exist, it will be created dynamically. If the Memory Table member variable does not exist, then it will also be created dynamically.

For example, if a query returns the following rows:

INDEX	STEP	TEST
1	12	Sample'

And the selection to SetValue or AppendValue to the Memory Table is active, then the Memory Table variable will automatically get created or extended such that the member variables contain the query column values return.

Before query:

test[ABC]

After query:

test[ABC].INDEX

test[ABC].STEP

test[ABC].TEST

Request Task: Actions->File Manipulation

Use this option to create, append, or delete a file. This action task fires prior to running the request. Variable substitution is enabled such that data to be written to a new file, or appended to an existing file can be static data, or dynamic data based on the current values for the runtime test.

Request Task: Actions->Add Test Delay

To add a specific wait duration before the test is run, choose the Action->Add Test Delay and enter the delay value in milliseconds.

Request Task: Actions->Update Global Variable

Global variables can be updated while running a test to have future tests running in the current test suite use the update value(s) of the variables. This includes the ability to map global variable values to other test variables such as runtime variables, ADS variables, etc.

Request Task: Actions->Update Memory Table

Memory Tables provide the means to store complex data across test case calls and can be used anywhere that variables can be referenced to aggregate, preserve, or substitute information. In the test cases. The Update Memory Table task provides the means to do the following Memory Table operations:

SetValue

Set the Memory Table member variable to the specified value (which can also be a variable type). If this value was previously appended to (i.e. contains multiple entries), a SetValue call will clear the variable and replace the value with the specified value.

AppendValue

Append to the Memory Table member variable to the specified value (which can also be a variable type). Memory Table member variables can always be represented as multiple value entries (i.e. collections/arrays) simply by using the AppendValue method for the table.

GetValue

Gets the Memory Table member variable value. If this value is single value, then the call will return the single entry. If the variable contains multiple entries, then this call will result in a comma-delimited list of values. If the Memory Table variable was defined with an XML template, then this call will return with each value surrounded by the designated XML template

ReplaceValue

Replaces all instances of the find string with the replace string in the Memory Table variable.

ClearValue

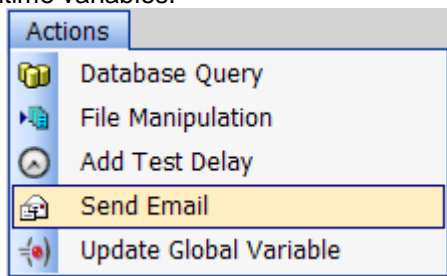
Clears all values associated with a MemoryTable member variable

ClearTable

Clears all member variables in the specified Memory Table

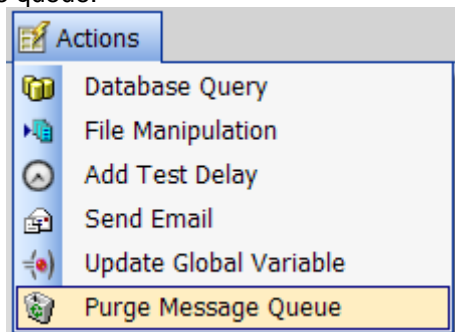
Request Task: Actions->Send Email

To send an email before a test request is sent, you can create a Send Email task action from the Actions menu. Settings include the SMTP server with optional authentication, SSL, and the to, from, title, and body of the email. As is the case with all tasks, variables are enabled to parameterize the email contents with runtime variables.



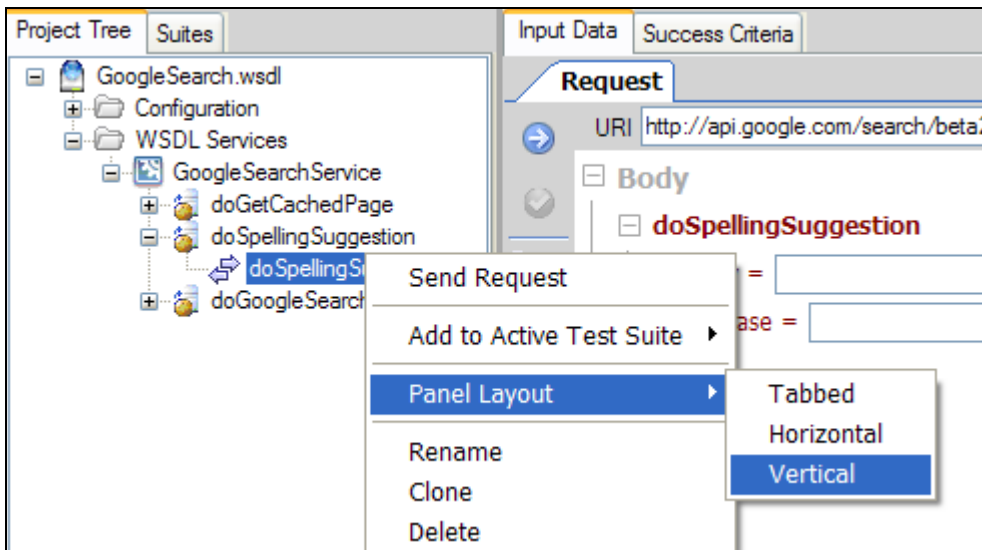
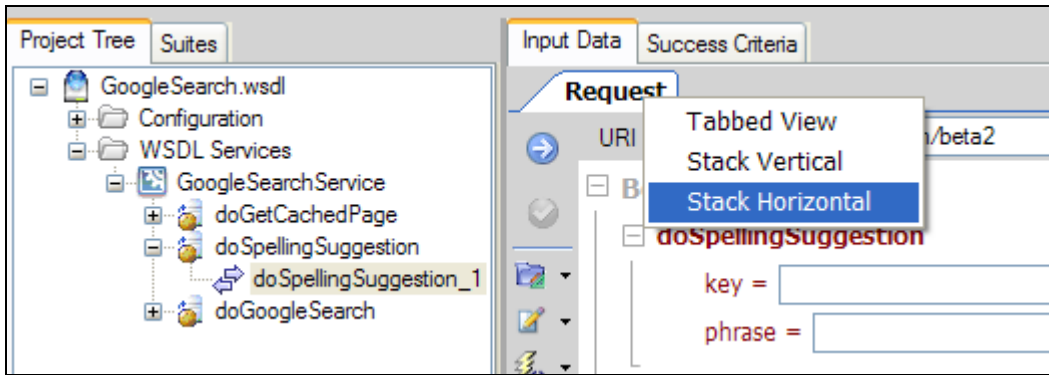
Request Task: Actions->Purge Message Queue

For JMS, Weblogic JMS, IBM MQ, or Tibco EMS Queues, the Purge Message Queue task can be used to clear the queue. This task will iterate through all messages on the target Queue and remove them from the queue.



Test Case Authoring – Change Request and Response GUI Layout

You can choose the layout for the Request and Response areas to be stacked vertically, horizontally, or in tabbed format. To alter the layout, right-click on the **Request** or **Response** tab, or right-click on the test case node itself and choose the Panel Layout option.

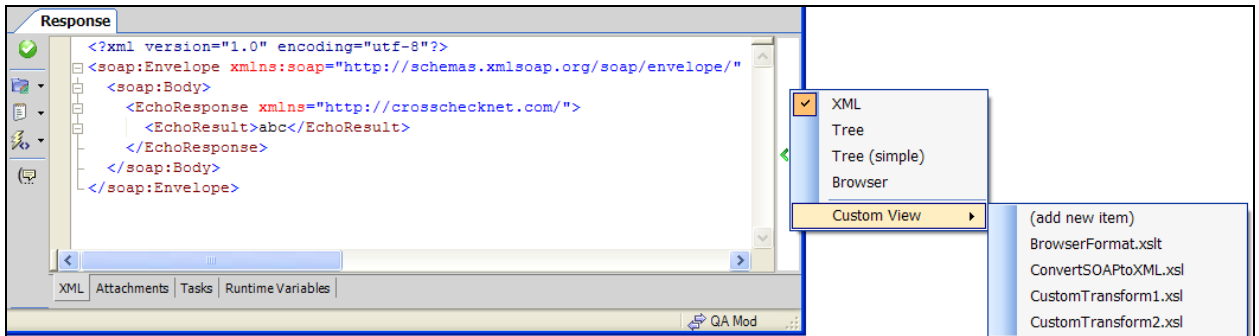


TEST CASE RESPONSE

The response area in project view allows you to see the response from invoking the back-end service using the latest settings for the test case input.

Response Data View Formats

There are several different ways to view the response data. You can view raw xml, tree view, simple tree view, browser, or define and load your own custom XSLT transform files to provide customized views and formats of data. To access the view perspectives, click on the icon to the right of the response display. The view perspectives are explained in more detail below.



- XML:** Used to edit and view raw response
- Tree:** Used to view graphical tree of XML response (read only)
- Tree (simple):** Omits Namespaces and Attributes. Shows only elements with values (read only)
- Browser:** Renders XML response similar to Internet Explorer (read only)
- Custom View** Allows selection of custom XSLT to format view of response data as HTML

For the Custom View option, selecting “add new item” will allow browsing for XSLT files to add to the list of available custom view options. These files can also be copied directly to the lib/xslt directory under the installation directory which is scanned at start-up.

Response Tasks

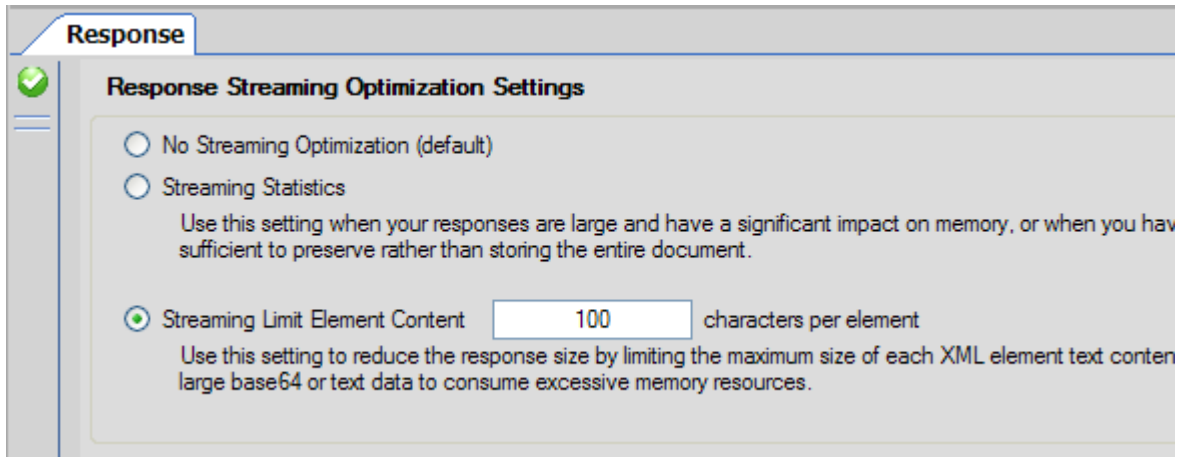
Response tasks operate on the API response data that comes back from the target endpoint. If you are performing Success Criteria analysis of the response, or creating a Runtime Variable reference to response data, you may first need to manipulate the content of structure of the response dynamically before evaluating the Success Criteria or capturing the response value associated with the Runtime Variable. This can be accomplished through the definition of Response tasks.

Response tasks will be applied dynamically to the response before Success Criteria evaluation is performed and before the dynamic Runtime Variable data capture is done. To create response tasks, navigate to the Tasks tab in the Response area.

These tasks are the same set provided for the Request task event, with the difference simply that the target of the tasks is the API response and the tasks trigger on the response event. For more information about each individual task, refer to the task definitions under Request Tasks.

Response Large File Streaming

SOAPSonar provides optimization settings that allow you to reduce the memory overhead associated with processing large file responses to your test cases. The settings can be found on the test case response tab under “Optimize”.



The options for optimization include:

No Streaming Optimization (default)

No streaming optimization will occur. The entire response document will be read and processed. This is the default behavior and setting.

Streaming Statistics

This setting will process the response information from the network as a stream and only store the statistics of what was processed, including the total bytes read, the total number of XML nodes detected, and the maximum level of node depth for the XML. This setting is useful when you don't actually need to store the response result data directly, but instead can validate the test scenario based on the stats rather than the data itself.

Streaming Limit Element Content

This setting will process the response information from the network as a stream and for each element reduce the text content to be limited to the specified limited length. As long as the actual response received is a valid XML document, the resulting optimized XML document will contain all of the original tags, but the data within the tags will be limited to the specified number of characters.

For example:

No Optimization

```
<response>
  <data>ABCDEFGHIJKLMNOPQRSTUVWXYZ</data>
</response>
```

Streaming Optimization Set to 10 characters Per Node

```
<response>
  <data>ABCDEFGHIJ</data>
</response>
```

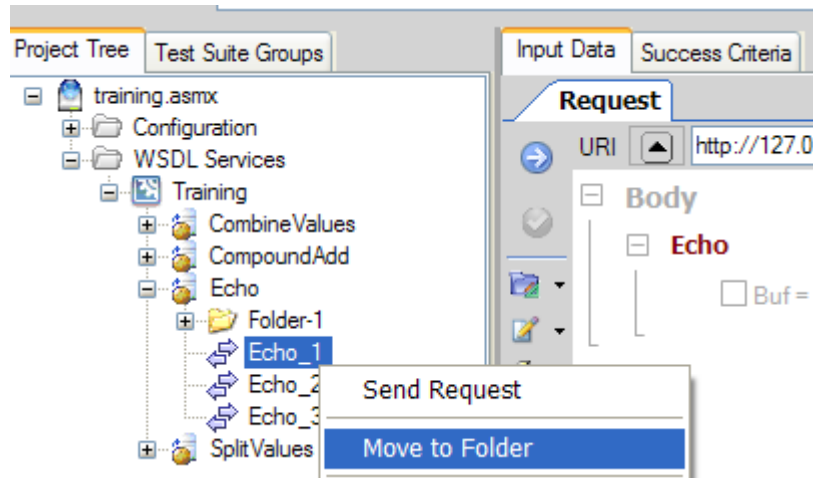
ORGANIZING TESTS

Tests in project view represent the base items that can be combined in difference sequences in Run View within Test Suites and Test Groups. As test projects grow larger, it is often useful to arrange tests

logically within folders so that the base test cases can be easily navigated as well as the Test Suites that will be run. There are 2 primary methods to organizing tests with the project. The first is to use Test Folders within Project View. The second is to allocate the tests to Test Suites, and then use the Test Groups tab within project view to isolate and only view the sequence.

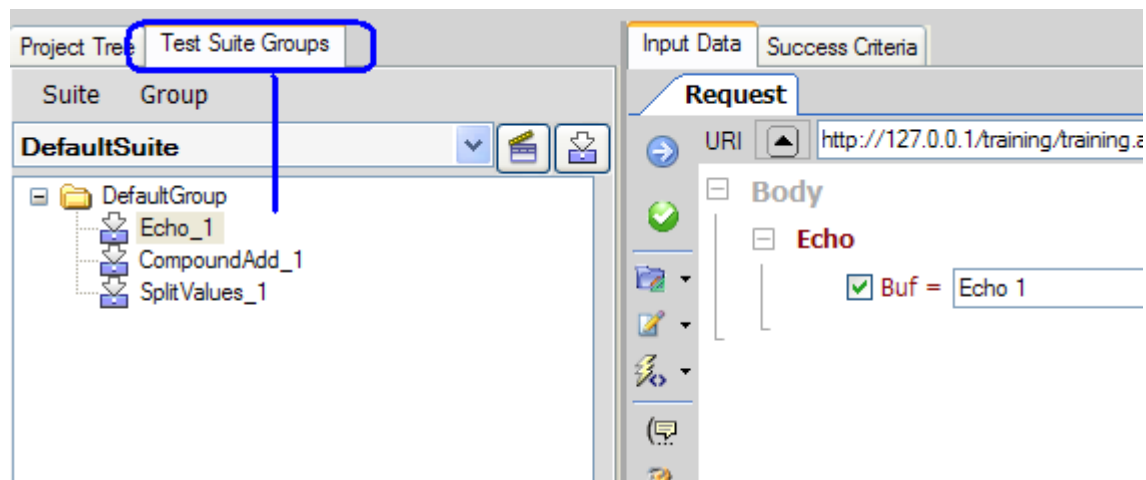
Project View Test Folders

In project view, the tests can be separated into folders for logical grouping and ease of use. To move a test case into a folder, simply right-click and Choose the “Move To Folder” option.



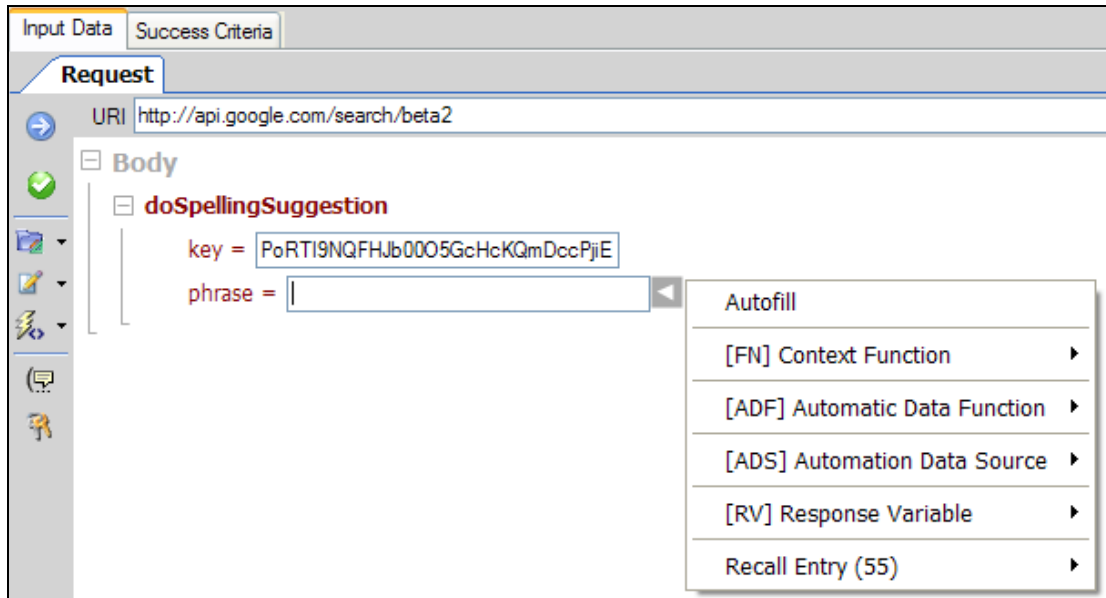
Test Suite Group Filtered View

In project view, you can work with the Test Suite Group flows directly to isolate the sequence and tests particular to a certain test suite. To use the Test Suite Group filtered view, simply click on the “Test Suite Groups” tab next to the Project Tree tab. The defined test suites are available for selection. New Tests can be allocated to the Suite groups from here, or within the Run View section.



TEST VARIABLES

Test cases can be dynamically parameterized through the use of special variable references. Each time the test is run the variable is replaced with the current value(s) that the function variable type maps to.



Variables can be used in the request anywhere within the body content, the header, and task configuration fields. For dynamic response evaluation and synchronized variable replacement with inputs, variables can also be used to parameterize success criteria values.

Variable references appear as strings bounded by "\$...\$" with individual formats specific to each variable function type. The subtopic in this section define each type of variable functions, which include:

[Context Function](#)

[Global Variables](#)

[Automatic Data Function](#)

[Automation Data Source](#)

[Runtime Variable](#)

[Memory Table Variables](#)

URI Substitution

Context Function Variables

Context Function variables are resolved for each test iteration. Context functions are useful for tasks such as tagging data with unique identifiers, populating date fields with the current date and time, generating unique IDs, and injecting static file contents into the XML document

A context function is simply a text string which is recognized and the designated function is ran whenever a test request is to be sent. Context function variables can be used within request data, headers, tasks, or success criteria. After the function runs, the return value is substituted in place of the referenced context function variable.

Context Function Variable Format

Variable references for Context Functions are denoted using the following syntax:

\$fn : Function (<optional parameters>) \$

Where:

- **Function** is the type of function replacement to be performed (see below)
- **Optional Parameters** are parameters that may be used by the function for additional options

Context Function: Now()

The Now() function will provide a current UTC time value. Each time the test is run the variable is replaced with the current UTC date and time. When no parameter is specific, the date and time are provided in ISO8601 format. The parameter value can also be set with a formatting string for designated date/time formats.

\$fn:Now()\$

Parameters:

() : Current date in ISO8601 format (*yyyy-MM-ddTHH:mm:ss.fffZ*)

(FormatString): Format for date. See [Appendix A](#) for supported data format values

Context Function: GUID()

The GUID() function generates a globally unique value. Each time the test is run the variable is replaced with a globally unique identifier. An optional parameter value can be supplied which is the length of the GUID string to use. Default is standard length GUID.

\$fn:Guid()\$

Parameters:

() – Standard length GUID

(Length) – GUID truncated to the value specified by the length parameter.

Context Function: FileContents()

The FileContents() function will retrieve the contents of the file at the specified full path location. Each time the test is run the variable is replaced with the contents of the referenced file. Parameters can be supplied to determine encoding options on the file contents. A full path filename is a required parameter.

\$fn:FileContents(<>,encode-xml-tags)\$

Parameters:

- (Filepath) – Full path to filename, encode any XML tags to > and <
- (Filepath,true) – Full path to filename, encode any XML tags to > and <
- (Filepath,false) – Full path to filename, no XML tag encoding

The Project Global policy settings define the default root directory for relative path references. This setting also applies to the FileContents() Context Functions.

Context Function: Random()

The Random() function will generate a new random value of type string, integer, or float. Parameters allow setting the data type, and the min and max value. For integer and float the min and max are used for the range of the generated random value. For string data type, the min and max parameters are used for the length of the string.

\$fn:Random(TYPE,MIN,MAX)\$

Parameters:

- (TYPE) – String, Integer, or Float
- (MIN) – Minimum value for the random values
- (MAX) – Maximum value for the random values

Context Function: B64()

The B64() function will base64 encode the target file. Each time the test is run the variable is replaced with the contents of the referenced file with the contents base 64 encoded. A full path filename is a required parameter.

\$fn:b64()\$

Parameters:

- (Filepath) – Full path to filename

The Project Global policy settings define the default root directory for relative path references. This setting also applies to the b64() Context Functions.

Context Function: MD5()

The MD5() function will generate an MD5 hash of the parameter contents. Each time the test is run the variable is replaced with the MD5 hash value applied to the contents of the string within the function parameter.

\$fn:MD5(HashString)\$

Parameters:

- (HashString) – String to compute hash

Context Function: SHA1()

The SHA1() function will generate an SHA1 hash of the parameter contents. Each time the test is run the variable is replaced with the SHA1 hash value applied to the contents of the string within the function parameter.

\$fn:SHA1(HashString)\$

Parameters:

(HashString) – String to compute hash

Context Function: PEM()

When the function is initiated the variable is replaced with the PEM format value of the referenced X509 certificate selected from the native certificate selection dialog. Unlike other context functions, this is not a dynamic function, the PEM value is statically inserted when the function is selected.

\$fn:PEM()\$

Parameters: None (certificate browser will launch when this function is selected from the context menu).

Context Function: SPACE()

If you want to add blank spaces, you can use the space(N) function.

\$fn:SPACE(N)\$

Parameters: N – number of blank spaces to add

Context Function X509Attribute()

The X509Attribute() context function allows extraction of X509 certificate property values such as Subject, Issuer, Serial Number, AltSubjectName, and other OID based extensions. When the test case is run, the X509 attribute value will replace the variable name.

\$fn:X509Attribute()\$

Parameters:

() – Standard length GUID

(Length) – GUID truncated to the value specified by the length parameter.

Context Function: Env()

The Env() context function is used to obtain a Windows environment variable value. When the test case is run, the current environment variable value will replace the variable name.

\$fn:Env(<Environment Variable Name>)\$

Parameters: Environment Variable Name

Context Function: URLEncode()

The URLEncode() context function is used to URLEncode a target value.

\$fn:URLEncode(<value>)\$

Parameters: Target Value

Global Variables

Global Variables allow for external mapping of name/value pairs to global variables that can be used within SOAPSonar anywhere variables are resolved, such as Request URI, Request Body, Request Header, Task Properties, Success Criteria rules etc. Global variable mappings can be maintained outside of SOAPSonar via the gvar.txt file located under the <install_dir>/lib folder.

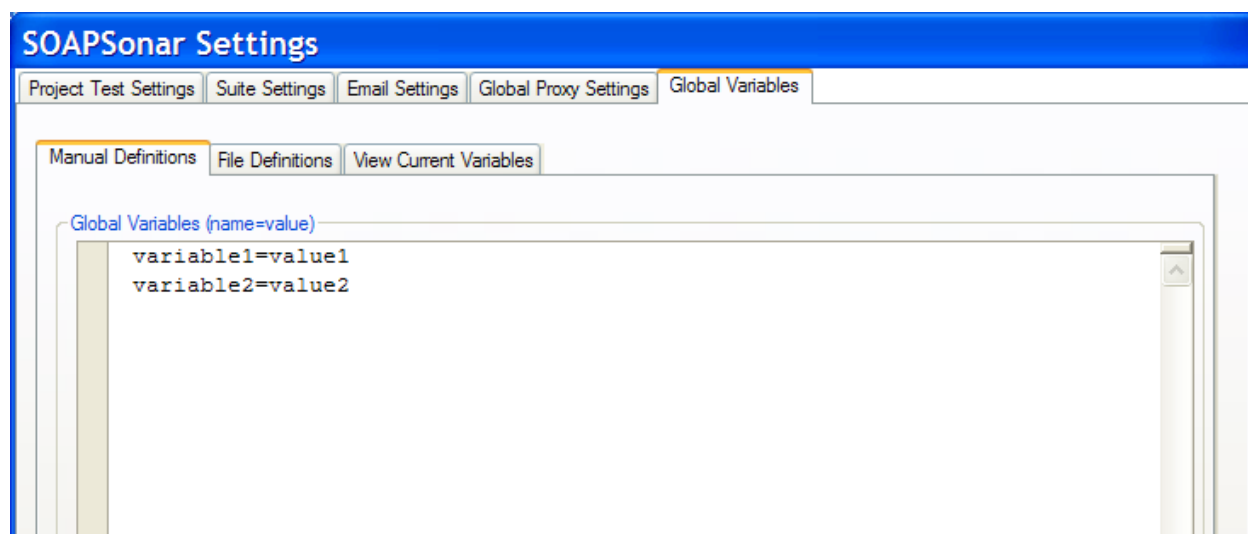
Global Variable Format

Variable references for Global Variables are denoted using the following syntax:

```
$fn : Global (<Global Variable Name>) $
```

Where:

- **Global** identified this as a global variable function
- **Global Variable Name** is the name of the Global Variable.



Global variables can be defined from the File->Settings dialog. Variable can be explicitly defined by name=value pairs, or you can point to one or more files which contain name=value pairs. The resulting compiled Global Variable file is stored in the <install dir>\lib\gvar.txt file. Thus, to alter the environment behavior of the SOAPSonar global variable settings, simply edit this file prior to running tests. Furthermore, this file can be shared with other SOAPSonar instances so that other instances can be synchronized with this instance's global variable settings.

Project Global Variables

Project global variables work the same as standard global variables, but project global variables take preference when defined, and Project Globals move with the SOAPSonar project file, unlike standard Global variables which reside and are referenced from the file system.

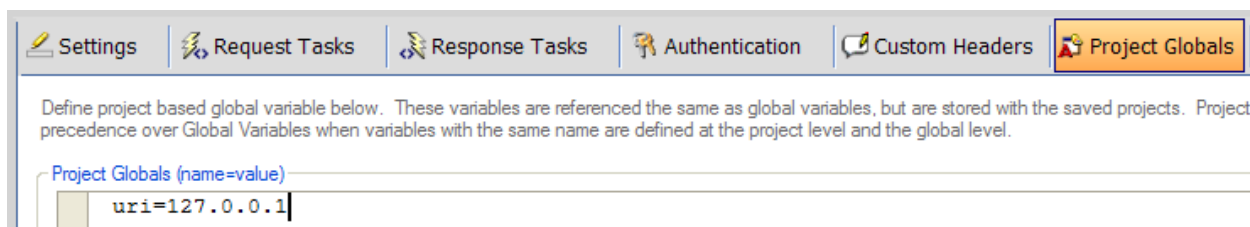
Project Global Variable Format

Variable references for Context Functions are denoted using the following syntax:

\$fn : Global (<Global Variable Name>) \$

Where:

- **Global** identifies this as a project global variable function
- **Global Variable Name** is the name of the Project Global Variable.



Project Globals are defined On the Policy Settings screen within each project. Variables can be explicitly defined by name=value pairs.

Internal Variables

Internal function variables are resolved for each test at the time it is being submitted to the back-end server. Internal variables can be referenced from request task parameters, response task parameters, and success criteria rule values. The internal variables will substitute the current test iteration value for each transaction during testing.

Internal Variable Format

Variable references for internal variables are denoted using the following syntax:

\$env : <Variable Name> \$

Where:

- **Env** identifies this as an internal environment variable function
- **Variable Name** is the name of the internal variable.

Internal Variable Function Types

The following variables are active and available for each test iteration.

REQUEST

The request internal variable allows a reference to the request document that was sent for the current test iteration

Format: \$env:request\$

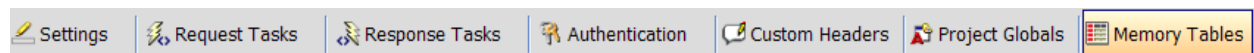
RESPONSE

The response internal variable allows a reference to the response document that was received for the current test iteration

Format: \$env:response\$

Memory Table Variables

Memory table variables allow storage of data that stays persistent through tests. In contrast to runtime variables, memory table variable association is arbitrary and does not create an automatic dependency link to another test case. Memory table variables can be statically defined or they can simply be created by setting a value to a new variable.



Static Memory Table variables are defined on the Memory Tables screen within each project, or variables can be simply be set and they will be dynamically created. To view Memory Table definitions, go to the Project Settings and click on the Memory Tables tab.

Memory Table Variable Format

Variable references for Memory Table variables are denoted using the following syntax:

\$mt : <Memory Table Variable > \$

Where:

- **mt** identifies this as a Memory Table variable function
- **Memory Table Variable Name** is the name of the Memory Table variable.

Memory table variables are hierarchical and provide both hash based indexing as well as member variable within each class. The variables can be in and of the following formats:

Table names are defined by a name and a set of indexes of one or more:

NAME [TABLE INDEX]

Member variables are defined under each table name such as

NAME [TABLE INDEX] . MEMBER_VARIABLE_NAME

Memory table variables can also be multiple levels deep, where the subclass is defined as another memory table variable:

NAME [TABLE INDEX] . NAME [TABLE INDEX] .MEMBER_VARIABLE_NAME

Some example Memory Table variable formats can be seen below:

```
test[STEP1].var1
test[STEP1].var2
test[STEP1].subtest[A].subvar1
test[STEP1].subtest[A].subvar2
test[STEP2].var1
test[STEP2].var2
```

Memory Table Variables in VBScript and JScript Code

Memory variable can be manipulated directly within the Update Memory Table Variable task. Direct manipulation is also enabled within the VBScript task and JScript tasks.

Setting and appending values to the memory table variables can also be performed from the results of the DB Query task and Runtime Variable target data captures. Memory Table variables are fully integrated into the VB Script environment allowing you to read and write Memory Table variables as well as enable direct manipulation of the collection objects of the memory table values.

Some example code showing memory table function in VB Script is below

```
Sub RunCustomTask(void)
    ' Get the index of the memory table from an ADS Variable (shows the optional
    ' ability to specify memory table variables, or portions of memory table
    ' variables dynamically from a data source such as CSV or Excel

    Dim SIndex
    SIndex = Parameters.GetVariableValue("$ads:STEP:012345$")

    'Set up the Memory Table reference
    Dim MemoryTableVar
    MemoryTableVar = "test[" & SIndex & "].AAA"

    'Add some values to the "AAA" member variable
    MemoryTable.SetValue MemoryTableVar , "value1"
    MemoryTable.AppendValue MemoryTableVar , "value2"

    'Enumerate the value collection of a memory table variable
    'and show the values in a popup Message Box
    Set memTableValues = MemoryTable.GetValueCollection("test[" & SIndex & "].AAA")

    For Each strItem in memTableValues
        MsgBox(strItem)
    Next

    'Replace the string PLACEHOLDER1 with the Memory Table string value
    Parameters.Data = Replace(Parameters.Data, "PLACEHOLDER1",
    MemoryTable.GetValue(MemoryTableVar))

End Sub
```

Memory Table Variables Template Resolution

Template resolution of memory table variables refers to the ability to resolve memory table variable as XML fragments by mapping each field definition to a prefix and suffix template. Memory Tables definitions can include the prefix and suffix that can be appended as templates. The format for this is:

```
memvariable[0],<parent>,</parent>
memvariable[0].id,<id>,</id>
memvariable[0].fullname[0],<fullname>,</fullname>
memvariable[0].fullname[0].firstname,<firstname>,</firstname>
memvariable[0].fullname[0].lastname,<lastname>,</lastname>
```

To invoke the Memory Table template resolution feature, use “**\$mtt**” when referring to the variable. For example, if the above memory table class structure is defined, then a class instance reference to a memory table object as follows:

\$mtt: memvariable[0].id\$

Would result in

<id>{id value}</id>

Or a call to a parent object as follows:

\$mtt: memvariable[0]\$

Would result in

```
<parent>
  <id>{value}</id>
  <fullname>
    <firstname>{value}</firstname>
    <lastname>{value}</lastname>
  </fullname>
</parent>
```

Memory Table Variables vs Global Variables

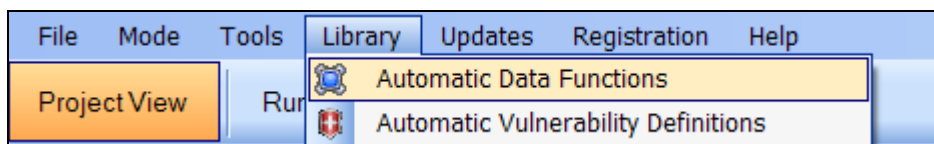
When to use Global variables instead of memory table variables depends on the scenario you are testing. Global variables, similar to memory table variables, can be stored within the project and are updated in memory as tests are run. Memory Table variable are class variables that can be arrays and can have members (i.e. MEMVAR[0].var1, MEMVAR[0].var2, MEMVAR[1].var1, etc). Also, memory variables support appending values (i.e. multiple values), vs Global Variables which have only a single name/value capability.

Automatic Data Function Variables

SOAPSonar provides a library of data generation and transform functions that allow you to create dynamic test iterations and value replacements. This library of functions is called the Automatic Data Function (ADF) library. ADF variables can be used within request data, headers, tasks, or success criteria.

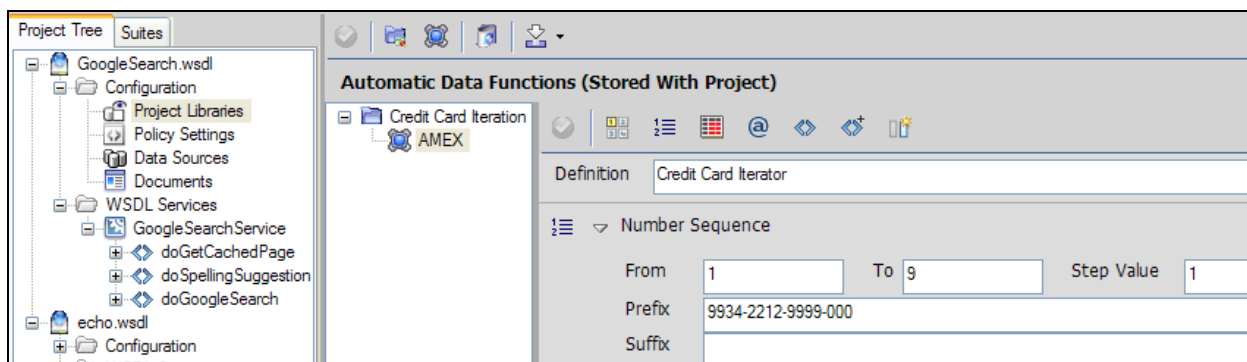
A function group can be created to perform such things as simple table enumeration, number and value sequencing, buffer injections, and more. Once a function group is created, it will be available as a variable reference when editing request data, headers, tasks, or success criteria. ADF functions can be stored within each project or globally for the installed instance. Global ADF functions are available to all projects loaded on that installed instance, while local ADF functions are transferred along with the project to other instances. To access the global instance functions, go to the Library menu and select Automatic Data Functions. To access the project based local instance functions, go to the Configuration->Project Libraries node in the project tree.

Automatic Data Function Library



ADF items defined under the Library menu item are stored to the file system and are retrieved by the running instance and available to all projects. These types of references are useful if you often use the same library group on the current installation instance. If you are sharing your project with others, consider using a shared network location, or use an ADF Project Instance Library instead.

Automation Data Function Project Instance Library



ADF items defined under the WSDL Project Libraries node are stored within the project and transferred with the project. Use these types of references if you will be sharing your project files with others.

Automatic Data Function Types

The following categories represent each type of ADF function that can be combined within a ADF group.

VALUE ENUMERATION

The value enumeration function provides enumerated list input support. Simply enter each value to define as a parameter on each line.

Sample Input: A
 B
 C
 D

Sample Result: Will create values A, B, C, D

NUMBER SEQUENCE

The number sequence function allows for creation of number sequences. The function takes a starting number, and ending number, and a step number to determine how many numbers in the sequence to generate.

Sample Input: 1, 100, 50

Sample Result: Will create values 1, 50, 100

Optional values: Prefix, Suffix

BUFFER INJECTION

The buffer injection function allows for creation of character buffers. The function takes a base character, starting length, ending length, and a step number to determine how many character buffers in the sequence to generate.

Sample Input: Z, 1, 10, 5

Sample Result: Will create values Z, ZZZZZ, ZZZZZZZZZZ

Optional values: Prefix, Suffix

OCCURENCE TRANSFORM

The occurrence transform function allows for changing the number of times an element appears in the SOAP document (cloning the current element, or omitting the current element). The function takes a base element value, starting number of element occurrences to generate, ending number of occurrences to generate, and a step number to determine how many occurrence instances in the sequence to generate.

Sample Input: NewElementValue, 0, 10, 5

Sample Result: Will omit node, clone node 5 times, and clone node 10 times.

NEW ELEMENT

The new element function allows for creating new element instances at the location of the current element. The function takes a base element name, a base element value, starting number of element occurrences to generate, ending number of element occurrences to generate, and a step number to determine how many occurrence instances in the sequence to generate.

Sample Input: NewElementName, NewElementValue, 1, 10, 10

Sample Result: Will create 1 new element node, and 10 new element nodes

NEW NESTED ELEMENT

The new nested function allows for creating new element instances at the location of the current element each nested as a child below the current element. The function takes a base element name, a base element value, starting number of nested element occurrences to generate, ending number of nested element occurrences to generate, and a step number to determine how many occurrence instances in the sequence to generate.

Sample Input: NewElementName, NewElementValue, 1, 10, 10

Sample Result: Will create 1 new nested element node, and 10 new nested element nodes

NEW ATTRIBUTE

The new attribute function allows for creating new attribute instances at the location of the current element. The function takes a base attribute name, a base attribute value, starting number of attribute occurrences to generate, ending number of attribute occurrences to generate, and a step number to determine how many occurrence instances in the sequence to generate.

Sample Input: NewAttributeName, NewAttributeValue, 1, 10, 10

Sample Result: Will create 1 new attribute for the current element, and 10 new attributes for the current element

Automatic Data Function Variable Format

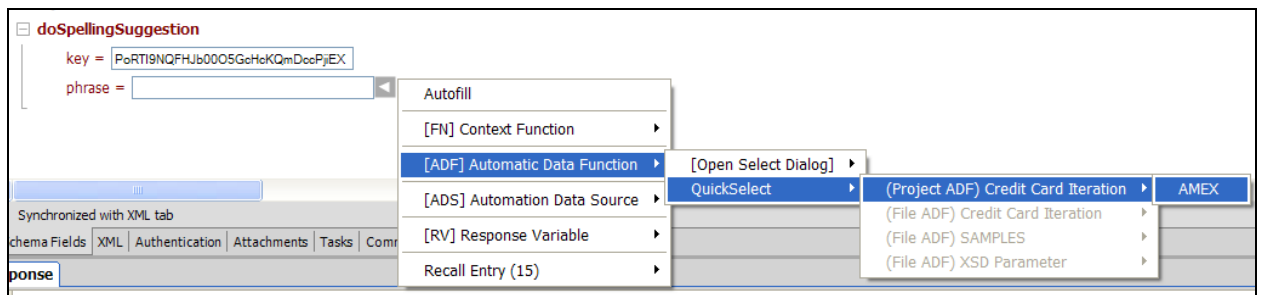
Automatic Data Function variables are denoted using the following syntax:

\$saf : RootName : FunctionGroup \$

Where:

- **RootName** is the root folder name on the ADF definition screen
- **FunctionGroup** is the sub-node which contains the allocated function definitions.

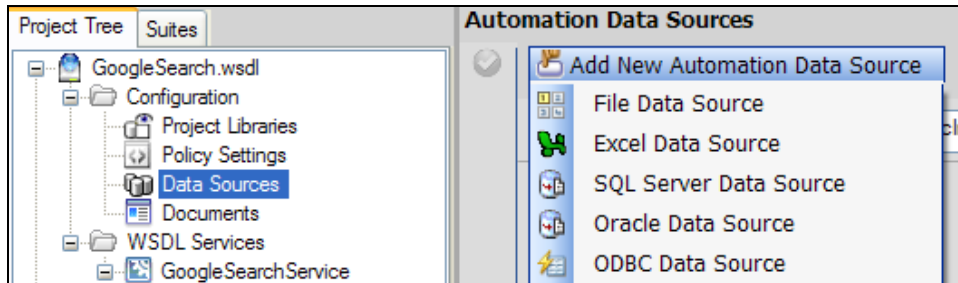
To associate an ADF group to a location within the request you can use the ADF Quick-Select context-menu to insert the ADF variable reference or use the ADF selection dialog to choose the ADF group to allocate to the parameter.



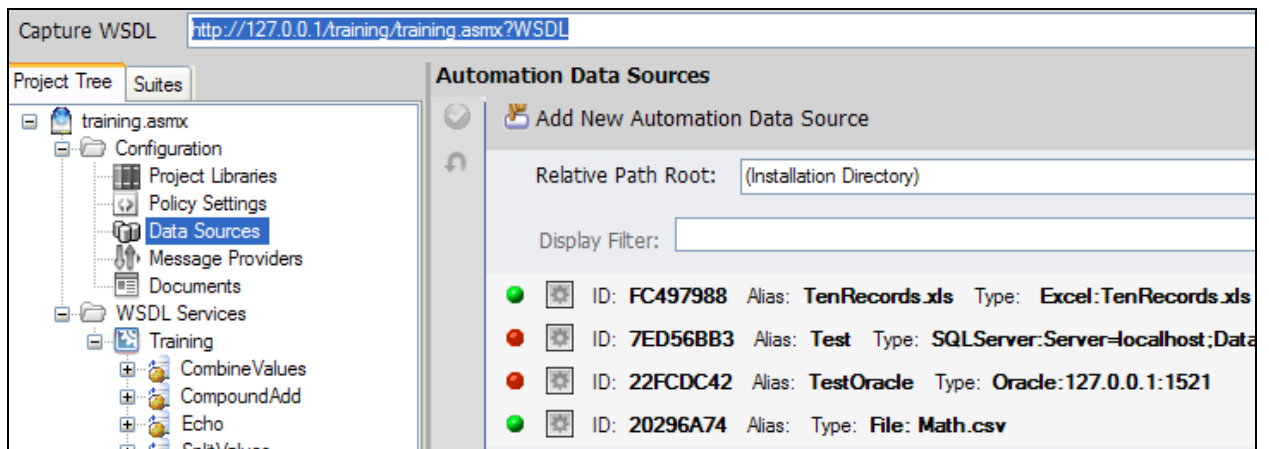
The resulting number of test iterations for this test case is determined by the ADF function(s) allocated to the request. To see the resolved variable replacements before running the test in a test suite, you can choose the [View Resolved Variables](#) tool on the request toolbar.

Automation Data Source Variables

Automation Data Source variables are used to map data from sources such as a CSV file, an Excel worksheet, or a database query to and substitute each column value in the referenced data source with a test iteration generated for each row in the data source. ADS variables can be used within request data, headers, tasks, and success criteria.



To use variables from an external source, the data source policy must first be created to indicate how to connect to the external data source. To configure a new data source policy, go to the **Configuration->Data Sources** node under the current project. Once an automation data source has been defined on the Data Sources node within a WSDL test group, each test case will be able to associate data source variables within the request body, request header, task fields, and success criteria rules.



	A	B	C
1	Column-1	Column-2	Column-N
2	A	1	Value-1
3	B	2	Value-2
4	C	3	Value-3
5	D	4	Value-4
6			
7			

Each row represents a separate test iteration when the test is run in a test suite in Run View

Each column represents a data variable that can be used for dynamic substitution

Each column of the associated data source is available to be references as an ADS variable. The number of rows in the data source represents the number of test iterations which will be executed when running the test case within a test suite. The rows from the data source can be selectively filtered using the limit rows option on the data source definition screen.

ADS Policy Options

Each Automation Data Source type has different properties required to make the connection to the data source and extract the information provided within. Once the configuration properties are correct, you can click on the **Refresh Data** icon to see the column names appear in the Column Variable field. To see the actual table data, press the **View Data** icon. Within each ADS policy type, there are also several options common to the data extraction from the referenced data table, which are explained below.

Encode XML Tags

This option will detect any XML specific data in the table cell and encode the tags using URL encoding which will convert "<" to "<" and ">" to ">" respectively. Leave this value unchecked to perform direct XML injection with no encoding.

Limit Rows

This option can be used to selectively choose the rows from the data source to use. The format for specifying rows is similar to that of specifying pages when printing. Sample format: 1,2,4-6

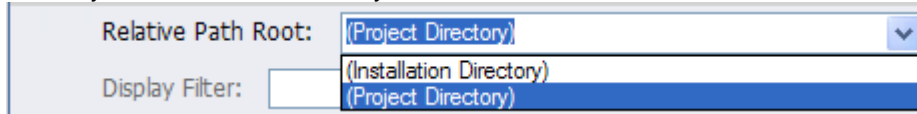
Null Value

This option allows specifying a value (including an empty string, or no value) that will result in removing the substituted node in the document where the variable has been associated. Consider a case where in the 3rd row of the data source, the resulting test substitution should omit a certain element. By using an empty value, or a known value such as "OMIT", or "NULL", the resulting substitution will detect this value

and remove the associated XML element. The default is unchecked which performs direct text substitutions with no element omission.

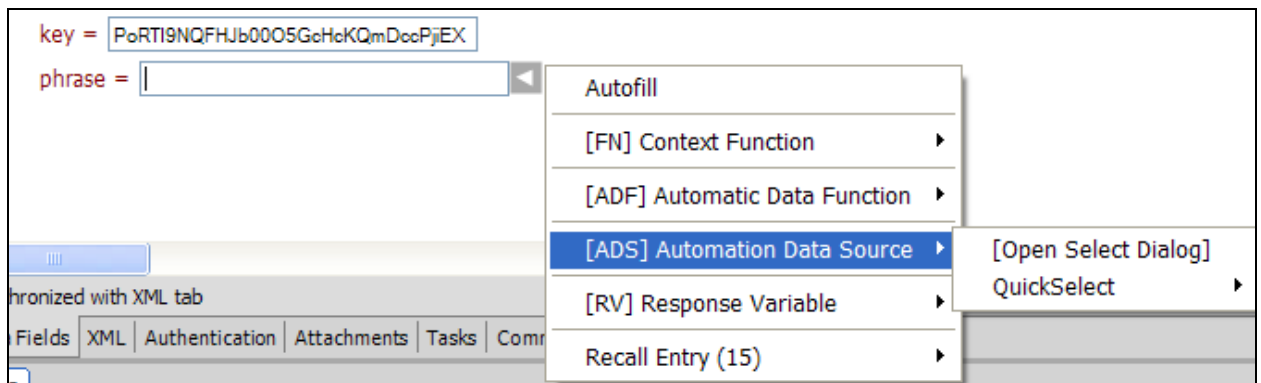
Relative Path Root

For File based ADS references, such as Excel and CSV File, target files can be defined as absolute paths, or relative paths. Relative paths (paths that do not contain a root) can be set to the installation directory or the current project directory. Relative file references will be searched for within the specified Project directory, or Installation directory.

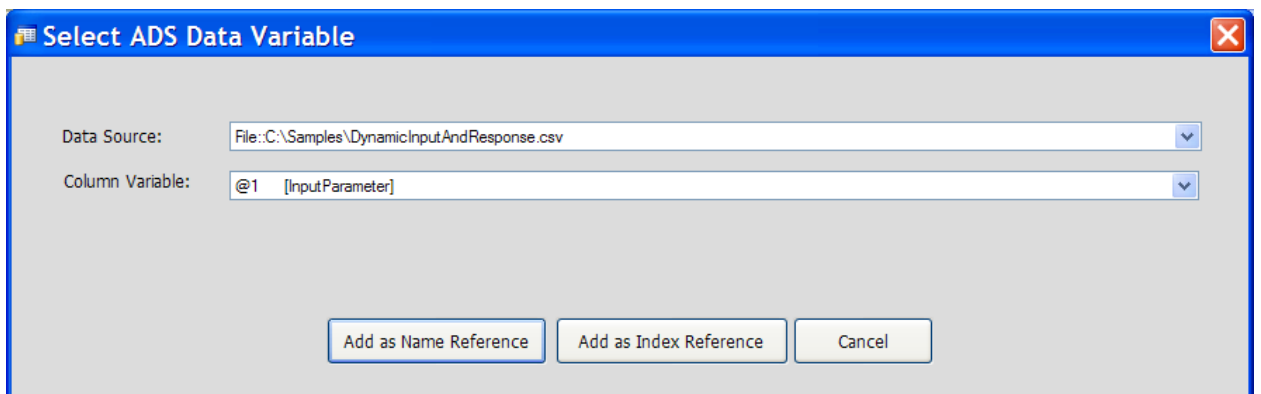


Selecting and Associating ADS Variables

To select an ADS variable, right-click in the current edit area or select the option menu at the end of the current schema field being edited.



You can choose the dynamic context menu or the ADS browse dialog to choose the ADS variable. The ADS browse dialog allows you to select from the list of available ADS variables.



By choosing the column, the ADS variable will appear. If you have created the ADS variable reference in schema fields view, the tooltip of the variable can be seen by moving the mouse over the variable to see details about the ADS variable.

key = PoRTI9NQFHJb0005GcHcKQmDccPjEX

phrase = \$ads:InputParameter:{dc525565}\$

Automation Data Source Reference

Type: File

Source: C:\Samples\DynamicInputAndResponse.csv

Data Series in Rows

Data Variable: InputParameter

([CTRL] + Double-Click To Edit Variable Settings)

When running this test in a test suite, there will be a separate test iteration created for each row in the ADS data source and the ADS variable will be replaced with the actual column data in each row.

Excel vs CVS Formats

There is a known Microsoft limitation with regard to type mappings from Excel spreadsheets to code data types as outlined in Microsoft KB Article 194124. To prevent the issues in data type conversion which may occur due to this known limitation, it is recommended you use CSV format rather than XLS format. In Excel you can configure this simply by going to **File-Save As** and choosing the CSV file type.

Automation Data Source Variable Format

Automation Data Source variables are denoted using the following syntax:

\$ ads : ColumnName : DataSourceID \$

Where:

- **ColumnName** is the name of the column in the data source table where values are to be extracted. You can also refer to the column index for this value using the “@” symbol followed by the column index (e.g. \$ads:@1:{ID}\$, \$ads:@2:{ID}\$, etc)
- **DataSourceID** is the unique Reference ID used to associate the variable back to the definition of the connection and settings to the defined data source

Runtime Variables (Request and Response Capture Variables)

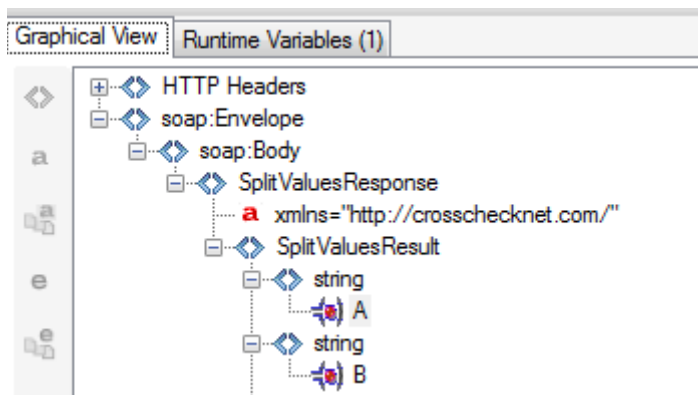
Runtime variables represent dynamic memory storage and links for test case request and response data when a test or set of tests are running. Runtime variable are used to capture information from a previous test case request or response that can be referenced in variable format in other tests. These referenced create a dependency (test chain link) which invokes the dependent tests in order to resolve the variable references based on real-time test data.

Associating runtime variables from another test case creates a dependency chain. This provides the ability to perform sequence testing where values of one test are used by another test. Runtime variables are used to dynamically capture the request or response values (Header, XML, Element, or Attribute) at the time the test is run and then substitute the runtime value in place of the variable reference. Runtime variables can be associated across WSDL and custom project groups to chain multiple services, or within the same service, operation, etc. to chain tests together. Any test case defined in the project can be chained with any other test in the project.

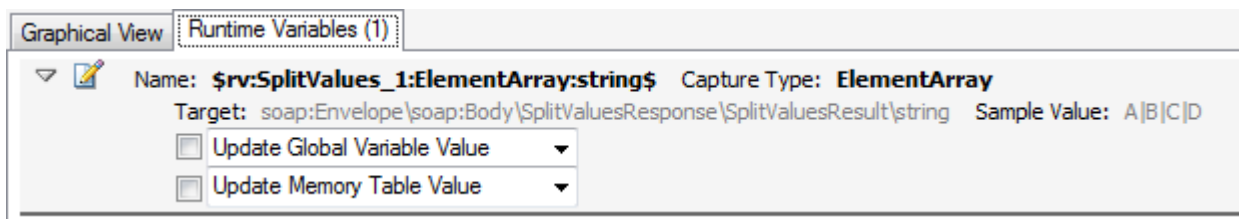
Runtime dependency variables are defined on the test case where they are to extract the data from. The data extraction can be from the request or the response of a running test, and also be values extracted from the HTTP header, or the XML body. In the Request and Response tabs is a runtime variables tab which is used to create Runtime variables can be used in header, body, task field entries, or success criteria values in subsequent tests.

To create a Runtime Variable on a test case, go to the Runtime Variables tab on the Request or Response panel and choose from the graphical tree view what value(s) you want to create references. Use the toolbar on the left, or right click on the tree area. Once the named variable is created, it will appear in all other tests in the project as a runtime variable available for selection.

Graphical View



Rule View



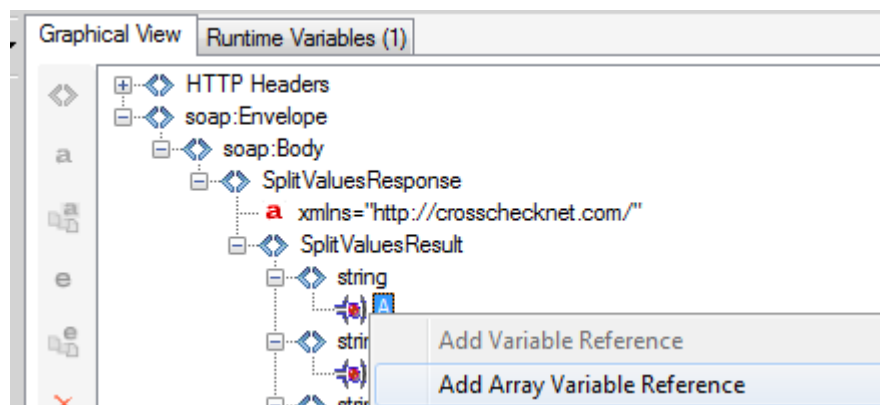
Update Memory Table with Runtime Variable

Memory table variable offer flexible means of capturing Node, Element, or Attribute information and preserving it for later use. This variable option provides all of the existing capability of list variable and global variable mappings. The list and global variable settings remain for legacy configurations. See [Memory Table](#) variables for more information on how to use Memory Table variables to capture, preserve, aggregate, and apply functions to captured data.

By mapping the runtime dependency variable to a Memory Table variable, this provides the means to reference this value later in the test sequence, without creating a specific test dependency link to the current test. Not creating a specific test dependency link offers more flexibility to define the sequence of dependency, and the order of dependent test execution.

Capture Runtime values as Array Variable

When values are returned in a repeating structure such as a collection, or array, you can capture all values at that target array location by selecting the “Add Array Variable Reference” option



Once captured as an array, the dynamic values can then be retrieved via index based notation on the variable formats, where indexes start from 0. For example:

`$rv:TestCaseName:CaptureValueType:VariableName(0)$`

Where (0) is the index of the value to retrieve. Value indexes can be comma separated for multiple values such as

`$rv:TestCaseName:CaptureValueType:VariableName(0,1)$`
`$rv:TestCaseName:CaptureValueType:VariableName(2,5,9)$`

or the list variable can be used directly to collect all values stored in the list:

`$rv:TestCaseName:CaptureValueType:VariableName$`

When multiple values are retrieved, the results are concatenated together. If you want to separate results using a delimiter, you will need to specify each index separately and add your own delimiter. For example:

`$rv:abc:ElementArray:VariableName(0)$<mydelim>$rv:abc:ElementArray:VariableName(1)`

Mirror Value in Global Variable

When enabled, this option will store the runtime variable value in a global variable with the same name. This option is useful if you want to store a value and use it in another test case, but do not want an explicit dependency to be created on the runtime variable reference.

Include Encoded XML setting

When sending or receiving XML content in string encoded format, this setting will decode the XML within the existing document tags and allow variable capture within the encoded XML.

Runtime Variable Format

Runtime variables are resolved for each request at the time it is being submitted to the back-end server (i.e. at runtime). Runtime variables can be used within request data, headers, tasks, or success criteria.

Runtime Variables are denoted using the following syntax:

\$ rv : TestCaseName : CaptureValueType : VariableName \$

Where:

- **TestCaseName** is the name of the current test case which this variable is generated
- **CaptureValueType** is the type of data to be extracted from the defined location (value, attribute value, XML fragment)
- **VariableName** is the variable name used to help identify this variable for selection in other test cases.

Or for Array Values:

\$rv : TestCaseName : CaptureValueType : VariableName (0)\$

Where (0) is the index of the value to retrieve. Value indexes can be comma separated for multiple values such as

\$rv:TestCaseName:CaptureValueType:VariableName(0,1)\$
\$rv:TestCaseName:CaptureValueType:VariableName(2,5,9)\$

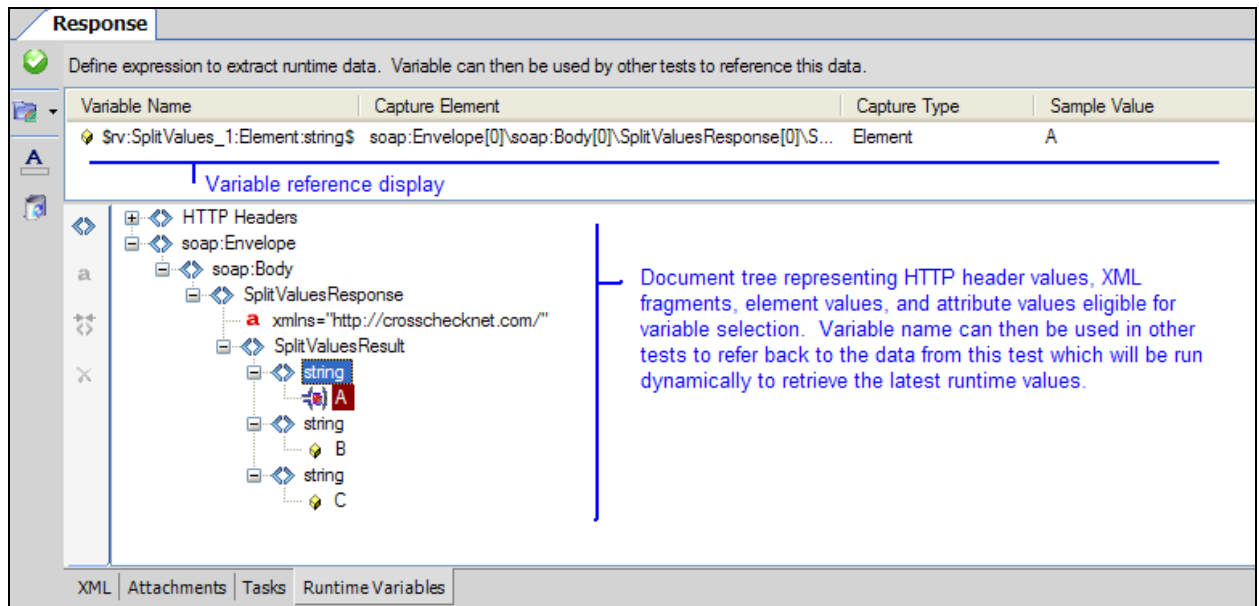
Runtime Variables can be configured to capture response HTTP header values, XML fragments, XML element values, and XML attribute values.

Capture / Replacement	HTTP Header Value	XML Attribute Value	XML Element Value	XML Node
HTTP Header Value	X	X	X	X
XML Attribute Value	X	X	X	X
XML Element Value	X	X	X	X
XML Node	X	X	X	X

The document structure and header structure are used to build the path designator to the location of the data that is to become the new parameter. The type of Runtime Variable can be any of the following:

- Attribute:** Runtime Variable path points to an XML attribute to extract the value from
- Element:** Runtime Variable path points to an XML element to extract the value from
- Node:** Runtime Variable path points to an XML node to extract the entire XML fragment from

The runtime variable definition area is found under the “Runtime Variables” tab in the Request and Response tabs. Selecting this tab will show the Runtime Variable capture screen.



Once you have created at least 1 Runtime Variable in your project, it will be available for selection from any other test case in your project.

Capturing Array Targets with Runtime Variables

To handle values where the target data comes back in an array format where there are actually 1 or more values at a certain depth in the document that represent a collection or array of values, the array capture capability of the runtime variables enables you to automatically capture all array values for a target attribute or element and normalize each value as a string value consisting of the segments of the array separated using the “|” delimited.

For example, say an XML document contains 4 array values as follows:

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SplitValuesResponse xmlns="http://crosschecknet.com/">
      <SplitValuesResult>
        <string>A</string>
        <string>B</string>
        <string>C</string>
        <string>D</string>
      </SplitValuesResult>
    </SplitValuesResponse>
  </soap:Body>
</soap:Envelope>

```



To extract all of the target element values from this array into a runtime variable, we can use the ElementArray capture feature of SOAPSonar runtime variable capture.

Create variables based on runtime content. Variables can reference HTTP Headers, XML fragments, XML elements, and XML attributes

Variable Name	Capture Element	Capture Type	Sample Value
\$rv:SplitValues_1:ElementArray:string\$	soap:Envelope\soap:Body\SplitValuesResponse\SplitValu...	ElementArray	A B C D

Element Array Capture (converts to string with each value concatenated using the "|" delimiter)

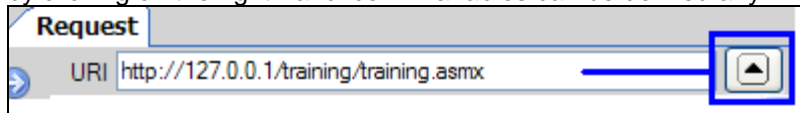
Visual indication of all elements/attributes targetted by the array capture variable

The ElementArray and AttributeArray capture types will iterate through all array items for the target component and extract each value and create a concatenated string containing all target values separated by the "|" delimiter

For this example above the runtime variable \$rv:SplitValues_1:ElementArray:string\$ would resolve to the string-based array mapped values "A|B|C|D", or whatever set of array values were seen in the XML at this level at runtime. For more details, see [Capture Runtime values as Array Variable](#).

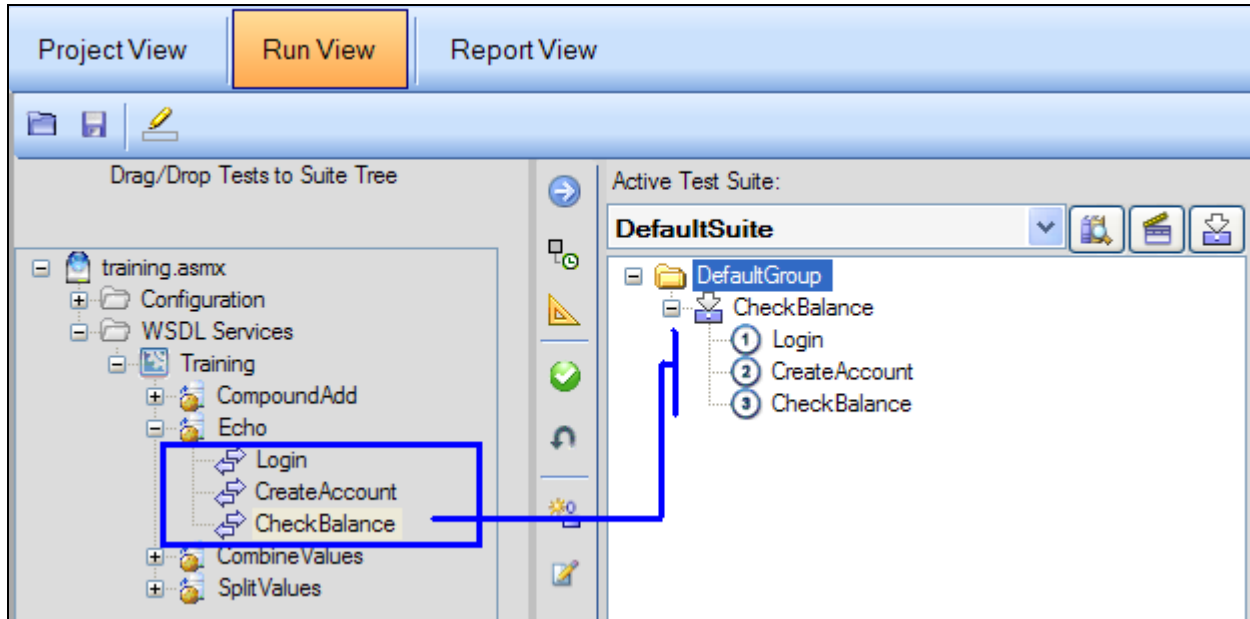
Request URI Variables

The Request URI defines the endpoint to send the test to. This value can be statically defined, or it can be build with any of the aforementioned test variable types. The request URI field provides a build screen by clicking on the right-hand icon. Variables can be defined anywhere within the URI.



Automatic Test Case Chaining

Test case chaining is enabled by default and results in SOAPSonar detecting dependent runtime variables in each test. Dependencies then appear in the Run View when the test is added to a test suite. The dependency chain reflects the order in which the dependencies will be resolved.



For example, in the screenshot above Login has a runtime variable capturing its response value. The CreateAccount test uses the runtime variable response from Login and a new runtime variable is created for the CreateAccount response. CheckBalance then uses the runtime variable from CreateAccount completing the test chain. When CheckBalance is added to a test suite the chain will appear indicating that each time CheckBalance is run, Login and Create Account will run in sequence first and the runtime variables will be replaced in real-time by the test values.

If you want to disable the SOAPSonar auto-dependency checking, click on the “disable auto test chain dependency checking” checkbox on the Test Suite properties.

A test case chain is a sequence of tests which have dependencies on each other where a portion of the request or response of one test case is used as the input for the next test. Test case chains are created using [runtime variables](#). Associating a runtime variable with a test case creates a dependency on the test case which the Runtime Variable belongs to. A test case with a runtime variable can itself create a runtime variable based on its request or response which in turn can be used by another test case which would create a 3 level test chain. The test dependency chain will appear in the run view test suite tree when associated with a test suite. The test dependency chain will run in sequence as it appears in the test suite tree. Runtime variables are populated in real-time when the test is run and the values of the variables are replaced in the subsequent dependent test case with the real-time response data.

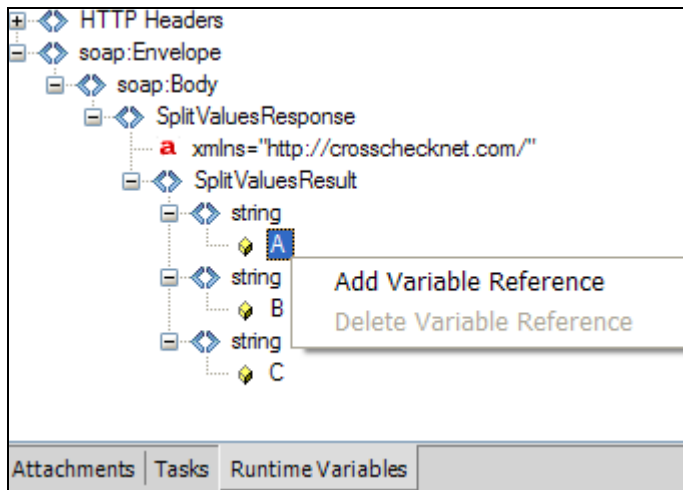
Steps to Create a new Test Case Chain

1) Plan the dependent sequence

The first step in creating a test case chain is to determine the order of sequence for invoking the tests. The first test will be fired before the second test, the second test will be fired before the third test and so on until the final test represents the end of the dependency chain.

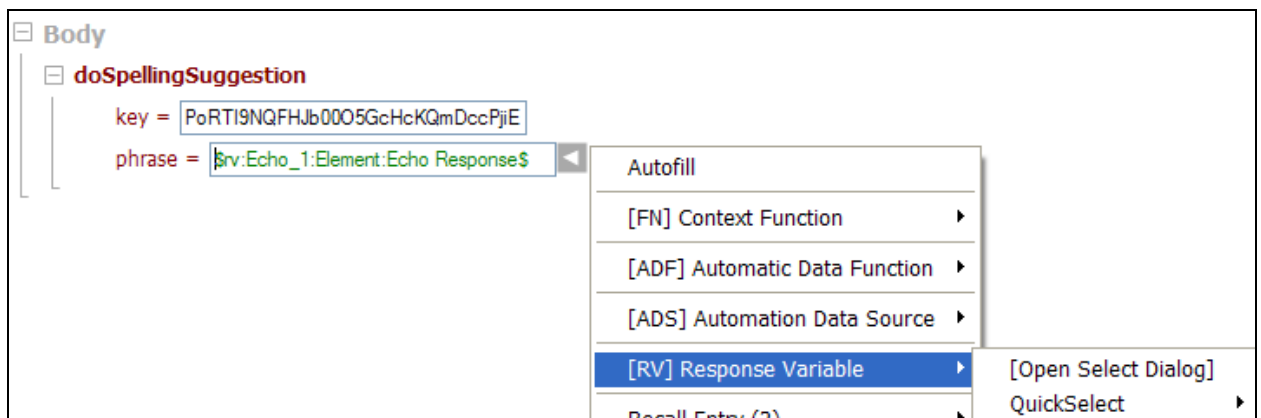
2) Start building the runtime variables

The dependency between test cases in a test case chain is created through the creation and referencing of runtime variables. Starting with the first test case in the chain invoke the back-end service and go to the Request or Response tab and select the Runtime Variables tab and create the runtime variable reference to the data (attribute, element, or fragment) that will be used by the subsequent test case. This will create a data reference that will be substituted whenever the referencing test is run.



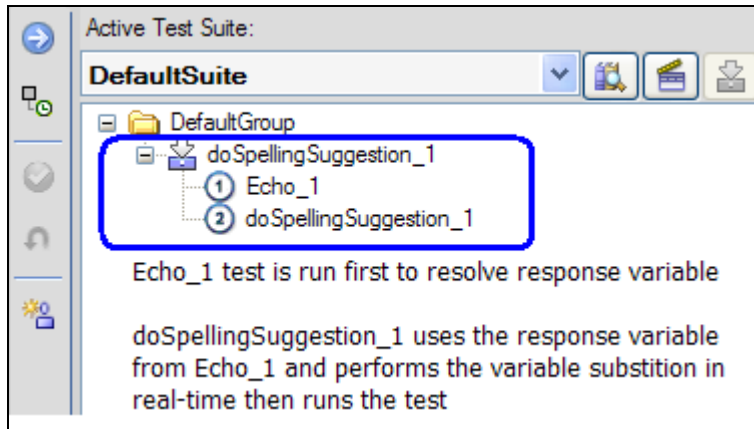
3) Associate the runtime variable to the test case

Runtime variables can be associated with a test case in the HTTP header, Schema Fields, raw XML, task configuration fields, and success criteria fields. To associate a Runtime Variable, right-click for the context menu wherever you are editing values (XML View, Header, Tasks, Authentication, etc).



4) View the sequence of tests for the test chain

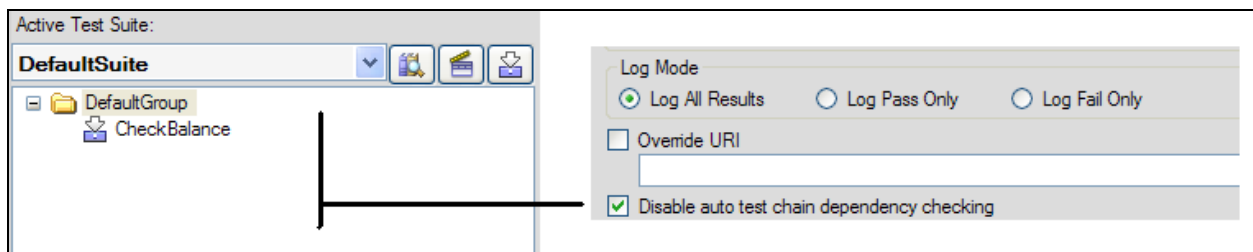
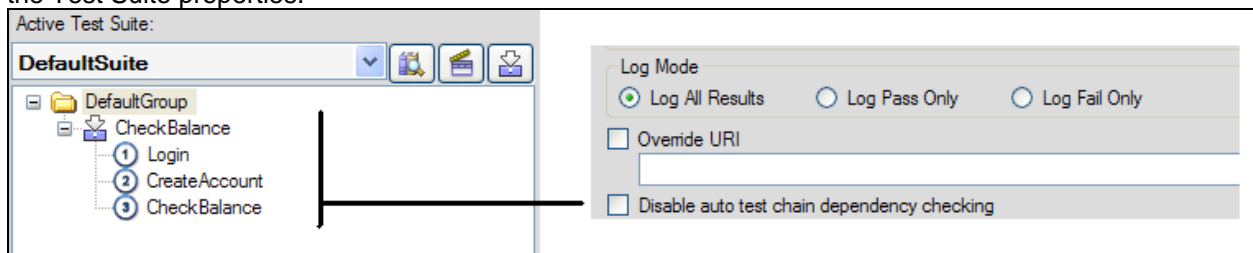
To see the test case dependency hierarchy drag the final test case in the dependency chain to a test suite. The test suite will show a “+” expand node for test cases which are dependency chains. Expand the node to see the list of tests in the dependency chain. The list is run in the order seen from top to bottom. The bottom test case is the primary test and each test above it are the dependent tests which have Runtime Variable references linking them together.



When the test suite is run from Run View each test in the dependency chain will be run and each Runtime Variable will be obtained dynamically for each test response in real-time.

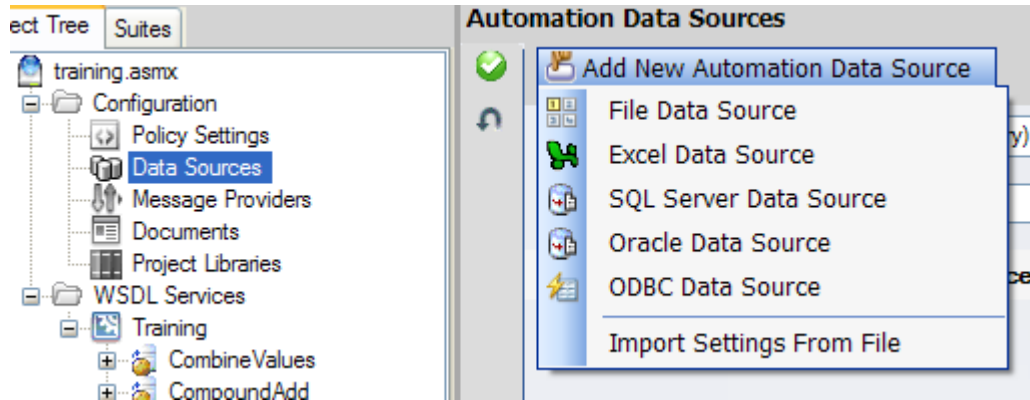
Disable Automatic Test Dependency Detection

If you want to disable the automatic SOAPSonar dependency checking feature to force only single tests to run and not run dependencies, click on the “disable auto test chain dependency checking” checkbox on the Test Suite properties.



DATA SOURCES

Data sources provide the means to drive test iterations, perform data substitution, define dynamic input and response values, as well as define test suite test sequences and data flows. Data sources can be defined in the Configuration->Data Sources screen. A data source can be a CSV file, an Excel Spreadsheet, or any ODBC enabled database.



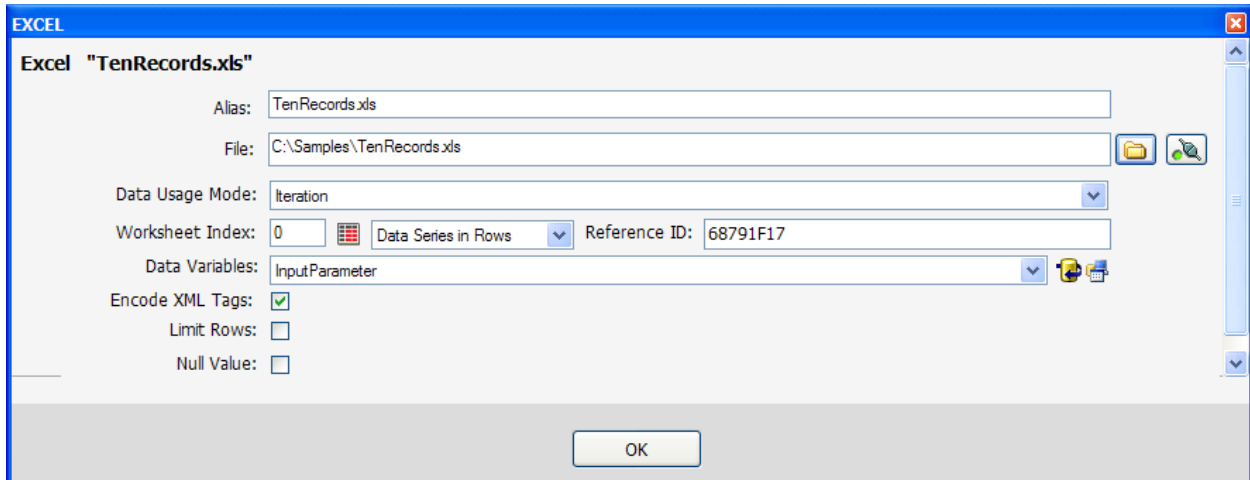
Iteration Mode Data Sources

Once a data source is created, you can then set the Data usage mode that will be used to process the target data in the table. Iteration mode will process the column names and convert each column name to a variable reference. These variables can then be associated in various locations when defining the test case request, or for dynamic validation of test case response information

In iteration mode, when a test is run that detects one or more variable references to the Data Source, each row of the data table will be run sequentially until all rows have been read. The number of test iterations is the number of rows in the data table that contain values.

Data sources can be limited to a certain number of rows, or specific rows by enabling the Limit Rows option.

If the Null Value parameter is set, you can defined a test string that when detected in the data source table, will result in the element where the variable reference has been associated to be removed. This enables building of a default request template and then using the Null Value feature to remove nodes from the request.



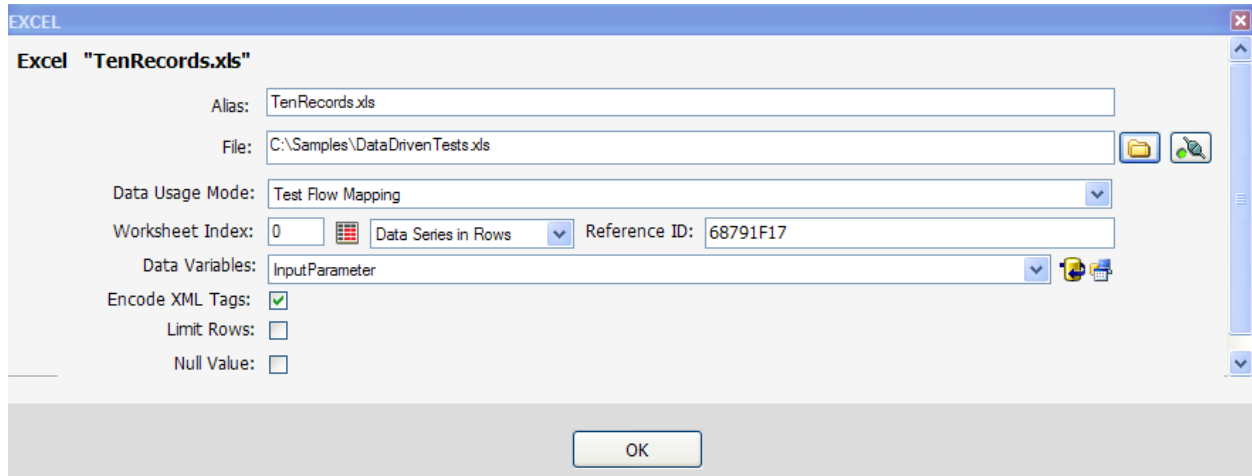
Test Flow Mapping Data Sources

Once a data source is created, you can then set the Data usage mode that will be used to process the target data in the table. Test Flow Mapping mode will process the data in the table and the first 2 columns of the table will be used to create the Group and Test references which can be used to link a test suite in order to define the test flow. The first column is the Group name, the second column is the Test Case name. Tests are associated based on the name of the test case. The data format of the table is as follows:

	A	B	C	D
1	Group	TestName	SampleNumber	SampleLetter
2	Group1			
3		Test_1		1 A
4		Test_1		2 B
5		Test_2		3 C
6	Group2			
7		Test_1		4 D
8		Test_1		5 E
9		Test_2		6 F
10	Group3			
11		Test_3		7 G
12		Test_4		8 H

Each additional column after column 2 is processed and converted to a variable reference. These variables can then be associated in various locations when defining the test case request, or for dynamic validation of test case response information and when the test is ran, each test reference will link directly to the specific row and the data substitution will occur for that explicit row.

If the Null Value parameter is set, you can defined a test string that when detected in the data source table, will result in the element where the variable reference has been associated to be removed. This enables building of a default request template and then using the Null Value feature to remove nodes from the request.



HP QC Linked Data Sources

If your license key enables the HP Quality Center native integration features, you can enable the direct link capability to directly link an Excel or CSV data source to HP QC project attachments. This allows central, shared, data source definition and consist usage of the same data sources across multiple teams or projects.

To enable this feature, go to File->Preferences and Settings, click on the HP Quality Center tab and enable the checkbox for "Enable linking to attachments in HP Quality Center project repository". Once enabled, a new button will appear to the right of the browse button icon which when pressed will allow you to authenticate to an HP QC project and link to an attachment for the file source. The data source definition works the same as it does when referencing the information from a file, except the data is extracted from HP QC instead.

TEST TOOLS

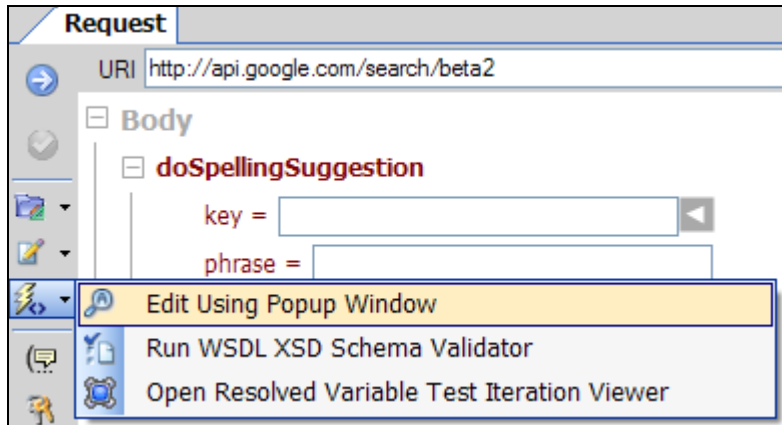
When authoring test cases there are several configuration and viewing tools available to assist in the test generation process. Tools are available on the request and response toolbar as well as on the Tools menu.

Subtopics in this section include

- [Edit Using Popup Window](#)
- [Run WSDL Schema Validator](#)
- [Resolved Variable Test Iteration Viewer](#)
- [Native PKI Management](#)
- [UDDI Browser](#)
- [Auto Fill Schema Fields](#)
- [Map Fields to Automation Data Source](#)

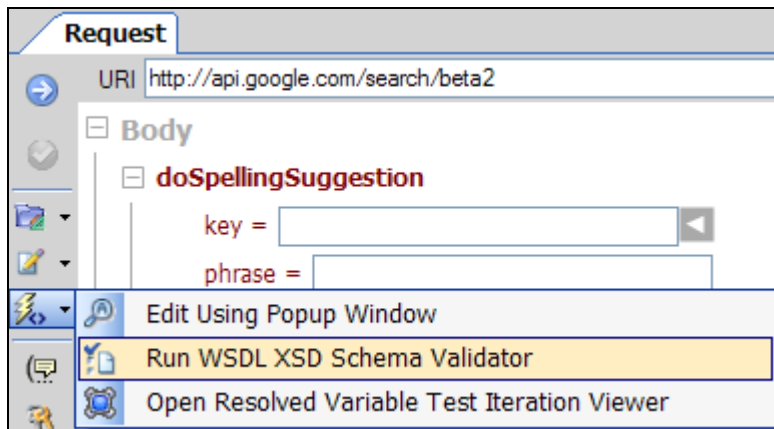
Edit Using Popup Window

In project view when editing a test case on the schema fields view or the XML view, the current edit screen can be viewed and edited in a separated window for more editing and viewing screen area.



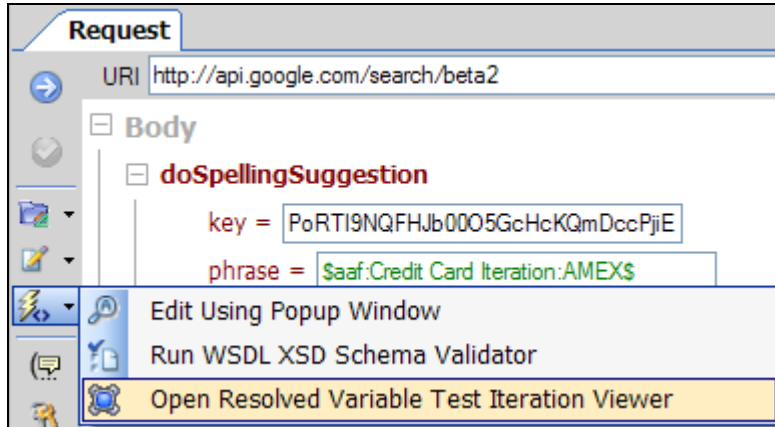
Run WSDL Schema Validator

In project view when editing test case requests or viewing test responses, this data can be validated against the WSDL schema to check that the structure and data integrity meet the requirements of the schema.



Resolved Variable Test Iteration Viewer

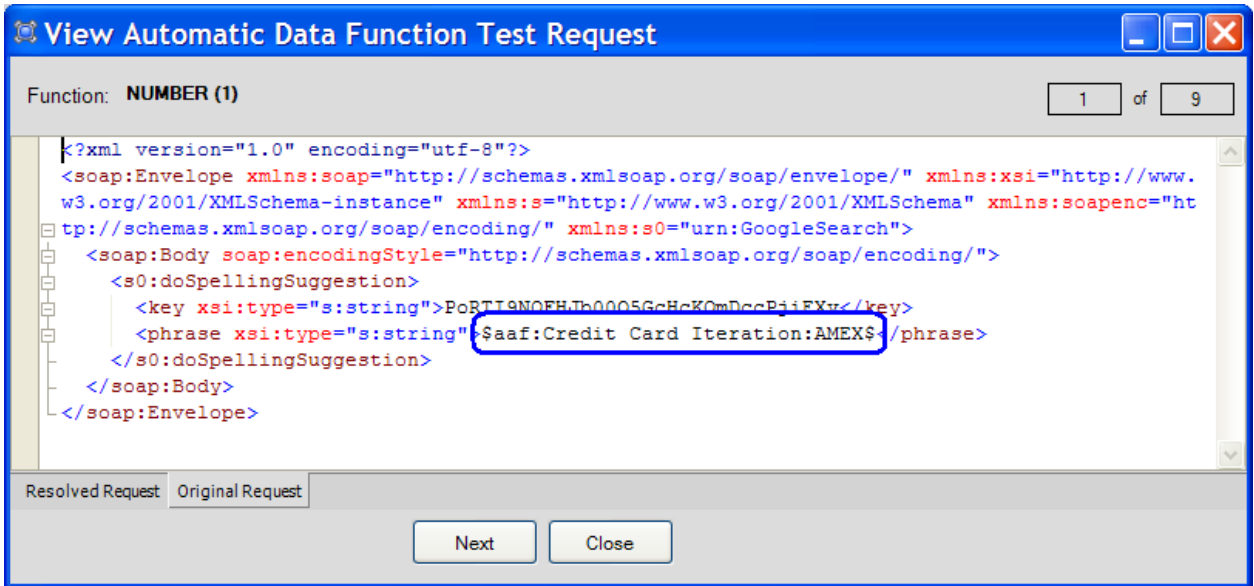
In project view when editing test cases with variable references enabled the resolved variable replacements can be viewed at test design time to see the value substitutions that will occur when the test is run in a test suite. Please refer to the [Using Variables in Test Case](#) section for more information about how to create variable references to response values, data sources, and internal library functions.



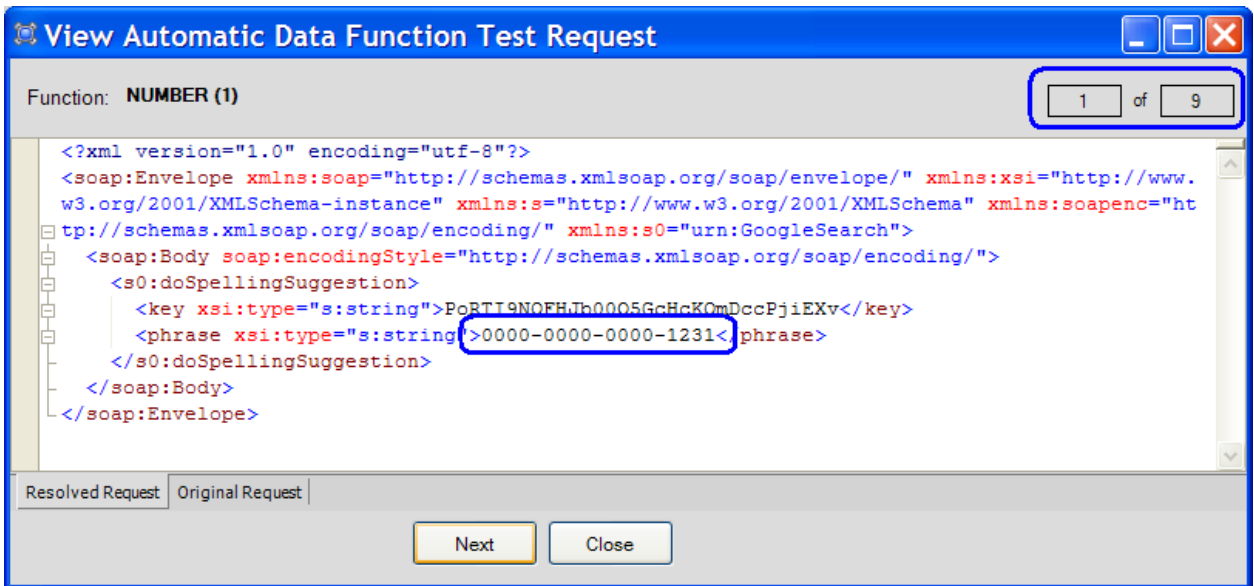
If there are detected variables in the request then the resulting viewer will appear with tabs to see the original request and each test iteration resulting from the data source substitutions. For ADS variables this will show each column in the database and its value replacement. For RV variables, this will show the value of the Runtime Variable at the time the value was captured (these values will be resolved to their actual test response data when the test is run). For ADF variables, this will show each value generated by the ADF function group. Context functions will be resolved to the respective referenced function.

The viewer dialog allows navigating among the iteration substitutions to preview each iteration that will be generated when the test is run in a test suite.

The original request tab shows the request with variables.



The Resolved Request tab shows the resolved values for each test iteration.



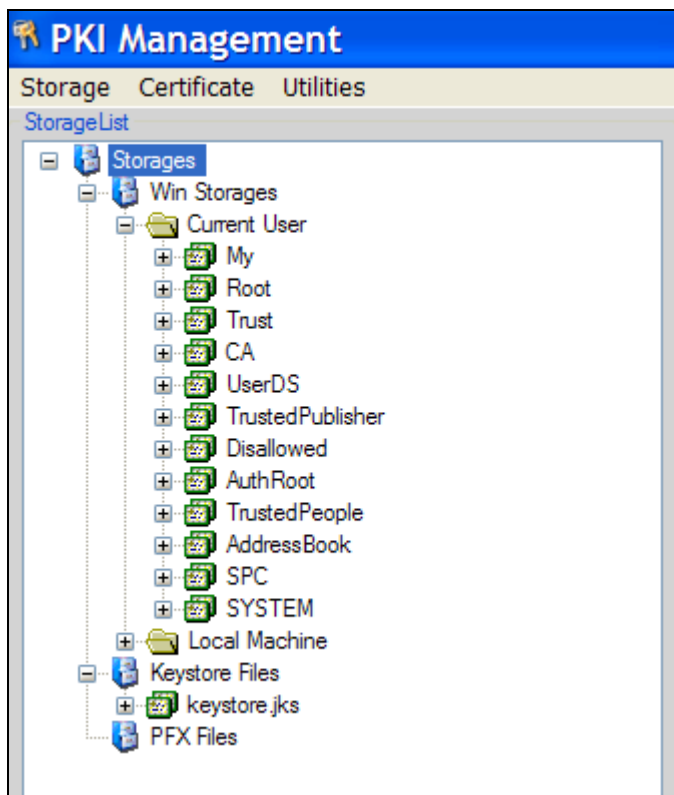
In the above screenshots there is a variable which will result in 9 test iterations being generated when the test is run in a test suite from Run View. This dialog allows navigating through each test iteration to see the value substitutions that are to occur in the document.

NATIVE PKI MANAGEMENT

SOAPSonar provides integrated PKI management features to allow direct, native access to the windows keystores. Java keystores, PFX files, and SmartCard CardReaders. Anywhere you require access to a certificate or private key, an icon will be provided to access the native PKI interface to browse and select the keying material required.

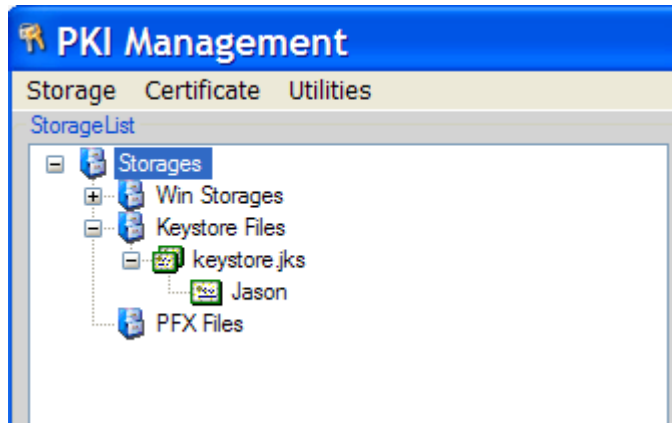
Windows Keystore

All keys in your windows keystore are presented for selection. You can also import, export, and generate keys in the “Win Storage” section.



Java Keystores

A Java keystore can be used to access public and private keys. To access keystore files, copy files with .jks extension to the SOAPSonar “keys” subdirectory. These will then appear under the “Keystore Files” folder when viewing the PKI screen. You will be prompted for a password to access the keys within the keystore.



PFX Files

SOAPSonar provides access to file based PFX keys under the “PFX Files” folder. To access PFX files, copy files with .pfx extension to the SOAPSonar “keys” subdirectory. These will then appear under the “PFX Files” folder when viewing the PKI screen. You will be prompted for a password to access the private key of the PFX file.

SmartCard Keys

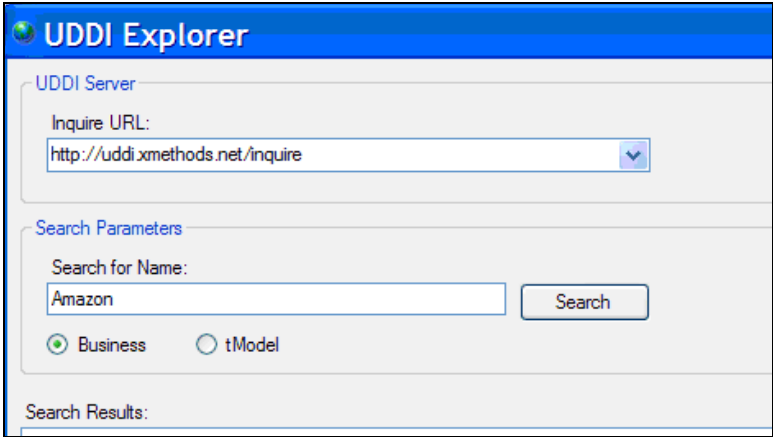
SOAPSonar provides the ability to use keys from a SmartCard to perform digital signatures, encryption, decryption, and SSL X509 mutual authentication. For more information how to configure and use smart card readers and card keys, please refer to the [SmartCard Integration](#) section.

Key Alias Management

For all key aliases defined within the current session, you can manage these definitions and associate, or de-associate aliases at the global level. To manage these aliases, select “Key Alias Management” from the Tools menu.

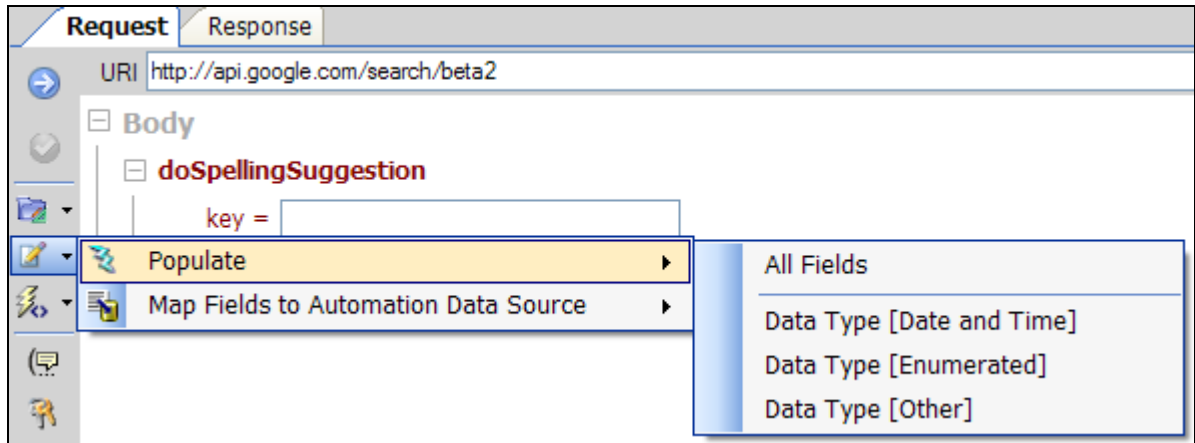
UDDI Explorer

For browsing a UDDI registry for services matching particular search criteria SOAPSonar provides a UDDI explorer from the Tools menu.



Populate Schema Data Types

In project view when editing a test case on the schema fields view you can choose to automatically create data entries according to the schema data type defined by the WSDL schema. Selective choices are presented for the common data types as well as the ability to auto-fill all data entries based on schema type.



Map Fields to Automation Data Source

In project view when editing a test case on the schema fields view you can map all the enabled schema field items to a CSV data source. This will extract the current values from the request and associate them in the data source with new ADS variable references. Each enabled schema field will be replaced with a variable reference to the data source value. Options for field mapping include:

Array XML Mapping

This option allows data source substitution to substitute dynamic array sizes along with standard element value replacement. With this option selected a new "Series by Row" data source will be generated from the schema field values enabled. For array items, the XML structure of the array will be extracted directly into the new CSV data cell. Use this option if you want to be able to change the dimensions of an array from the data source definitions for dynamic arrays.

Static Structure Mapping

This option will create a "Series by Column" data source where the first column contains the variable name and test iterations are generated by each successive column of values detected (this is the reverse of a "Series by Row" data source where a new row results in a new test iteration). This option will map each element in the schema fields according to its hierarchy in the SOAP document. Since this is a direct structure map, the resulting variable names are based on the XPath of each node.

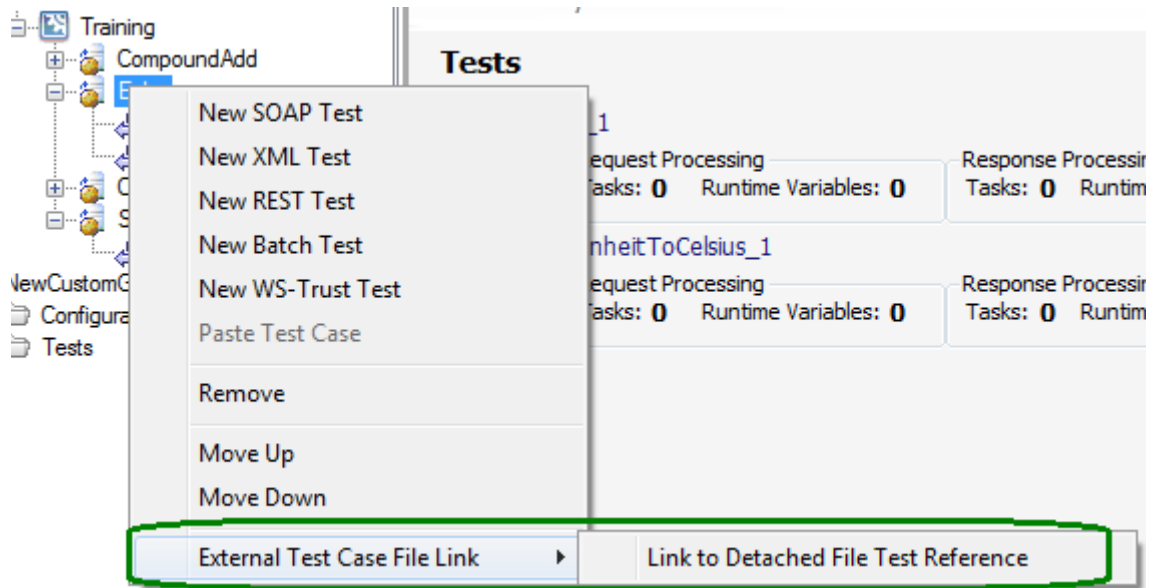
LINKED TEST CASES

SOAPSonar test cases can be detached from the .ssp project file and stored instead as links to disk locations. Linked test cases work just as standard test cases, but the content and settings for the test case come from the linked file path rather than from the SOAPSonar project.

Detached linked test cases provide the ability to publish shared test cases that can be used across several different projects that all share a common base set of linked tests.

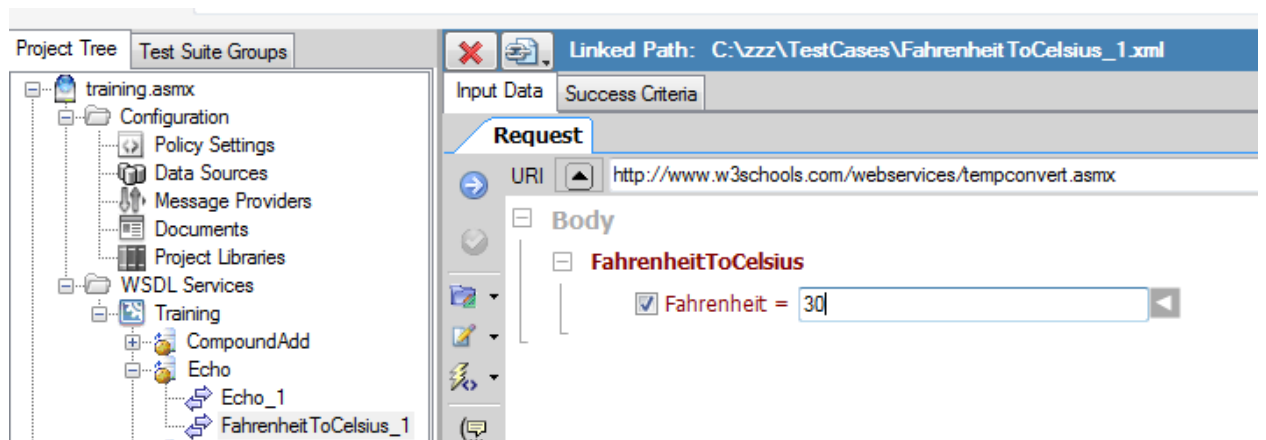
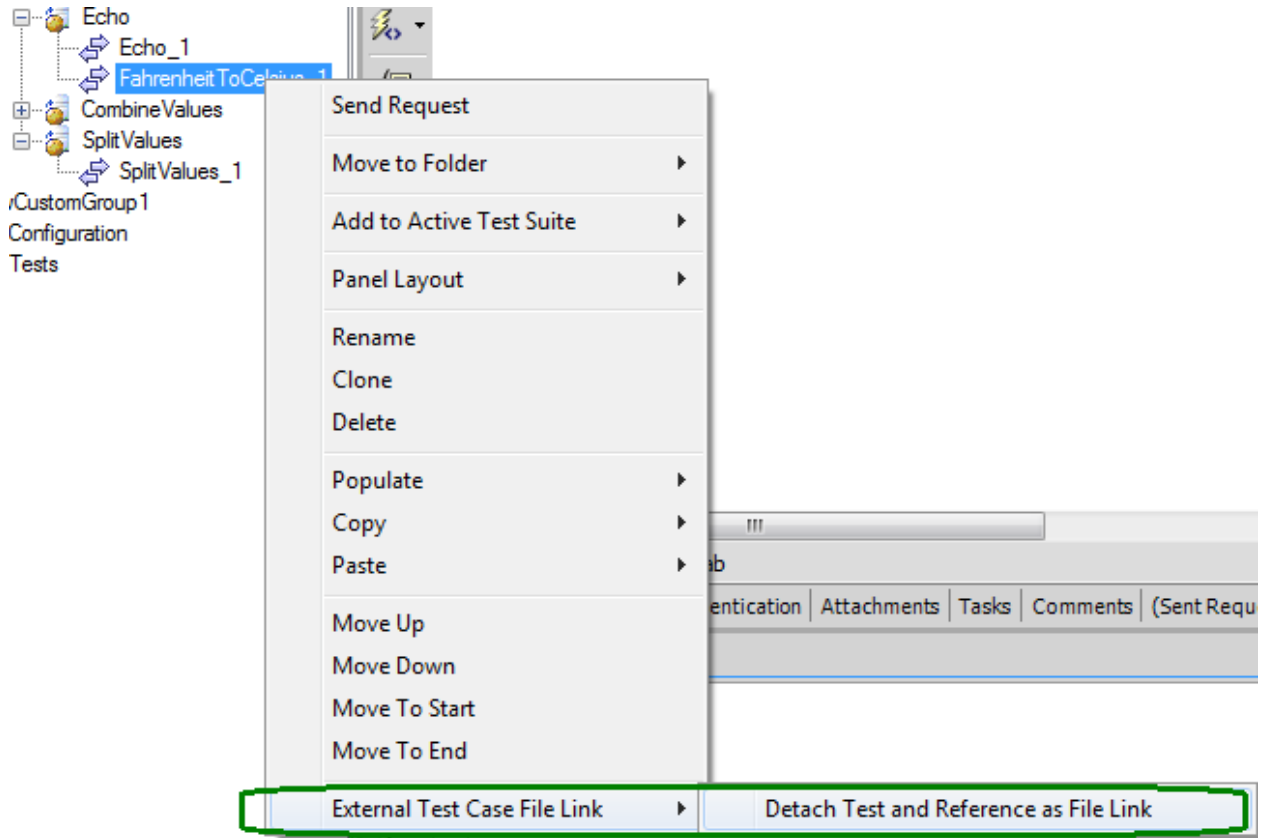
Importing a Linked Test Case into the Project

Detached test cases can be linked to the project by going to the Operation node (WSDL Project) or the Tests node (Custom Project) and right-click to select the **External Test Case File Link -> Link to Detached File Test Reference** context-menu option.



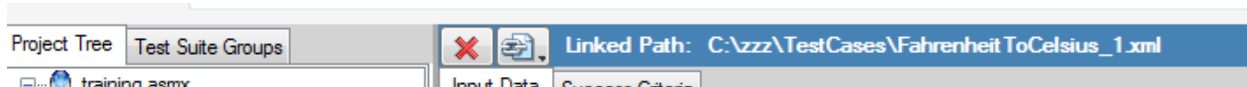
Detach a Test Case as a File Link

Project test cases can be selectively detached from the project and referenced as linked objects to the file path specified. Detached test cases can be linked to an external file by going to the Test Case node and right-click to select the **External Test Case File Link->Detach Test and Reference as File Link** from the context-menu.



Remove a Test Case File Link

You can re-import a test case into the project and remove the link to a file by clicking on the “X” icon that appears next to the Linked Path display above the input data tab.



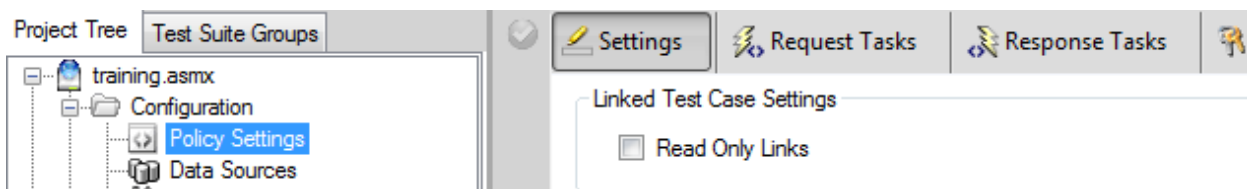
Change a Test Case File Link

You can change the Test Case file link by clicking on the link icon that appears next to the Linked Path display above the input data tab.



Read-Only Test Case File Links

The test case file links can be configured as Read-only to prevent unintentional overwriting of shared data. The read-only setting can be configured on the **Configuration->Policy Settings** screen.



TEST SUITE MANAGEMENT

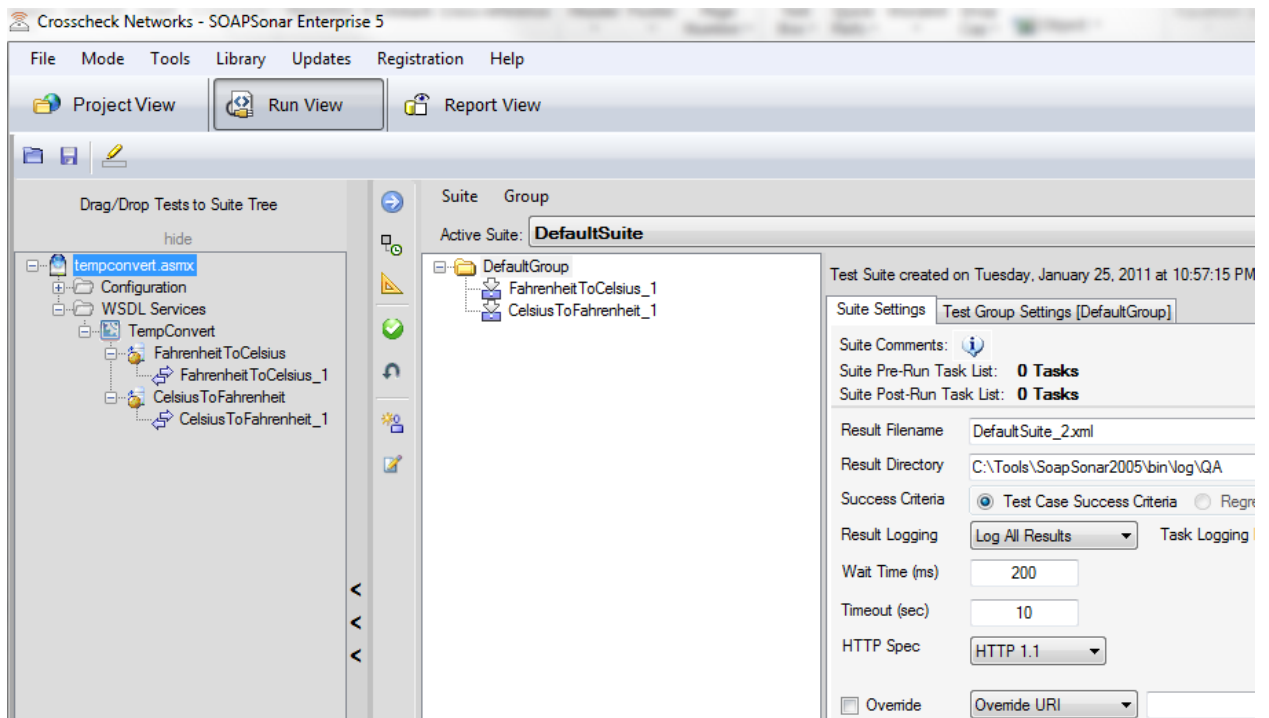
Test cases define the individual unit of information for the test input parameters and properties (inputs, authentication, tasks, success criteria, runtime variables, etc). Test Cases can then be grouped into the logical sequences for execution in the **Run View**. Test Suites are further broken down into Test Group folders which can contain references to the base test cases defined from **Project View**. When in Run View, the Project Tree is visible to the left allowing drag and drop allocation of test cases to the Test Suite Groups.

Test Suites can be run directly within the SOAPSonar interface, or the test suite can be configured to be run from the SOAPSonar command-line, or from within HP Quality Center.

Subtopics in this section include

- [Create a Test Suite](#)
- [Add Test Cases to a Test Suite](#)
- [Conditional Testing](#)
- [Test Suite Properties – QA](#)
- [Test Suite Properties – Performance](#)
- [Test Suite Properties - Compliance](#)
- [Test Suite Properties – Vulnerability](#)

To navigate to the Test Suite Management view, click on the Run View item.



Create a Test Suite

Test sequences can be created by dragging and dropping tests from the Project Tree to the Suite Project Tree. You can add, clone, delete, and add comments to test suites. Each test suite will contain 1 or more groups where the test case sequences are defined. There will always be at least 1 test suite defined.

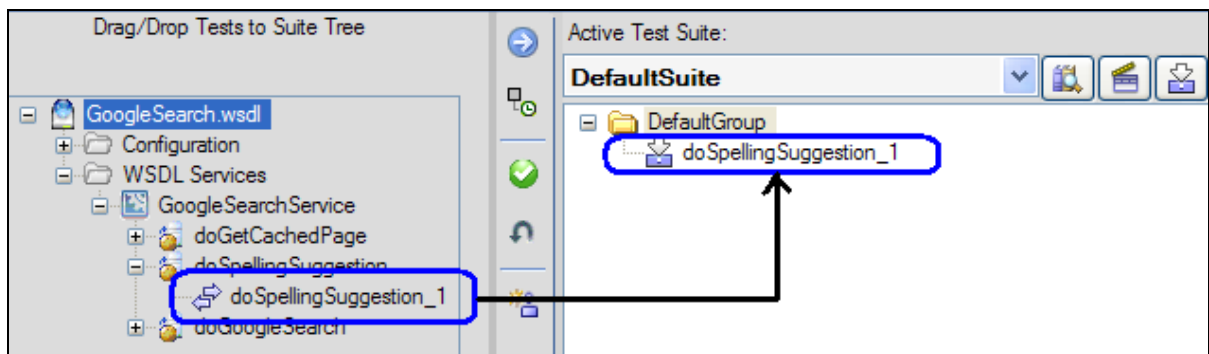
Add Test Cases to a Test Suite

There are a number of ways to add test cases to a test suite in either the Project View or the Run View.

Option #1

In Run View, Add the Test Case(s) to the TestSuite via Drag/Drop

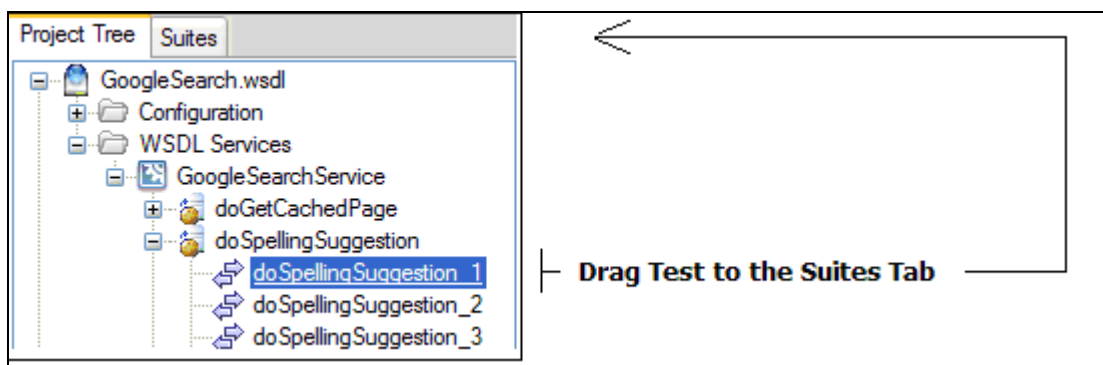
To add a test to a test suite in Run View, select a test case node from the left project tree and drag it to the Suite Tree



Option #2

Add Test Case to Suite via Drag-Drop (Project View)

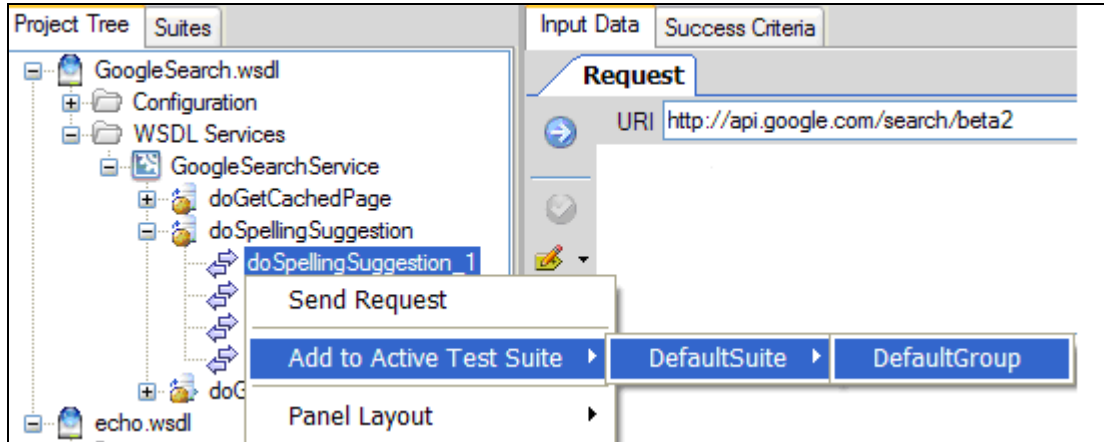
To add a test to a test suite via drag drop in Project View, simply click and hold the test case node and drag it over the Suites tab then drop to the location on the test suite where you want the test case to be run.



Option #3

Add Test Case to Suite via Project Tree Context-Menu (Project View)

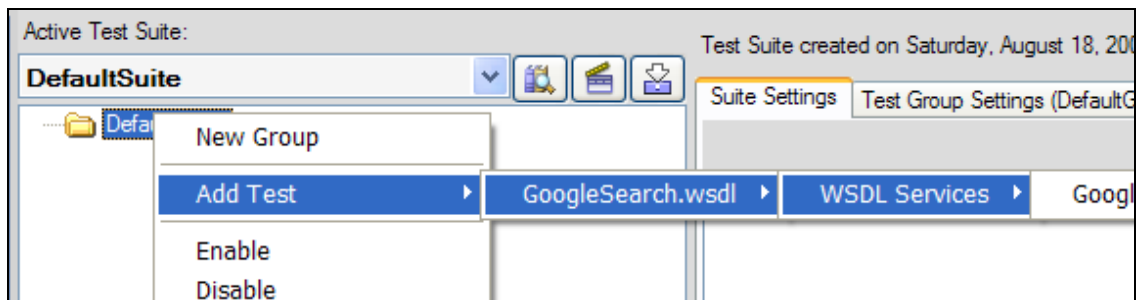
To add a test to a test suite via the context menu, right-click when a test case node is enabled and choose “Add to Active Test Suite” then select the test suite and test group.



Option #4

Add Test Case to Test Suite via Context-Menu

To add a test to a test suite in Run View, right-click on the test suite group and choose “Add Test” from the context-menu options and select the test case to add.



You can disable, clone, copy, and delete test case references across and within test suites. Right-click on the test group or test case node to see available options. Tests can be disabled by selecting the disabled option on the right-click context menu. If you disable a test group, all tests within the group will automatically be disabled. You can disable or enable all tests by right-clicking on the Test Suite name and select “Enable All Tests” or “Disable All Tests”.

Test Suite Pre-Run Tasks

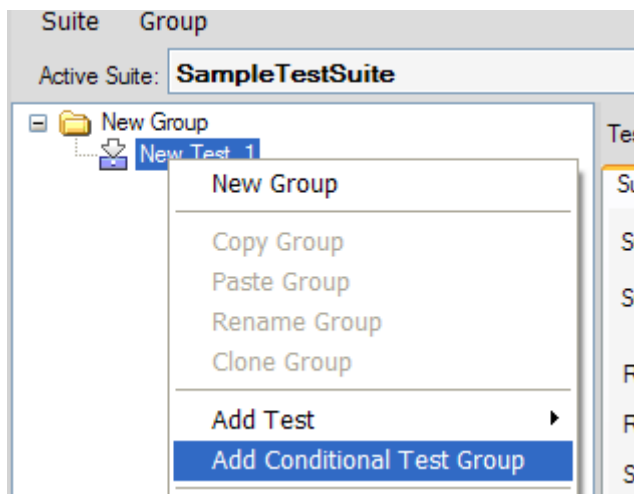
In the Test Suite view, you can set up a task list to run before the test suite runs. Tasks include Database Task, File Manipulation Task, VBScript Task, JScript Task and others. These tasks can be used to set up environment conditions for staging the tests to be run. The scripting tasks enable re-use of existing scripts from applications such as QuickTestPro (QTP) or any other VBScript or JScript framework..

Test Suite Post-Run Tasks

In the Test Suite view, you can set up a task list to run after the test suite runs. Tasks include Database Task, File Manipulation Task, VBScript Task, JScript Task and others. These tasks can be used to publish information, reset environment conditions, etc. The scripting tasks enable re-use of existing scripts from applications such as QuickTestPro (QTP) or any other VBScript or JScript framework..

Conditional Testing

In the Test Suite view, once you have associated a test case with a test suite group, you can create conditional tests under this test case by right-clicking on the test and selecting “Add Conditional Test Group”



A “Conditional Tests” folder will be created under the test node. Drag the tests that are to be evaluated for conditional execution into this new folder. Once the tests are allocated, click on the “Conditional Tests” folder itself to set the conditions for which member tests will execute.

Run View Toolbar Options

The Run View toolbar provides options for managing and running the test suites. Options include running the test, scheduling the test via the windows Task Scheduler, commit settings, rollback settings to previous commit, and if you are in QA mode, a create baseline option is also provided to run the test sequence and store the results as a baseline.



Run Suite

The run option will launch the active test suite and run the enabled tests according to the test suite settings defined.

Create and Schedule Command-Line Script

You can create a command-line script which can be used to run the test suite. Options for the command-line script also include emailing a resulting report and scheduling the script via the Windows Task Scheduler.

Create Baseline Regression Data Set

The create baseline regression data set option will only appear in QA mode and this option will run the entire test sequence and preserve all response data in a regression set that you can then run subsequent iterations of the test suite and choose the "Use Saved Regression Baseline Success Criteria" option to compare the new test run to the stored baseline. Baselines are stored within the test suite and saved in the project file so they can be shared within and across teams. .

Test Suite Properties

Test suite properties are settings such as the result filename, result directory, protocol version, request timeout values, remove URI override, etc. Each run mode (QA, Performance, Compliance, and Vulnerability) expose suite properties which apply to the selected mode. Test suite properties are preserved within the test suite and saved with the project file.

Test Suite Properties: QA Mode

In QA mode, the test suite properties provide settings such as where to store the test results, which protocol version to use, request timeout settings, the active evaluation type (test case criteria, or stored baseline criteria), what information to store in the result logs, and an override URI to redirect the entire test sequence to the specified back-end server. For more information about these settings and running tests in QA mode, please refer to the [QA Mode](#) section of this document.

The screenshot shows the 'Suite Settings' tab. At the top, there are 'Suite Comments' and task lists: 'Suite Pre-Run Task List: 0 Tasks' and 'Suite Post-Run Task List: 0 Tasks'. Below this is the 'Test Suite Settings' section. It includes a 'Result Filename' field set to 'DefaultSuite', a 'Result Directory' field set to 'D:\Tools\SOAPSonar2005\bin\log\QA', and two radio buttons for 'Success Criteria': 'Test Case Success Criteria' (selected) and 'Regression Baseline Rules'. There are also dropdowns for 'Result Logging' (set to 'Log All Results') and 'Task Logging Level' (set to 'INFO'), with a checkbox for 'Use Optimized XML Logging'. Input fields for 'Wait Time (ms)' (200) and 'Timeout (sec)' (10) are present, along with a 'HTTP Spec' dropdown set to 'HTTP 1.1'. At the bottom, there are several checkboxes: 'Override' (unchecked), 'Disable Test Case dependencies' (unchecked), 'Publish Result Summary to CSV File' (unchecked), and 'Publish Results to HP Quality Center Project' (unchecked). An 'Override URI' dropdown is also visible.

The screenshot shows the 'Group Settings' tab. It features a 'Group Definition Mode' dropdown set to 'Manual' and a checkbox for 'Always Show Settings Panels Expanded'. A section titled 'DefaultGroup' is expanded, showing 'Repeat Settings' with a 'Repeat tests' input field set to '1'. The 'Conditional Testing' section includes 'On Success Criteria Result = Fail' and 'Continue Running Tests'. Under 'Result Logging', the checkbox 'Log Results for Tests within this Group' is checked. A 'Comments' link with an information icon is located at the top right of the 'DefaultGroup' section.

Test Suite Properties: Performance Mode

In performance mode, the test suite properties provide settings such as where to store the test results, which protocol version to use, request timeout settings, whether to run the profiles for a duration of time, or a specified set of iterations, how many concurrent loading clients to launch, whether to throttle the requests, and an override URI to redirect the entire test sequence to the specified back-end server. For more information about these settings and running tests in performance mode, please refer to the [Performance Mode](#) section of this document.

The screenshot shows the 'Test Suite Properties' dialog box with the 'Performance Loading Agents' tab selected. The 'Suite Comments' field is empty. The 'Suite Pre-Run Task List' and 'Suite Post-Run Task List' both show '0 Tasks'. Under 'Test Suite Settings', the 'Result Filename' is 'DefaultSuite_31.xml' and the 'Result Directory' is 'D:\Tools\SOAPSonar2005\bin\log\Performance'. The 'HTTP Protocol' is set to 'HTTP 1.1' and the 'Response Timeout (sec)' is '10'. There are several checkboxes: 'Override URI' (unchecked), 'Perform detailed transaction analysis' (unchecked), 'Disable test dependencies' (unchecked), 'Segment data source rows uniquely among virtual clients' (unchecked), and 'Publish results to HP Quality Center project' (unchecked). There are also two dropdown menus: 'Store Errors Only' and 'Write Data to External Files'.

Property	Value
Suite Comments	
Suite Pre-Run Task List	0 Tasks
Suite Post-Run Task List	0 Tasks
Result Filename	DefaultSuite_31.xml
Result Directory	D:\Tools\SOAPSonar2005\bin\log\Performance
HTTP Protocol	HTTP 1.1
Response Timeout (sec)	10
Override URI	
Perform detailed transaction analysis	<input type="checkbox"/>
Store Errors Only	
Write Data to External Files	
Disable test dependencies	<input type="checkbox"/>
Segment data source rows uniquely among virtual clients	<input type="checkbox"/>
Publish results to HP Quality Center project	<input type="checkbox"/>

Test Suite created on Thursday, November 14, 2013 at 1:57:32 PM

Performance Testing Mode Synchronous (Run each Group in Sequence)

Suite Settings **Group Performance Settings** Performance Loading Agents

Group Definition Mode: Manual

Always Show Group Settings Expanded

▼ **DefaultGroup** Comments

Concurrent Loading Clients

Virtual Clients Ramp Strategy Linear Ramp Up (sec) Ramp Down (sec) Think Time (ms) IP (Default)

Interval Setting

Duration Type Duration Test Duration (seconds per test)

SLA Rate Throttling

Throttle Requests Per

▼ **New Group** Comments

Concurrent Loading Clients

Virtual Clients Ramp Strategy Linear Ramp Up (sec) Ramp Down (sec) Think Time (ms) IP (Default)

Interval Setting

Duration Type Duration Test Duration (seconds per test)

SLA Rate Throttling

Throttle Requests Per

Test Suite created on Thursday, November 14, 2013 at 1:57:32 PM

Performance Testing Mode Synchronous (Run each Group in Sequence)

Suite Settings **Group Performance Settings** Performance Loading Agents



Enabled	Agent Display Name	Agent IP Address	Comm Port
<input checked="" type="checkbox"/>	Local Agent		
<input checked="" type="checkbox"/>	MyAgent	<input type="text" value="10.5.6.10"/>	<input type="text" value="9570"/>

Test Suite Properties: Compliance Mode

In compliance mode, the test suite properties provide settings such as where to store the test results, which protocol version to use, request timeout settings, and an override URI to redirect the entire test sequence to the specified back-end server. For more information about these settings and running tests in compliance mode, please refer to the [Compliance Mode](#) section of this document.

The screenshot shows the 'Suite Settings' dialog box. At the top, there is a 'Test Suite Comments' text area. Below it, the 'Result Filename' is set to 'DefaultSuite.xml'. The 'Result Directory' is set to 'C:\Tools\SoapSonar2005\bin\log\Compliance'. The 'Protocol' is set to 'HTTP 1.1', 'Response Timeout (sec)' is '10', and 'Wait Time (ms)' is '200'. Under 'Success Criteria', the radio button for 'Require SOAP Faults and HTTP Error Codes' is selected. The 'Override URI' checkbox is unchecked.

Field	Value
Result Filename	DefaultSuite.xml
Result Directory	C:\Tools\SoapSonar2005\bin\log\Compliance
Protocol	HTTP 1.1
Response Timeout (sec)	10
Wait Time (ms)	200
Success Criteria	Require SOAP Faults and HTTP Error Codes
Override URI	Unchecked

Test Suite Properties: Vulnerability Mode

In vulnerability mode, the test suite properties provide settings such as where to store the test results, which protocol version to use, request timeout settings, and an override URI to redirect the entire test sequence to the specified back-end server. For more information about these settings and running tests in compliance mode, please refer to the [Vulnerability Mode](#) section of this document.

The screenshot shows the 'Suite Settings' dialog box. At the top, there is a tab labeled 'Suite Settings'. Below the tab is a section for 'Test Suite Comments' with a large empty text area. The main settings area includes: 'Result Filename' with a text box containing 'DefaultSuite' and a '.xml' suffix; 'Result Directory' with a text box containing 'C:\Tools\SoapSonar2005\bin\log\Vulnerability' and a folder icon; 'Protocol' with a dropdown menu set to 'HTTP 1.1'; 'Response Timeout (sec)' with a text box containing '10'; 'Wait Time (ms)' with a text box containing '200'; and an 'Override URI' checkbox which is unchecked, with an empty text box below it.

PROJECT MANAGEMENT

All the information configured within Project View and Run View can be stored as a project file to be used to share among teams, version using source control, or used by the command-line interface to run test suites.

Subtopics in this section include

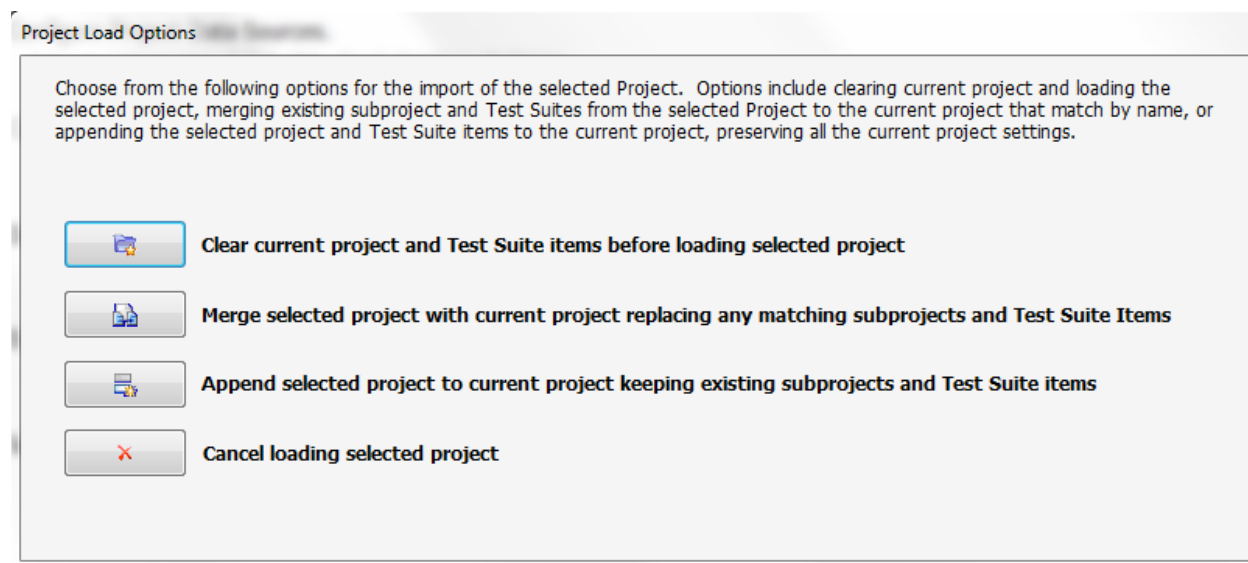
- [Import Project](#)
- [Merge Project](#)
- [Append Project](#)
- [Save Project](#)
- [Upgrade WSDL in Project](#)

Project files also preserve any regression baselines defined, such that the project files can be shared among teams to provide simply sharing of test sequences with expected results. For example, in one environment a sequence is created which induces a bug. This project is then shared with another team to show how to reproduce the problem. Another example is the developer responsible for a product feature can build the unit tests and then store and share the project file for the test team to use as a starting point.

Project management includes the features of loading, saving, exporting, and import project and WSDL files. Project files can be merged into the current project simply by loading an existing project while working on a project. You will be prompted to indicate whether you want to merge the incoming project with the existing projects, or clear existing settings before bringing in the selected project.

Import a Project

When loading a previously saved project, the option presented will depend on whether you currently have existing subproject loaded. If you already have items loaded, you will be presented with several options on how you would like to load the project. These include:



Load as new Project

To load a new project, select the **“Create current project and Test Suite items”** option. This option will remove all settings and start with the loaded project only

Merge Project

To merge the selected project, select the **“Merge selected project with current project”** option. This option will search for all existing subproject names that match those in the selected project and then merge all settings in the selected project with the settings currently loaded. This is useful when sharing projects and merging updates to the same project.

Append Project

To merge the selected project, select the **“Append selected project to current project”** option. This option will simply append all the selected subproject names to the end of the currently loaded ones. This setting is useful for combining projects into master projects.

Save a Project

At any time you can preserve the current settings into a project file. Options include the ability to save all tests and test suites, or selectively save a single project nodes and related settings.

Save Full Project

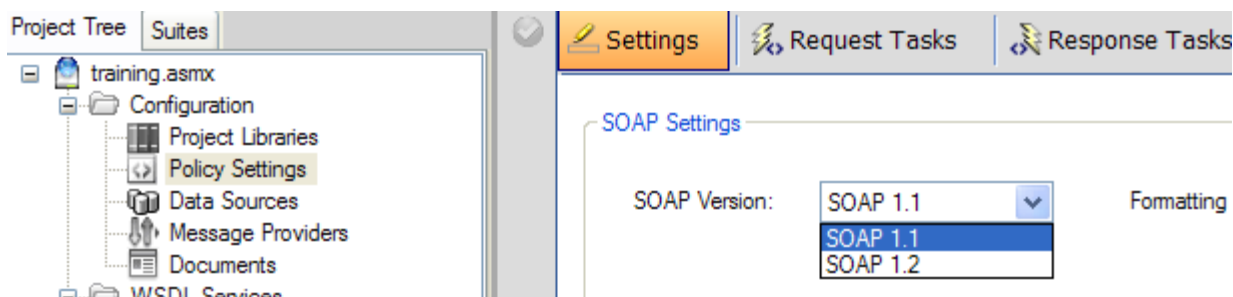
Choose File->Save Project to create a project file from the current settings.

Export Project

Individual projects can be exported by right-clicking on the subproject node in the project tree and selecting the **Export to File** option.

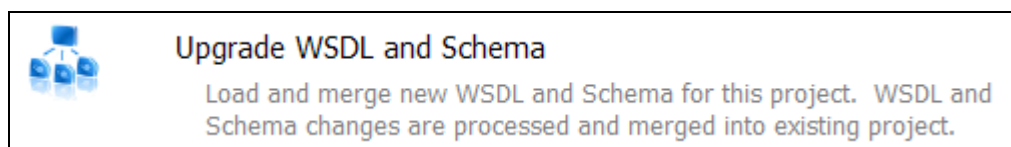
WSDL Project SOAP Version

For WSDL based SOAP testing, the WSDL projects contain a Policy Settings tab under the Configuration area. On this screen, the SOAP Version can be changed from SOAP 1.1 to SOAP 1.2 and vice-versa. When changing the SOAP version on this screen, it will have the effect of updating all existing and future test cases to use the selected SOAP version. You can also set the default behavior of which SOAP version SOAPSonar should use from the File->Settings and Preferences menu.



Upgrade a WSDL

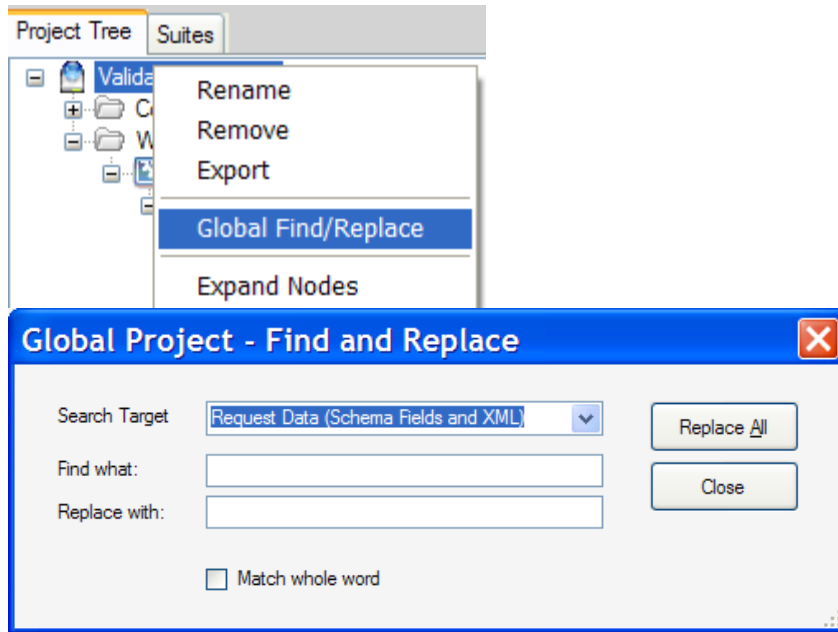
If you have a new version of a web service which has an updated WSDL document, SOAPSonar provides sophisticated WSDL merging capability which allows you to preserve existing project and test case settings when refreshing the WSDL in the project with later revision of the WSDL file. To update an existing WSDL in the project with a new revision, click on the “Merge WSDL” icon on the toolbar and then choose the location to obtain the new WSDL.



In the event that you load a WSDL that already exists in the project, you will automatically be prompted with options to load this WSDL as a separate and new instance to this project, or merge the WSDL with the existing WSDL. Once you choose to merge the WSDL, SOAPSonar will capture the schema differences between the 2 versions and automatically adjust your current test cases with the applicable SOAP schema structure and type changes. Existing test case and WSDL enrichment settings will be preserved.

Global Find/Replace Options

You can right-click on the WSDL project node for the global find/replace option. This option will allow you to search and replace data across the entire WSDL project.



These options are meant for replacing data values within the selected search target. If you want to replace other project data, including custom replacements of XML element names, or other type of arbitrary global replacements, please see instructions in the next topic regarding accessing the SOAPSonar project file as raw XML.

Convert SSP Project to Raw XML

By default ssp project files are stored as compressed XML. It is a simple process to convert the project file to raw XML for direct manipulation (i.e. global find/replace).

Follow the steps below to access convert an .ssp project to a raw .xml file:

- a) Add a .gz extension to the .ssp file
- b) Expand the file using a gunzip utility
- c) Rename the .ssp file to .xml (or .txt)
- d) Modify the file with an XML or text editor.

Be cautious not to corrupt your project with inadvertent replacements - it is good practice when replacing xml to use the "<" and ">" in the replace text to ensure only the selective values are replaced, otherwise project corruption could occur.

e) Rename the .xml back to .ssp - the project will be automatically compressed once it is saved within SOAPSonar again

RUN MODES

SOAPSonar provides 4 run modes which are designed to provide features specific to each of the 4 pillars of web services diagnostics testing. Each mode provides reports and diagnostics capabilities that map to tasks and result analysis specific to each of these 4 test perspectives.

Authoring tests is a common task and interface across all run modes. Switching run modes changes the run-time behavior of SOAPSonar, or more specifically, changes the types of testing that occurs when you run your test suites. For example, in QA mode, each request and response is analyzed against defined success criteria to provide Pass/Fail reports on the functional validation test suites. These same tests can also be run in performance, compliance, and vulnerability mode, each mode altering the reports which are available and the type of diagnostics taking place. A test suite run in performance mode will result in analysis throughput profile statistics options for reporting and logging that isolate these characteristic of the test run per the selected mode.

QA Run Mode

This mode is used to create test inputs and responses, both simple request/response and more complex test sequences to validate functional requirement adherence based on how the back-end server is responding to the requests. Defining success criteria in this mode provides the ability to automatically determine and report success and failures for each test.

Performance Run Mode

This mode is used to validate throughput and response time profiles for your back-end server when presented with simple request/response tests, or more complex test sequences. In this mode, you can allocate concurrent clients to simultaneously load and calculate statistics. When running tests in performance mode, each loading client has it's own connection and memory space to independently load and gather statistics. These statistics are then consolidated and displayed in the logging and reporting interface.

Compliance Run Mode

This mode is used to validate the active run-time SOAP compliance posture of a back-end web service. Using patent-pending XSD-Mutation technology, this run mode will automatically mutate your base test case request to specifically alter the message in order to isolate each requirement for SOAP messaging in the WS-I Basic Profile 1.1 Section 3 SOAP Messaging specification. These requirements call for proper SOAP specification adherence as well as best practices for SOAP message handling that help to ensure the highest level of interoperability among connecting clients. Using XSD-Mutation technology, SOAPSonar will automatically break each assertion rule for the SOAP messages by transforming the base message itself and then analyze the response of the web service to see whether it is handling the input in a WS-I compliant manner. The lower the number of compliance violations detected, the higher level of robustness and interoperability compliance posture your web service has.

Vulnerability Run Mode

This mode is used to verify the error handling robustness and risk posture of a web service. Using patent-pending XSD-Mutation technology, SOAPSonar will iteratively transform your base test message to isolate attack characteristics that are dynamically determined from the WSDL Schema. This patent-pending technology allows SOAPSonar to build the attack profiles dynamically for each web service leveraging the known inputs and parameters supplied by the WSDL. Attack vectors such as parameters, structure, occurrence, specification breakage, type injection, are examples of auto-generated dynamic vectors. This mode provides automatic code boundary API error robustness validation by isolating characteristics of each SOAP message that inject the attack deeper within the application layer of the web service to exercise the code path error handling logic.

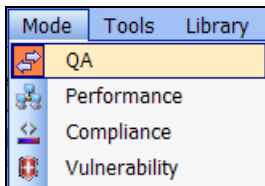
QA MODE

QA mode is used for functional validation of your back-end service, including positive and negative testing. In QA mode the test request information as well as the expected response success criteria is configured to allow repeated testing and result analysis. In QA mode you can perform simple request response message scenarios as well as more complex test case chaining multiple service invocations. In addition to content based message testing, the Identity Management and Policy tasks are also available to validate authentication and authorization requirements and security provisions.

Subtopics in this section include:

- [QA Test Cases](#)
- [Success Criteria Rules](#)
- [Baseline Regression Testing](#)

In QA mode test sequences are combined in test suites which are run to produce pass/fail results. Result analysis is configured using the various success criteria evaluation rules. To switch to QA mode, go to the Mode menu and choose QA.



Functional tests may include scenarios such as:

- Request/response validation with static input
- Request/response validation with dynamic input (Data Driven)
- Request/response sequences with dependent inputs (from other test responses)
- Authentication and authorization verification
- Enrichment features such as WS-Header tokens, WS-Security, etc.
- SOAP with Attachment message processing
- Baseline regression testing
- Automated command-line testing

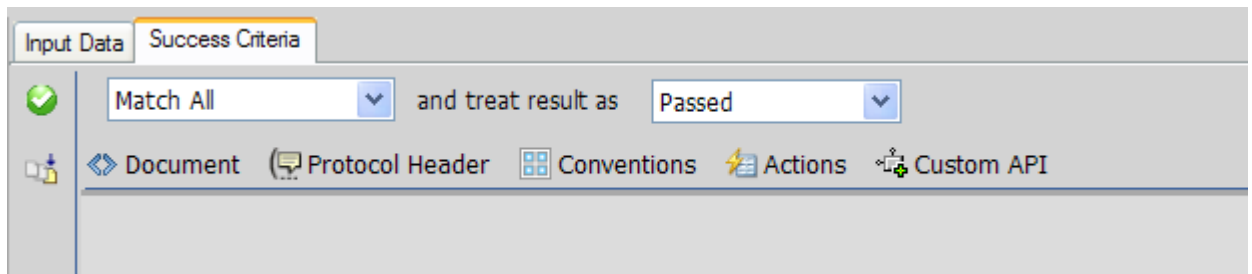
QA Test Cases

Tests are created and managed the same across each test mode. This allows the same test cases and message paradigms to be used across each run mode. Refer to the [Building Tests](#) section for instructions how to create and manage test cases.

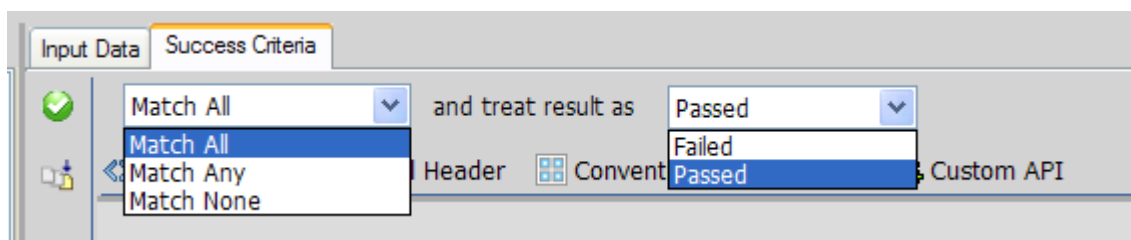
QA mode test cases use the test configuration with the defined success criteria to create pass/fail results for the test suite run.

Success Criteria Rules

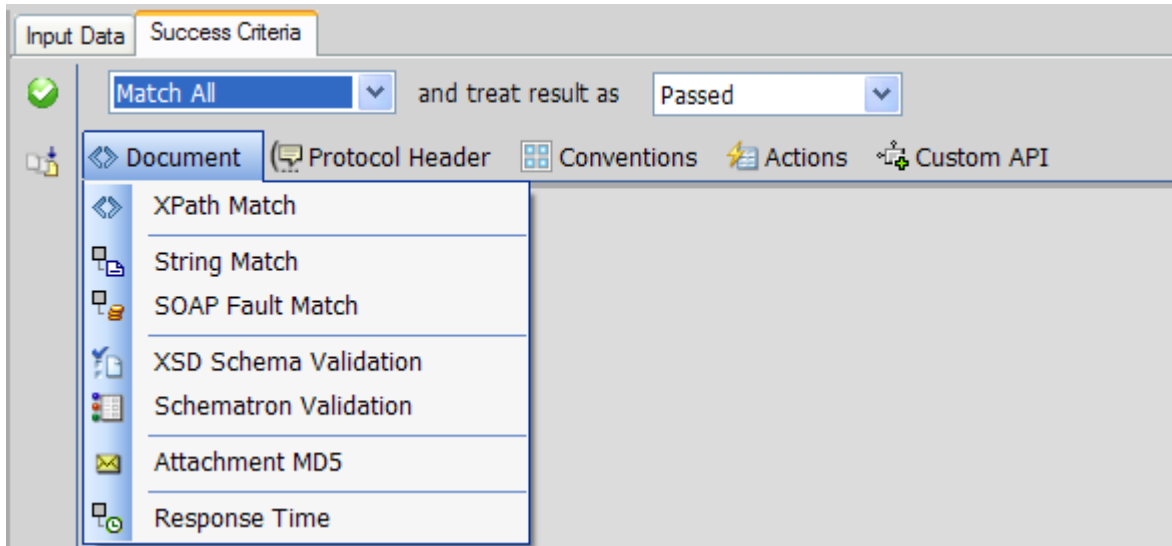
The SOAPSonar Success Criteria evaluation engine provides a full range evaluation functions that allow you to define the analysis criteria for the response to automatically determine whether the API response is a Pass or a Fail. In QA mode this criteria is used to report whether the test step was a Pass or Fail. In Performance mode the criteria is used to determine whether to trigger error tracing (to preserve the request and response that triggered the response analysis).



You can define as many criteria items as you want to accomplish a range of evaluation results. You can choose how you want to apply the set of criteria function in terms of matching all the items, matching any one of the defined items, or matching none of the items. The result of the matching criteria is then treated as configured to be a Pass or a Fail result.



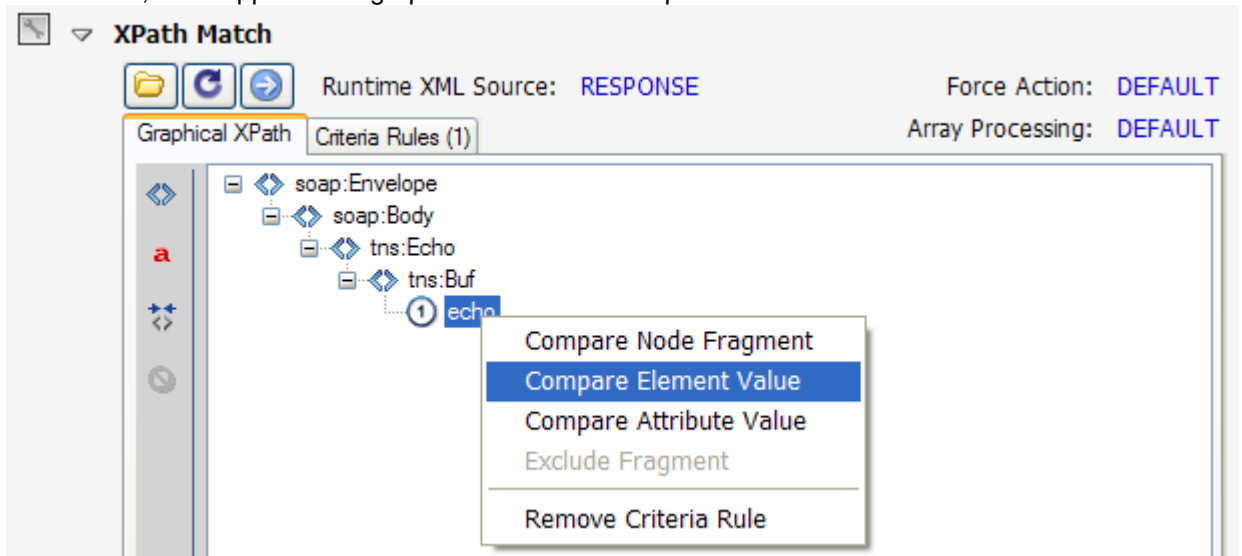
To add new criteria, click on the menu category items shown to create each type of criteria rule.



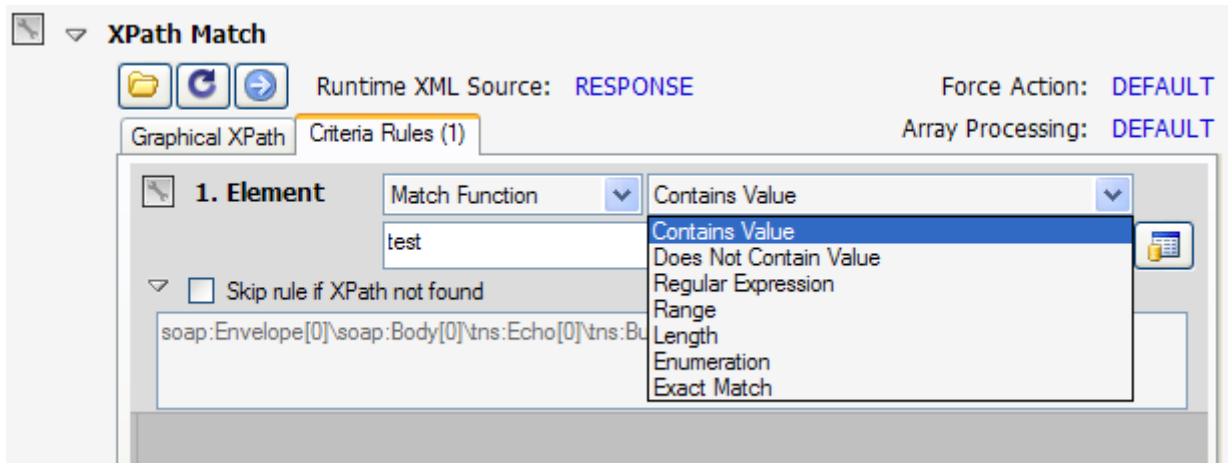
Value matching can be a substring match, a substring non-match, a regular expression pattern match, a numeric range, a string length range, or an enumeration of acceptable values. Additionally, each criteria specified can be set with a Force Action to force the result as a Pass or Fail.

Success Criteria Rules: XPath Match

The XPath Match rule gives a graphical representation of the expected XML or JSON and allows configuration of criteria rules to isolate parts of the target document to compare against. Sample documents can be loaded from file, or taken from the most recent test case response, or you can send the current test case request to the back-end service to obtain a sample response. Once the sample document is loaded, it will appear as a graphical tree on the Graphical XPath tab as shown below.



You can then right-click on XML/JSON Nodes, XML/JSON Element Values, or XML/JSON Attribute values to create a new XPath criteria rule to analyze and evaluate that target value. On the Criteria Rules tab, the target XPath value that was selected will show up as a path query in the bottom text box. The rules that can be set up for analyzing the data depend on the type of data selected. The match function are discussed in detail in the sections below.



To configure the XPath Match Success Criteria, select Add->XPath Match. This will create a new Success Criteria rule with a Tree view representation of the current test case response. You can use the button on the top to invoke the web service to obtain the latest response document, or load a document from file. With the graphical representation of the request showing, you can choose one or more node, element value, or attribute value to create a rule comparison against. To select portions of the document to be evaluated, click on the node, element, or attribute value and then right-click, or select the active button on the left toolbar to create an XPath criteria rule for the selected value.

Note that JSON data is converted to XML structure in memory to allow the XML-based criteria functions to work for both XML and JSON data.

Once an XPath criteria rule is created, the types of evaluation matching available are:

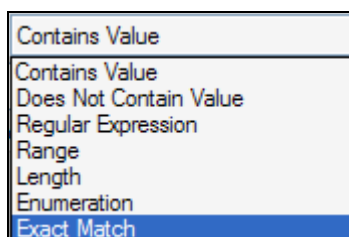
Match Baseline

Compare the live test response to the stored test case expected document at the XPath locations specified.

Match Function

Define the criteria to be used to evaluate the portions of the document selected. This criteria can be explicitly noted, or this criteria can be dynamically extracted from a database or Runtime Variable (when the [Automation Module license component](#) is enabled).

When match function is selected, the evaluation criteria options are:



Contains Value – This will search the referenced response data for the substring value specified

Does Not Contain Value – This will search the referenced response data for no instances of the substring value specified

Regular Expression – A RegEx pattern match criteria used to match the referenced response data. Acceptable format for expression is any valid pattern match regular expression.

Range – A minimum numeric and maximum numeric value entered with a “|” delimiter as follows: MIN|MAX. For example, the value **23.4|23.4** would match the number 23.4, 23.40, and 23.400. The value **0|100** would match any number between 0 and 100.

Length – A minimum string length and a maximum string length entered with a “|” delimiter as follows: MIN|MAX. For example, the value **5|5** would require a string of exact length 5. A value of **0|100** would match if the referenced data was between 0 and 100 characters.

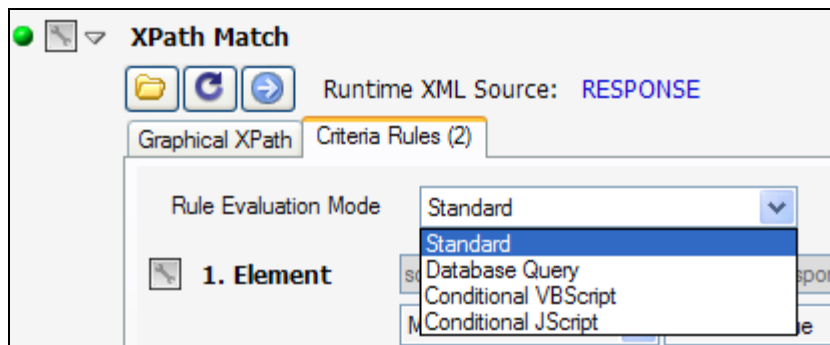
Enumeration – A set of acceptable data values entered with a “|” delimiter as follows: VALUE1|VALUE2|VALUE3|...|VALUEN. For example, the value **10.0|undef|none** would allow the referenced data value to be exactly “10.0”, “undef”, or “none”.

Exact Match – This will compare the referenced response data for the exact value specified

Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate popup window.

XPath Success Criteria Rule Evaluation Modes

The XPath Success criteria rule provides several evaluation modes as to how to evaluation and diagnose the detected criteria.



Standard – Uses the rule functions such as substring match, regular expression, etc in order to compare the values at runtime to those configured.

Conditional VBScript – Allows creating of “nickname” variables for each XPath expression that are available within your own VBScript scripting function to be able to create conditional logic and complex expressions.

Conditional JScript – Allows creating of “nickname” variables for each XPath expression that are available within your own JScript scripting function to be able to create conditional logic and complex expressions.

Database Query – Allows creating of SQL queries that map to the element or attribute names targeted by the XPath and then directly querying a database to validate the data.

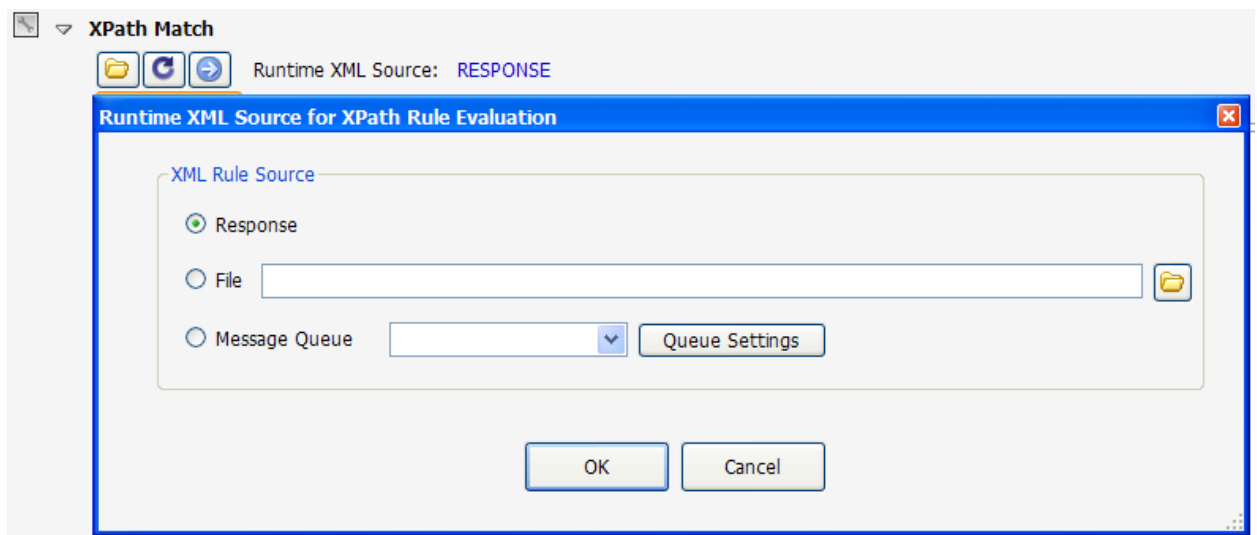
Using XPath Match Rule To Validate File and JMS, MQ, WLS, and EMS Message Queues

By default, the XPath match rule will use the document that was received in response to the sent request. However, the XPath Match rule can also be used to verify information in a target file, or on a target message queue (JMS, Weblogic JMS, IBM MQ, and Tibco EMS).

Using the “Runtime XML Source” setting, the document target can set to use:

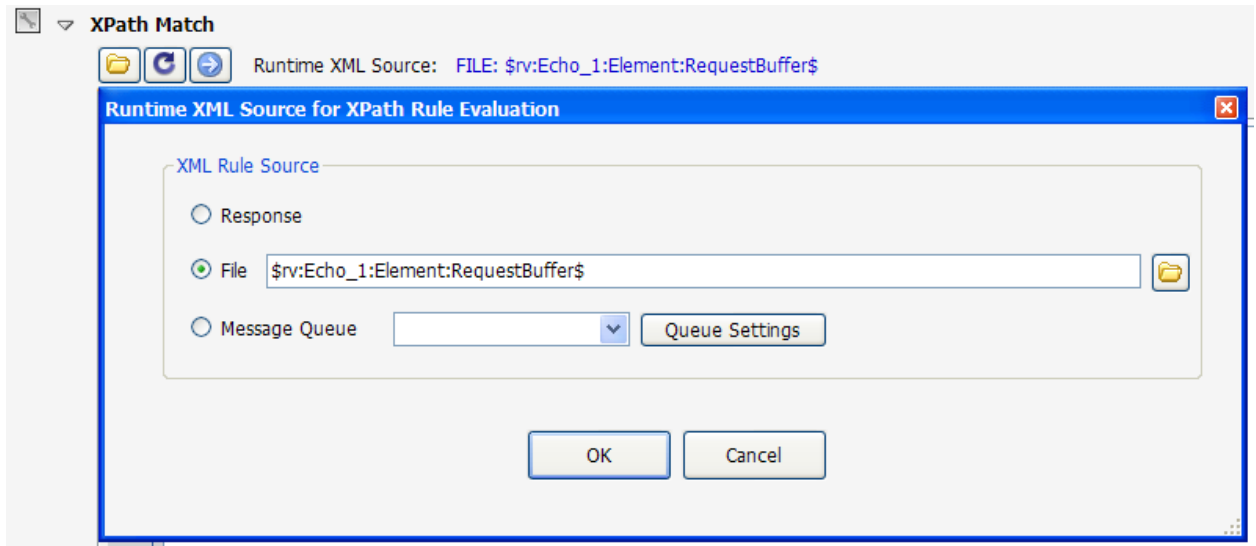
XPath Runtime XML Source: Response

Uses the active response as the target of the XML/JSON to evaluate against the XPath rules. This is the default setting.



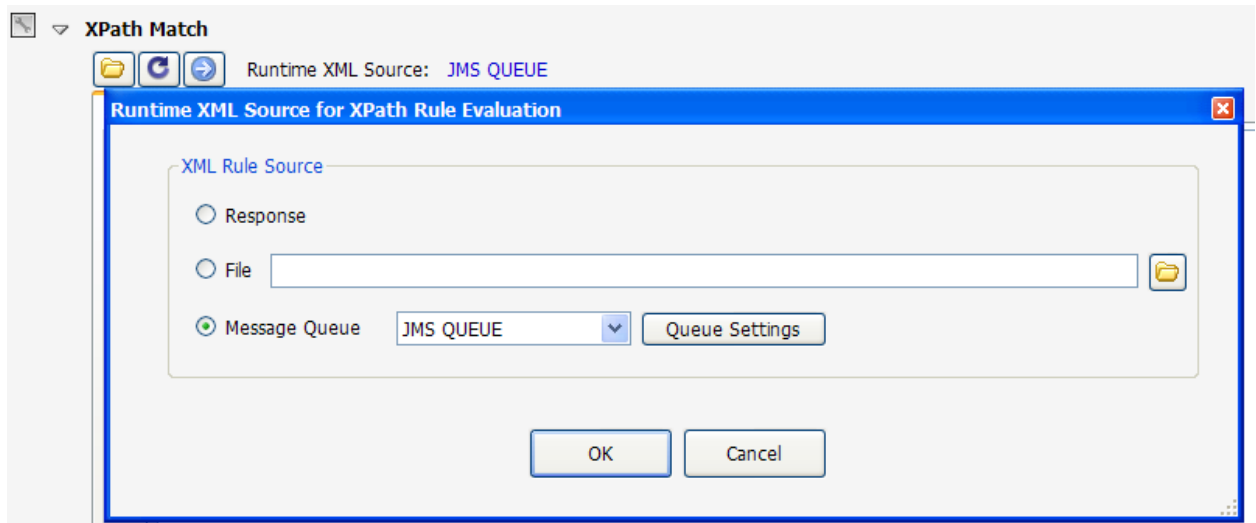
XPath Runtime XML/JSON Source: FILE

Sets the target document as a file reference. The full path value for the filename can be set as a static file, or can use dynamic variables such that the target filename is dynamically resolved at runtime. At runtime, the target file contents are obtained and used as the source document to validate against the specified XPath rules.



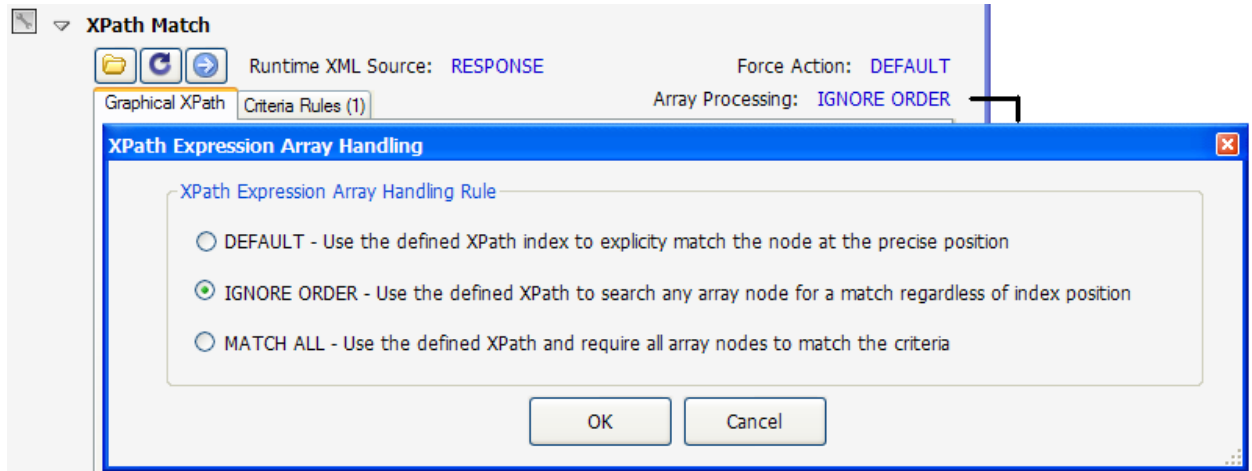
XPath Runtime XML/JSON Source: MESSAGE QUEUE (JMS, MQ, WLS, EMS)

Sets the target document for the Xpath analysis rules to be extracted from a JMS, Weblogic JMS, IBM MQ, or Tibco EMS message queue. The queue settings define the access configuration for connecting and retrieving the message from the queue. The extracted message is then used as the source for the XPath rules.



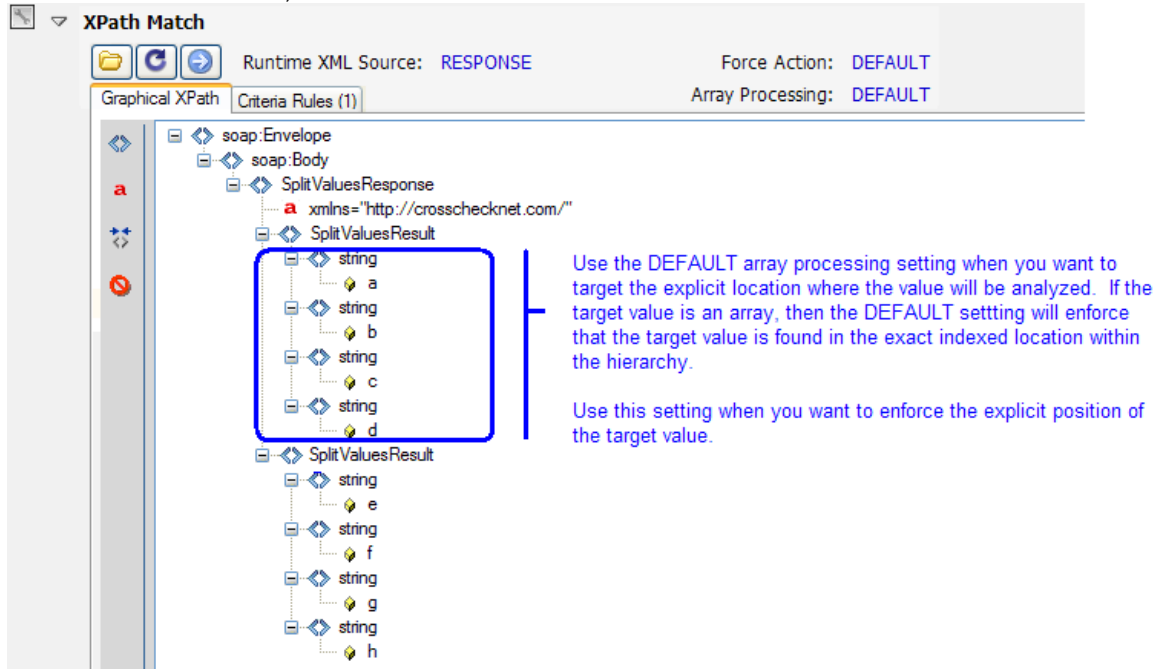
Success Criteria Rules: XPath Array Handling Options

The XPath match rules provides array handling options to provide flexible analysis when dealing with validating array or collection values. The 3 settings of Default, Ignore Order, and Match All are explained in detail in the sections below.



XPath Array Handling: DEFAULT

The default array handling setting will enforce that the XPath expression match precisely the indexed location of the value within the XML/JSON. In other words, if the Array value is found in the second collection set instead of the first, it will not match.



XPath Array Handling: IGNORE ORDER

When testing scenarios where the array data may change locations within the XML/JSON document you may want to ignore the order in which the elements appear as long as the target criteria in question appears somewhere within the array collection. For this the Ignore Order setting can be used. The

Ignore Order setting will allow the XPath expression to search through all elements at that hierarchy level, regardless of order, to evaluate the criteria rule.

The screenshot shows the 'XPath Match' configuration window. At the top, it indicates 'Runtime XML Source: RESPONSE' and 'Force Action: DEFAULT'. The 'Array Processing' dropdown is set to 'IGNORE ORDER'. The tree view shows a SOAP envelope with a body containing a 'SplitValuesResponse' element. This element has an attribute 'xmlns="http://crosschecknet.com/"' and contains two 'SplitValuesResult' elements. Each 'SplitValuesResult' contains an array of five 'string' elements. The first array contains values 'a', 'b', 'c', and 'd', while the second array contains 'e', 'f', 'g', and 'h'. Blue boxes highlight these string arrays. To the right, text explains: 'Use the IGNORE ORDER rule when you want to target a single match of the criteria rule, but do not care where the value appears. If the values are not returned in a consistent order within the array, the IGNORE ORDER setting will look through all values for a single match of the criteria rule.' Below this, it says: 'Use this setting when the XPath locations may change from one test to another.'

XPath Array Handling: MATCH ALL

When testing scenarios where the array data must contain certain data in all instances of the array, use the Match All setting. The Match All setting will enforce that the target XML rule applies to all XML/JSON values found at that hierarchical level. So, if one document has 1 array instance, and another has 10 array instances, this rule would enforce that the 1 value match in the first case, and all 10 values match in the second case.

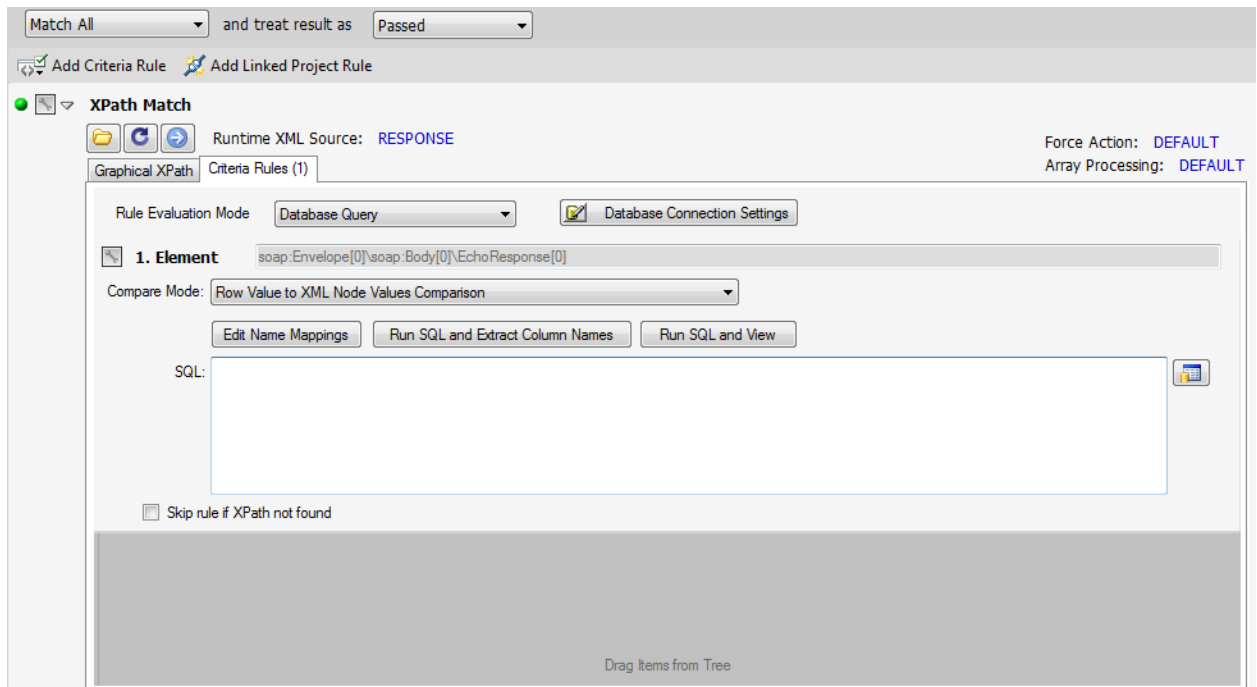
This screenshot is similar to the previous one, but the 'Array Processing' dropdown is set to 'MATCH ALL'. The tree view and XML structure are identical. The explanatory text on the right states: 'When setting the Array Processing setting to MATCH ALL, each value at the target location will be analyzed for the criteria rule match. Use this setting if you have dynamic arrays which may have multiple sets of values at the target Array.' Below this, it adds a note: 'Note: This setting can also be used in conjunction with the "Skip rule if Xpath is not found" to allow 0 length array values.'

Success Criteria Rules: XPath Compare Node Fragment Array to Database Query

To compare XML data against database tables, the XPath Match function has a setting that enables direct comparison of XML node structures and elements to a database query.

To create this rule, select **Document->XPath Match**, then right click on a node and select “**Compare Node Fragment Array with Database Query**”

This success criteria rule provides name mappings to map column names in the database to the element names, if they differ.



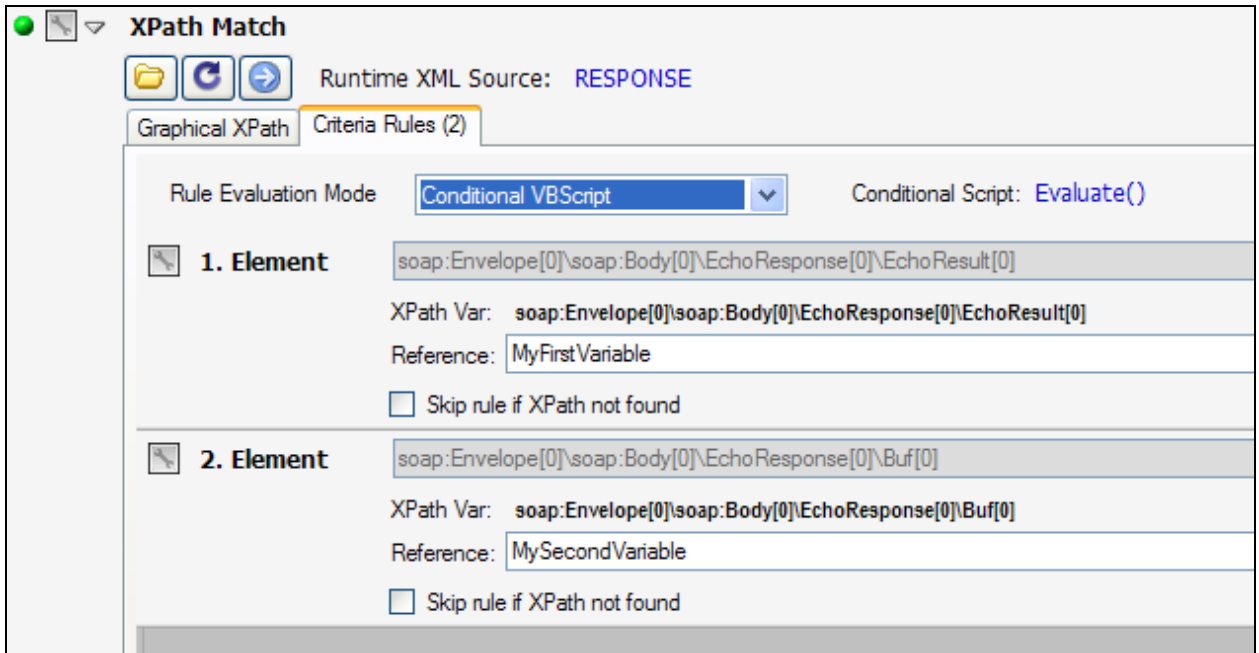
The SQL query can be parameterized using variables.

The Database Connection Settings button enables configuration of the database target. For connection strings to use for access to any ODBC compliant database, visit www.connectionstrings.com

Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate popup window.

XPath Rule Evaluation Mode - Conditional Analysis Scripting Rules

The XPath success criteria rule Conditional Script mode allows analysis in the context of your own custom VBScript or JScript. When this mode is enabled, the XPath expression criteria rules are presented with the variable names and a custom “Reference” name that you can provide that can be referenced from within the script code.



When clicking on the Conditional Script link, the script editor will be presented where you can define your own custom logic. The script will have available the Parameters global object which provides information regarding the current test case, as well as variables.

The screenshot shows a 'Script Editor' window with a menu bar containing 'File' and 'Parameters'. Below the menu is a toolbar with various icons. The main area contains the following VBA code:

```

1 Sub Evaluate (unused)
2
3     'Sample evaluation code below.  Replace with your own logic.
4     'show the evaluation information accordingly
5     If( Parameters.GetVariableValue("$MyFirstVariable$") = "test"
6         'Return Values - Set values for result - these values sho
7         Parameters.MatchString = "Data content successfully analy
8         Parameters.MatchContext = "Full Document"
9         Parameters.MatchExpression = "Substring Match"
10        Parameters.MatchCriteria = "Found Substring"
11        'Report back success or failure for the evaluation of thi
12        Parameters.SuccessValue = True
13    Else
14        'Return Values - Set values for result - these values sho
15        Parameters.MatchString = "Data content analysis failed"
16        Parameters.MatchContext = "Full Document"
17        Parameters.MatchExpression = "Substring Match"
18        Parameters.MatchCriteria = "Did Not Find Substring"
19        'Report back success or failure for the evaluation of thi
20        Parameters.SuccessValue = False
21    End if
22
23 End Sub

```

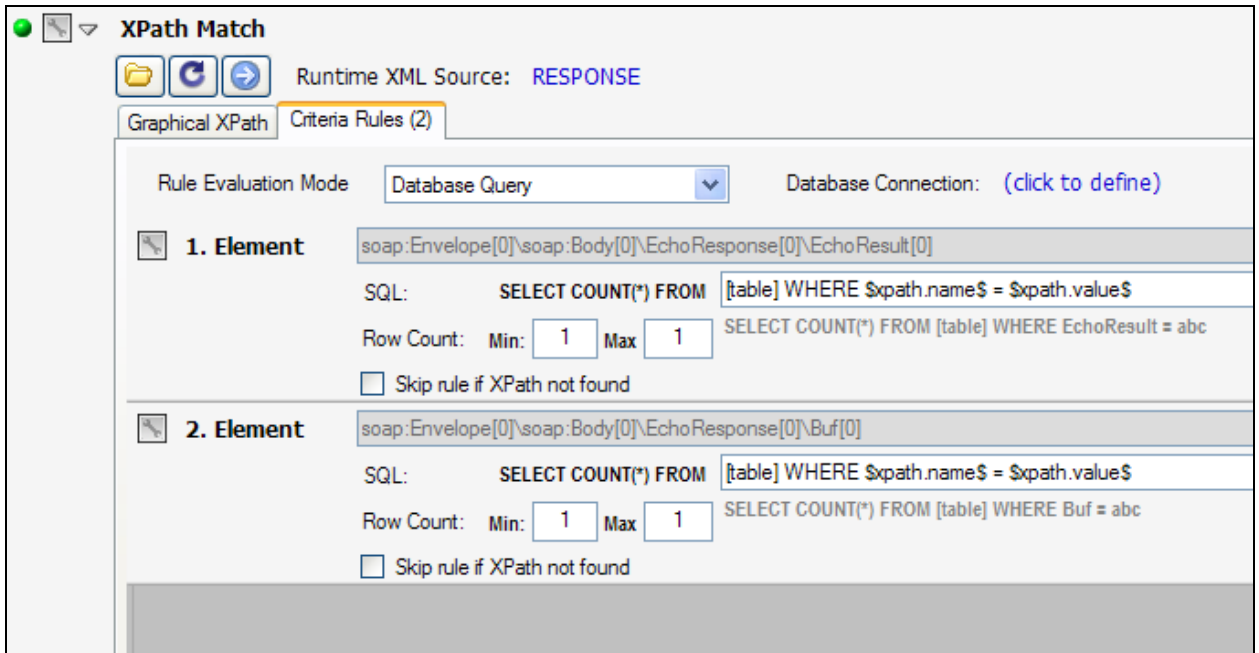
Using XPath Database Matching Rules

The XPath success criteria rule Database mode allows running a SQL query using the element or attribute target name and value to be used within the SQL query to validate the data against a data table. The variables enabled in this evaluation mode include the standard variables, and 2 special variables:

- \$xpath.name\$ -- this variable will resolve to the target attribute or element name resolved at runtime
- \$xpath.value\$ -- this variable will resolve to the target value resolved at runtime

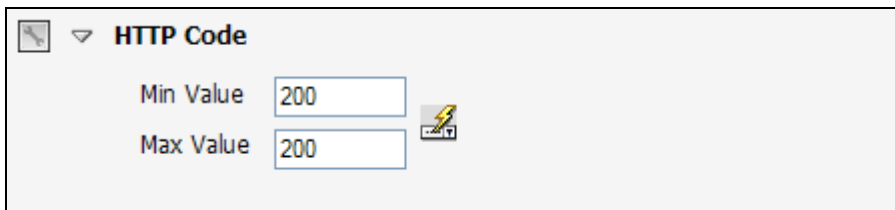
These variables can be used in the context of the SQL query so that the actual query resolves to the dynamic value and name at runtime.

The SQL statements are executed as count(*) statements that will return a row count from 0 to N rows. The validation is then checked against the Min and Max row value to determine if the rule is a success. To set the number of rows to a set value instead of a range, simply set them both to the same value (i.e. Min=1, Max=1).



Success Criteria Rules: HTTP Code Range

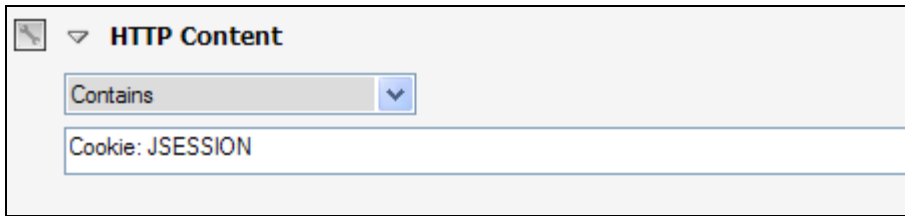
The SOAP specification provides known values for what the HTTP code should be in the event that the request was considered a request or a failure. You can use the HTTP Header response code as evaluation criteria to assess whether the response code falls within the range specified. A helper icon is provided to set the values for the SOAP success response (200), or the SOAP Fault response (400-599). The HTTP Code Success Criteria specifies the expected range that the HTTP response should fall between.



Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate pop-up window.

Success Criteria Rules: HTTP Header String Match

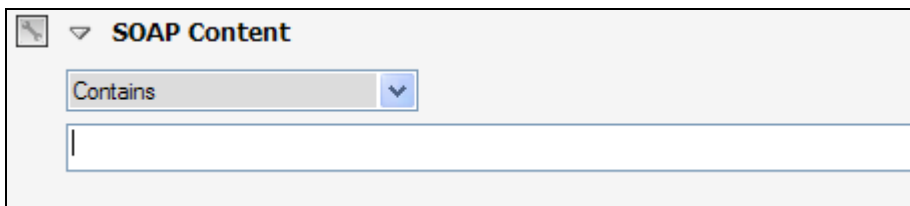
The HTTP Header criteria will evaluate the expression provided according to the match type specified. Contains will search for the presence of the string value in the HTTP response header. Does Not Contain will ensure that the string pattern does not occur in the HTTP response header, and Regular Expression allows you to specify a regex pattern match function to use to determine whether the regex pattern is found in the HTTP response header. The value itself can be a static function, or it can be a dynamic value extracted from a database table or Runtime Variable (with the [APC license component](#) enabled).



Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate popup window.

Success Criteria Rules: String Match

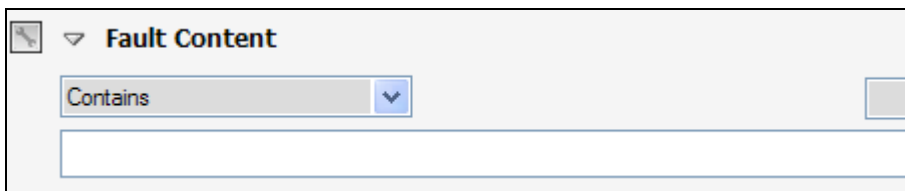
The Match String criteria will evaluate the expression provided according to the match type specified. Contains will search for the presence of the string value in the message data. Does Not Contain will ensure that the string pattern does not occur in the data, and Regular Expression allows you to specify a regex pattern match function to use to determine whether the regex pattern is found in the message. The value itself can be a static function, or it can be a dynamic value extracted from a database table or Runtime Variable.



Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate popup window.

Success Criteria Rules: SOAP Fault Match

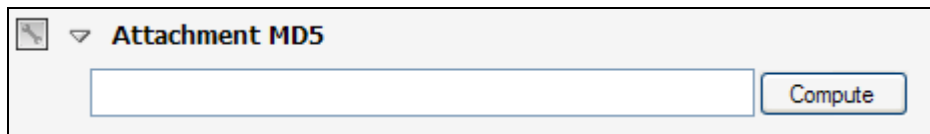
The SOAP Fault match criteria will evaluate the response to determine whether a SOAP fault is detected. The rule can be further extended by providing additional criteria for the contents of the SOAP fault. If the criteria is left blank, the rule will simply evaluate for the existence of a SOAP fault, regardless of the contents and details of the fault.



Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate popup window.

Success Criteria Rules: Attachment MD5

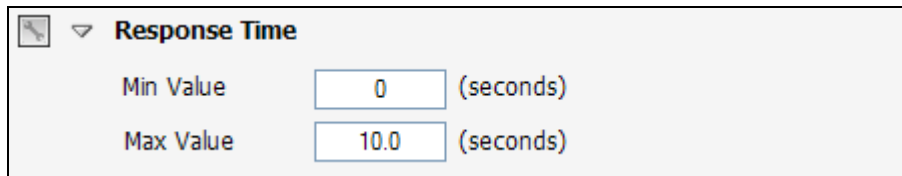
The Attachment MD5 criteria will evaluate the response attachment md5sum. This is useful when running tests which return a MIME or DIME attachment to ensure that the attachment being returned is consistently the same md5sum value.



Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate popup window.

Success Criteria Rules: Response Time

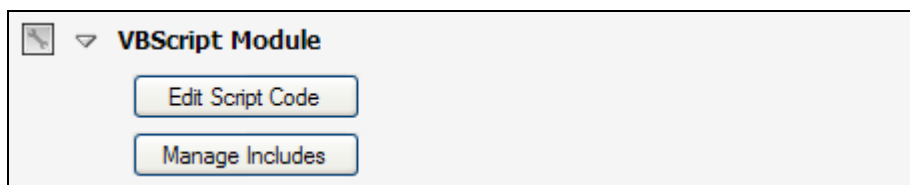
The Response Time criteria will evaluate the response time for each test iteration and ensure it is within the specified range. Values should be entered in units of seconds.



Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate popup window.

Success Criteria Rules: VBScript Module

The VBScript Module allows you to use inline VBScript functions to evaluate the response data.



You can define the script directly using the VBScript editor and you can also leverage existing functions and libraries by including these files using the Manage Includes dialog. A sample code snippet is included in the editor when you click on the Edit Script Code button.

The VBScript interface calls the defined function "Sub Evaluate(UNUSED)". Variables are passed to the VBScript environment through a global variable called Parameters which includes the following member variables:

- Parameters.VariableNames: array containing names of any existing active variables
- Parameters.VariableValues: array containing values of any existing active variables
- Parameters.Request: The current request
- Parameters.RequestHeader: the current request header
- Parameters.Response: The current response
- Parameters.ResponseHeader: The current response header

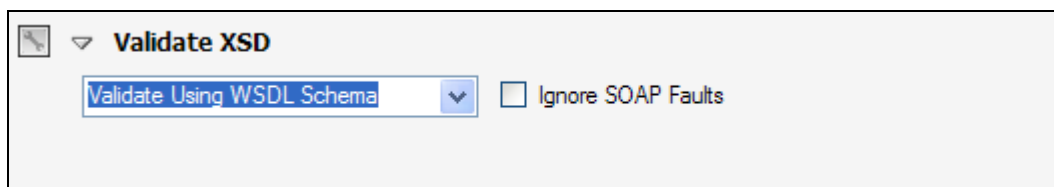
Additionally, Memory Table features are integrated directly within the VBScript modules. You can create, modify, set, append, and clear Memory Table variables.

A Memory Table variable action command can be created from the Memory Table menu, or whenever you type "MemoryTable." from the script editor.

```
MemoryTable.GetValue $MEMTABLE$  
MemoryTable.SetValue $MEMTABLE$ , VALUE  
MemoryTable.AppendValue $MEMTABLE$ , VALUE  
MemoryTable.ClearValue $MEMTABLE$  
MemoryTable.ReplaceValue $MEMTABLE$ , FINDVALUE , REPLACEVALUE
```

Success Criteria Rules: XSD Schema Validation

The Validate XSD Schema feature allows you to perform XSD schema validation of the response document against the Project schema to ensure the structure and data conforms to the expectations of the schema definitions. For WSDL projects the schema(s) are defined by the WSDL. For custom projects, you can specify the Schema by clicking on the documents node under configuration and choosing the load schema option.



For schema validation, you can choose to ignore SOAP faults since often a SOAP fault structure is not included in the schema, but SOAP faults is the appropriate provision per SOAP specification to use for error response. Thus, from a structure standpoint a SOAP fault can be considered valid. To allow for this, check the "Allow SOAP Faults" checkbox.

Note: This feature requires a valid XSD schema. If the XSD Schema is not able to be compiled due to errors in the schema, incomplete references, or invalid schema definition structure, the Validate XSD schema feature will not be able to properly validate the messages.

Success Criteria Rules: Schematron Validation

The Schematron Validation success criteria rule allows you to analyze the document against a set of Schematron definitions, which provide a language for making assertions about the presence or absence of patterns in XML documents. SOAPSonar supports two standards for Schematron: Schematron 1.5 and ISO Schematron. Based on the selection of this version, the appropriate Schematron engine will be used to evaluate the Schematron definition files against the document.

For Schematron 1.5 version engine validation, select the set of Schematron files to be used to validate the document. To add new files to the list, use the browse button to browse for the file, then click the "+" icon to add the file to the list.



To use the ISO Schematron validation engine, first select “ISO Schematron” from the engine list. On the dialog that appears, choose the primary Schematron file and be sure to add all include files. You will be prompted to add the detected include files once you press the “+” icon to add the primary Schematron file).




Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate popup window.

Success Criteria Rules: Invoke DLL Plugin

You can build your custom evaluation functions using the extensible SOAPSonar DLL plug-in interface. This interface allows you to build a DLL which implements the **EvaluateItem** function signature as documented in the interface document ISOAPSonarPlugin.vb found in the plugins directory under the installation directory. You can write the plug-in in any language since SOAPSonar invokes the DLL using reflection to determine whether the function signature has been implemented. If the function signature is found, the function **EvaluateItem** function will be invoked.

This function will be passed the request data, response data, and any variable substitutions in-scope for the current test iteration. You can also pass name/value pairs to the selected DLL by entering in the table provided.

 **Invoke DLL Plugin**

Custom Name Value Pairs to Pass to your EvaluateItem DLL Function

	Name	Value
1		
2		
3		
4		
5		
6		

Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate popup window.

Success Criteria Rules: Database Query

You can run any defined query against a target database as a success criteria event. Any SQL query, such as SELECT, INSERT, DELETE, DROP TABLE, etc can be run and the results of the query analyzed for the criteria (row count, value match, etc).

The screenshot shows the 'Database Query' configuration window. It includes a dropdown menu for criteria type set to 'Contains', a text input field for the criteria, an 'Execute Query' button, an 'ODBC' dropdown, a 'Connection String' text box containing 'Driver={SQL Server};Server=Test;Trusted_Connection=yes;', and an 'SQL' text box containing 'select * from Test'. To the right of the window, there are three blue callout boxes: the first explains the criteria field, the second explains the connection string field with a link to www.connectionstrings.com, and the third explains the SQL query field, suggesting the use of Count(*) for row count criteria.

When creating a database query, just as with other success criteria rules, you can use variables from the test within the SQL query itself, or as part of the criteria.

For connection strings to use for access to any ODBC compliant database, visit www.connectionstrings.com

Tip: To maximize screen space when configuring rules, click on the rule title to open the rule configurable in a separate popup window.

Success Criteria Rules: File Analysis

You can perform file analysis as a success criteria item. File analysis criteria include:

Exists – Referenced file exists

Does Not Exist – Referenced file does not exist

Contains – Referenced file contains a string match of the referenced data

Does Not Contain - Referenced file does not contain a string match of the referenced data

Regular Expression – Referenced file contents matches the referenced data regular expression

File Bytes (Min|Max) – File size is within min and max bytes

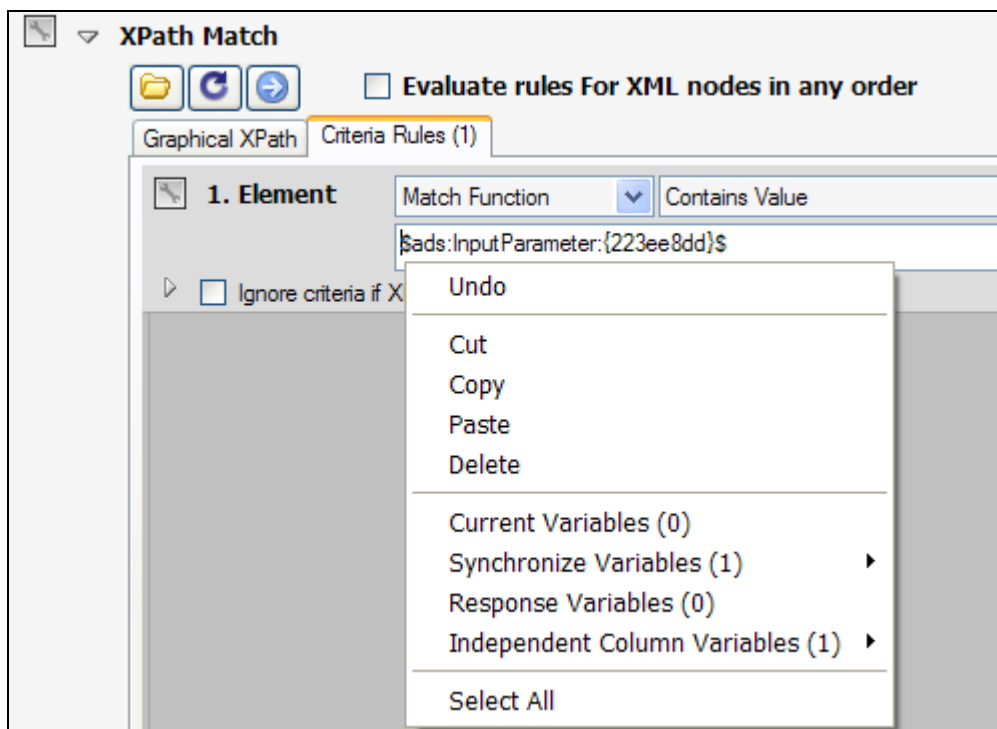
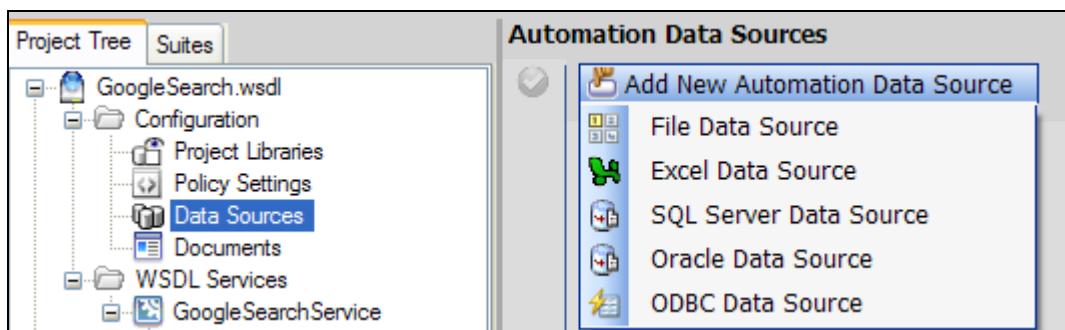
The screenshot shows the 'File Analysis' configuration window. It features a dropdown menu for criteria type set to 'Exists', a text input field for the filename, and a 'Browse...' button. The window title is 'File Analysis'.

Success Criteria Dynamic Variables

If you have the [Automation Module license component](#) enabled, you can dynamically parameterize the Success Criteria using test variables. This allows you to define data used to evaluate success and failure from a database, or from a previous test request or response values, or other dynamic variables sources.

For example, consider having a database table with 2 columns and 2 rows. In this example, we'll use the first row as the parameter for our SOAP input, and use the second column as the dynamic criteria. Each row in the data source will result in separate test iteration.

InputValue	ResponseCriteria
Spellin error	Spelling error
Another spellin error	Another spelling error



In this example since there are 2 rows, this would result in 2 test iterations and the response evaluation rule would change for each iteration based on the contents of the data table row

Once tests are automated using data sources for inputs and response success criteria rules can then easily extend your test inputs and related criteria simply by adding rows to your data source. See the [Automation Data Source](#) section for more information about dynamic data parameterization.

Dynamic Success Criteria matching provides an extensible means to extend a single test case to hundreds, even thousands of actual test iterations with varying success criteria for each defined externally within the referenced data source, runtime variable, ADF value, or context function.

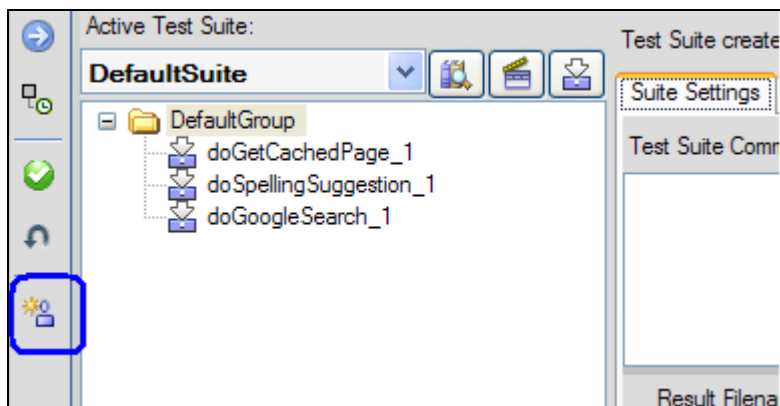
Baseline Regression Testing

SOAPSonar provides a rich set of features to allow you to create baseline result sets of your test sequences to use as baseline criteria for future test runs. Comparison to baseline measurements include sophisticated XML diff configuration which allows you to select which portions of the response documents to isolate and diff and also allows node fragment, element value, and attribute value exclusions to be defined such that values that tend to differ in each response (timestamps, etc) can be ignored so as to not skew the diff results. Regression baselines are associated with each test suite where they are defined and also stored with the project such that the project and the baselines can be shared among teams and departments. For example, your development team can set up a set of unit test baselines that are shared with the QA team for quick ramping of sample testing scenarios. Test suites with regression baselines can also be run using the command-line interface for build automation and scheduled regression testing.

Creating a Regression Baseline

Regression baselines are associated with a test suite so the first steps to creating a baseline result set is to create the test suite with the test cases that represent the set of inputs to be sent to the back-end server.

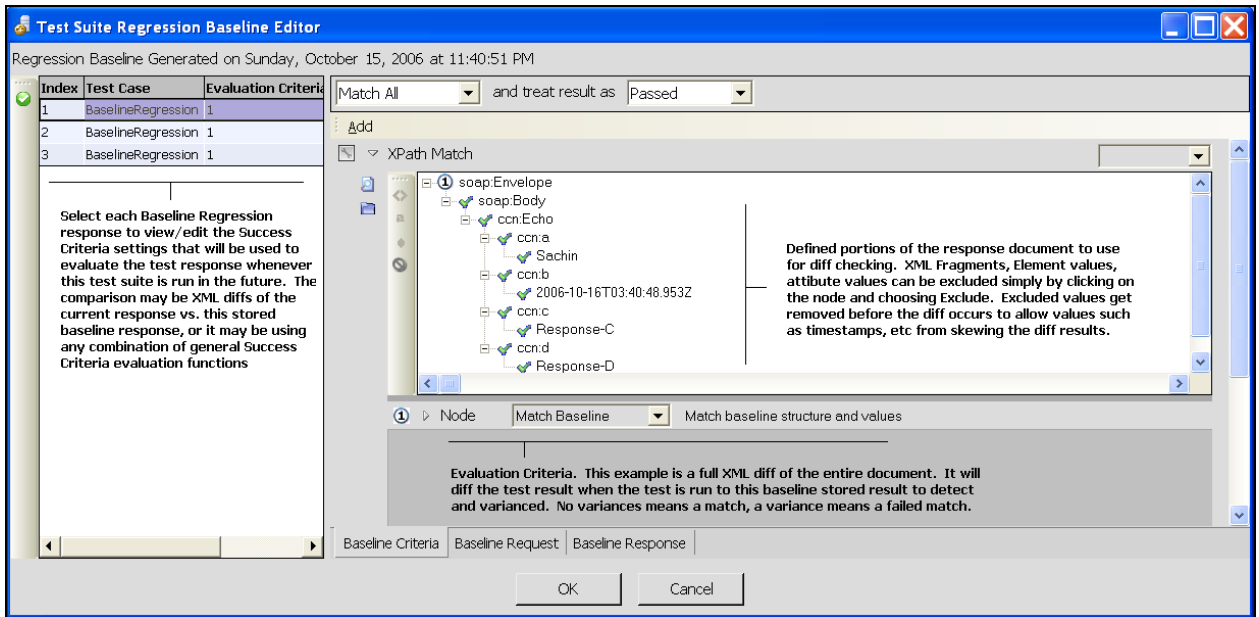
Once your test suite has been created, an icon appears when in QA mode for generating a baseline regression data set.



Generating a new baseline will run all test cases and test iterations in the test suite and store all the responses in sequence to an internal baseline regression data set. Before this baseline set of responses is generated, you will be presented with a dialog that allows you to choose how to initialize the success criteria functions for each baseline response. The options include an XML diff of the entire document, copying existing success criteria settings defined on each referenced test case, updating existing settings, or leaving the criteria empty by default where it can be edited and modified manually. The pre-populate options can reduce the time to generation regression testing to have the initial evaluation criteria built automatically.

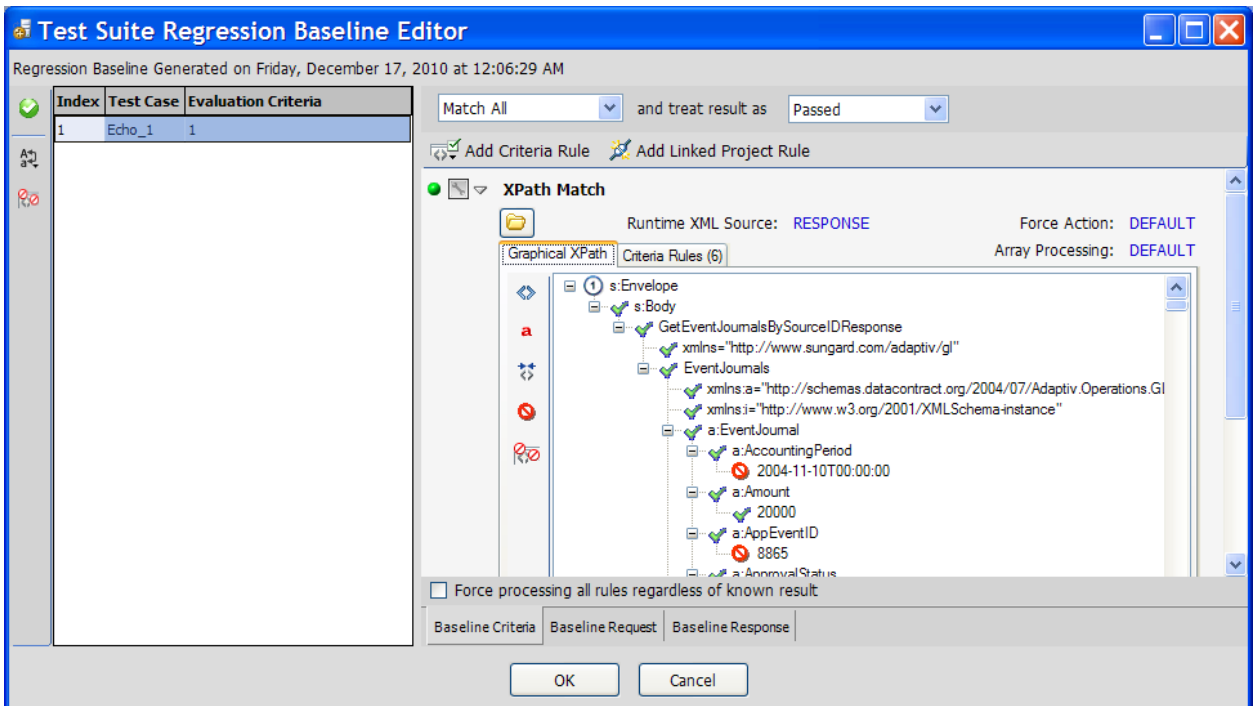
Regression Baseline Criteria Editor

Once the Baseline Regression data set has been created and stored, you are then presented with the Baseline Criteria editor where you can selectively choose for each test iteration how to determine success or failure. The criteria can be XML diff based where you choose to always diff the current test response against this stored baseline response and detect any variances. You also have the full range of success criteria settings for each baseline response that are available in the test case [success criteria](#) settings.



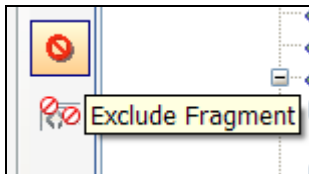
Omitting Nodes, Elements, or Attributes from Baseline XDiff Document Comparison

In many cases, there may be data that changes with each test run. If this data were to be included in the diff calculations, then the difference detection would continually trigger. To prevent items that are known to continuously change from impacting the diff calculation, the baseline editor provides the means to omit nodes, elements, and attributes from the diff.



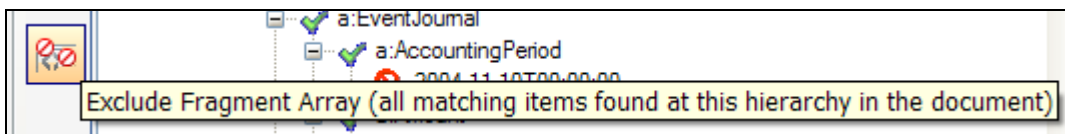
Omit Single Target

There are several ways to exclude items from the XML diff. You can directly-click on a graphical node, element, or attribute in the Editor and choose the Exclude Fragment option. This option excludes the value at the specific location in the response.



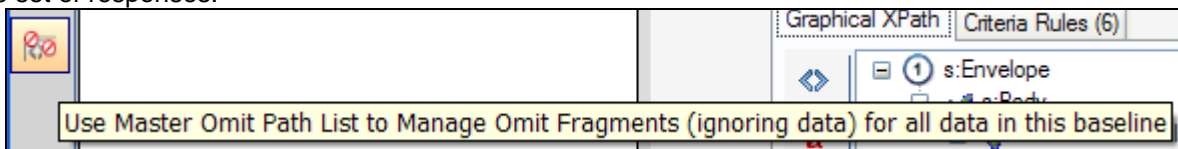
Exclude Array Targets

If the response values are within a structure or array and can be returned in varying order from one test run to another, or you want to omit this value in any instance at this depth, then choose the Exclude option to exclude all array representations of the target value at that hierarchy in the document.



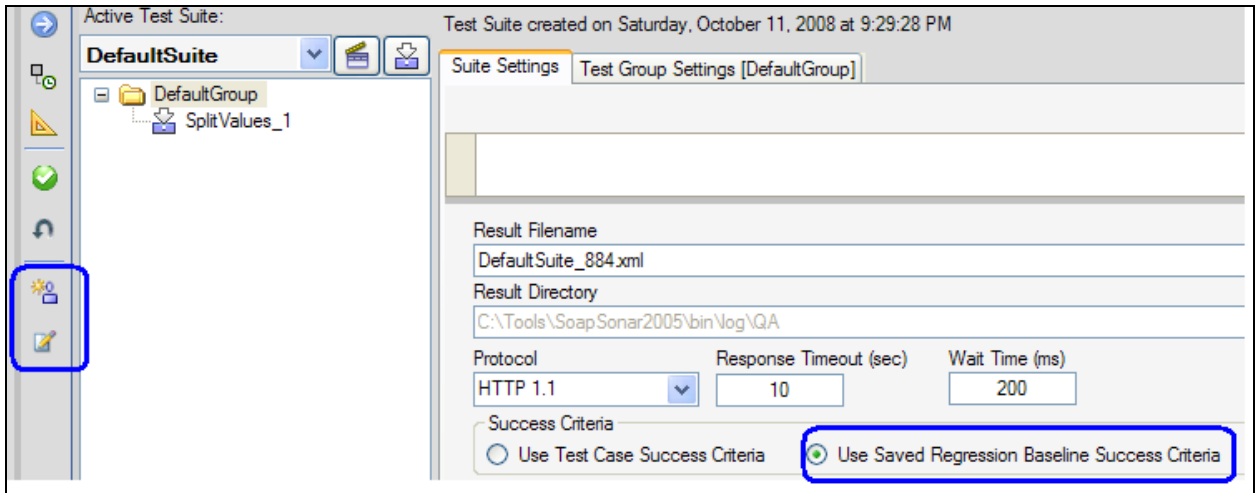
Exclude XPath Targets

To simplify the Exclude rule definitions, the individual settings can be overridden from a global set of XPath expressions that will define the Nodes, Elements, or Attributes to be excluded across the entire baseline set of responses.



Running a Baseline Regression Test

Once you have configured the success criteria for each baseline result in the test sequence, you can use this baseline set of data and criteria as the regression baseline when running the test suite. When you have generated a baseline regression set for a test suite, the option for Success Criteria on the Test Suite Properties will now include the setting for "Use Saved Regression Baseline Success Criteria". Choose that setting to run the test sequence using the stored baseline regression set and the configured criteria as the evaluation criteria for each response in the test suite sequence.



Generating an HTML Baseline Regression Diff Report

After running a test using baseline regression criteria you can generate an HTML report which highlights the Pass/Fail results of the test and also provides graphical diff reports for each test iteration. In this report, you can view the diff of the request vs. baseline request, response vs. baseline response, and each diff (with applied exclusions) for each XPath expression.

To generate the report, click on the log file and choose the report labeled “[HTML] Baseline Regression XML Diff Report”. You will be prompted for the directory to store the results. The results will be written to the selected directory with the name of the log file with a .htm extension and a subfolder of the test data created with the same name. The test data written with the report includes all actual XML requests and responses, as well as the baseline requests and responses. This data can be shared with teams analyzing the test results and also these files can be used to resubmit using the batch processing feature of SOAPSonar.

CROSSCHECK
networks

Functional Test Report - Test Suite [BaselineSuite]
Generated on 10/16/2006 at 12:31 AM

Show Detail ▾

Index	TestCase	Response Time	Request Bytes	Response Bytes	HTTP Code	TestResult	Request Diff	Response Dif
1	BaselineRegression	1.7	489	489	200	Pass		
Endpoint: http://1127.0.0.1:8080 Evaluation Rules: Match All and Treat Result as Pass Evaluation Data: soap:Envelope soap:Body ccn:Echo ccn:b Evaluation Function: Exclude from XDiff comparison () Evaluation Result: Pass. XML element value excluded from comparison soap:Envelope: Regression Baseline XML Node and Value Match () Evaluation Result: Pass. Response XML Nodes Match Baseline.								
2	BaselineRegression	1.7	487	487	200	Fail		
Endpoint: http://1127.0.0.1:8080 Evaluation Rules: Match All and Treat Result as Pass Evaluation Data: soap:Envelope Evaluation Function: Regression Baseline XML Node and Value Match () Evaluation Result: Fail. Response XML Element soap:Envelope soap:Body ccn:Echo ccn:b/ value does not match Baseline Element value.								

XMLDiff Report - Baseline vs Actual Response XPath Criteria Evaluation

Date and Time: Mon, 10/16/2006 12:31:23 AM

Baseline: BaselineRegression.Baseline-SC-1.3.xml
Response: BaselineRegression.Response-SC-1.3.xml

Compare File Results:

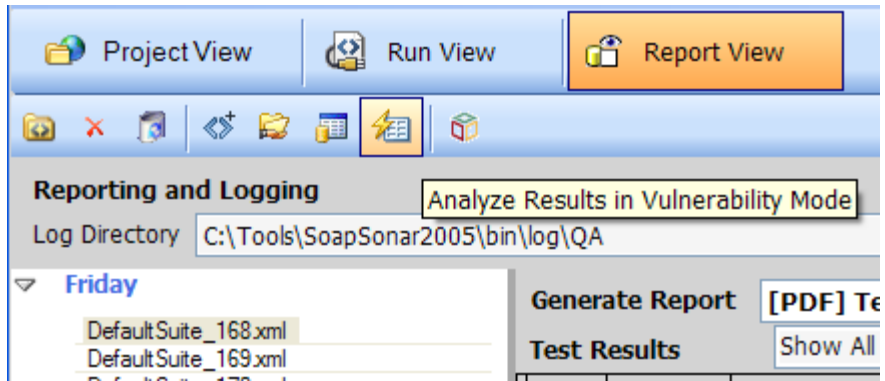
1 Change Found
0 Additions Found
0 Deletions Found

#	BaselineRegression.Baseline-SC-1.3.xml	BaselineRegression.Response-SC-1.3.xml	#
1	<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ccn="http://www.crosschecknet.com/sample/Echosvc">	<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ccn="http://www.crosschecknet.com/sample/Echosvc">	1
2	<soap:Body>	<soap:Body>	2
3	<ccn:Echo>	<ccn:Echo>	3
4	<ccn:a>Rahul</ccn:a>	<ccn:a>Rahul</ccn:a>	4
5	<ccn:b>2006-10-16T03:40:49.015Z</ccn:b>	<ccn:b>2006-10-16T04:31:13.578Z</ccn:b>	5
6	<ccn:c>Response-C</ccn:c>	<ccn:c>Response-C</ccn:c>	6
7	<ccn:d>Response-D</ccn:d>	<ccn:d>Response-D</ccn:d>	7
8	</ccn:Echo>	</ccn:Echo>	8
9	</soap:Body>	</soap:Body>	9
10	</soap:Envelope>	</soap:Envelope>	10

The resulting report provides icons to access the diff reports for request, response, and each XPath criteria defined.

Vulnerability Scan of QA Test Results

The results of a QA test run can be scanned against the active Automatic Vulnerability Discovery rules in Vulnerability mode. To perform a vulnerability scan of the test results, go to Report View, select a log file, then click on the Analyze results in Vulnerability Mode button.



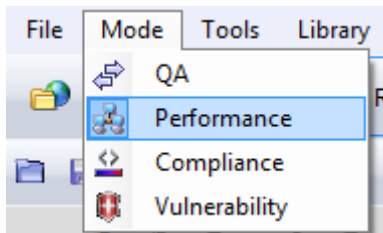
PERFORMANCE MODE

Performance run mode provides the ability to assess the throughput and response time characteristics of your back-end server through measurement diagnostics of tests and test sequences under concurrent (simultaneous) client load. Performance profiles include information such as Transactions per Second (TPS), Throughput, Min/Max/Ave response times, standard deviation, and 90% percentile values.

Subtopics in this section include:

- [Performance Test Cases](#)
- [Performance Test Property Settings](#)
- [Performance Mode Test Variables](#)
- [How Statistics are Calculated](#)

To switch to performance mode, select the mode menu and choose Performance.



Switching run modes does not impact the test cases and test suites that are already defined in the project, but rather changes the types of diagnostics that occur during the test run and made available in the logging and reporting section. Thus, you can choose to leverage existing tests in the project, or create new ones for performance analysis.

Performance Test Cases

Tests are created and managed the same across each test mode. This allows the same test cases and message paradigms to be used across each run mode. Refer to the [Building Tests](#) section for instructions how to create and manage test cases.

Performance mode test cases use the test configuration with the allocated virtual clients to generate the throughput and response time profiles for the transactions.


Performance Test Suite Property Settings

When in Performance mode, the options for running the tests are divided among settings applicable for all tests in the test suite, and settings for each test group in the test suite. These settings are available on the “Suite Settings” and “Test Group Performance Settings” tabs respectively. There is by default a loading engine included in the base console of SOAPSonar that can be used for load testing. This appears as “(local)”. You can also choose to use the SOAPSonar Distributed Loading Agents to use other machines to include in the load testing. This is configured on the Performance Loading Agents tab.

Test Suite created on Thursday, November 14, 2013 at 1:57:32 PM

Performance Testing Mode: Synchronous (Run each Group in Sequence)

Suite Settings | Group Performance Settings | Performance Loading Agents


Suite Comments: 

Suite Pre-Run Task List: **0 Tasks**

Suite Post-Run Task List: **0 Tasks**

Test Suite Settings:

Result Filename: DefaultSuite_84.xml .xml

Result Directory: C:\Tools\SoapSonar2005\bin\log\Performance 

HTTP Protocol: HTTP 1.1 Response Timeout (sec): 10

Override URI

Perform detailed transaction analysis

Store Transaction Statistics + Trace Errors Write Data to External Files

Disable test dependencies

Segment data source rows uniquely among virtual clients

Publish results to HP Quality Center project

Result Filename

The log file where the test result diagnostics will be written

Result Directory

The log directory where the test result diagnostics will be written

HTTP Protocol

Determines whether to use HTTP 1.0 or HTTP 1.1

Response Timeout

Amount of time each virtual client will wait for a response from a connected back-end server request before timing out and considering the request a failure.

Override URI

Redirects test request to the specified URI for all tests in the Suite, or replaces the IP address of all tests in the test suite, keeping the path portion of the URI the same.

Detailed Transaction Analysis (Error Tracing)

When checked, this setting enables full semantic parsing of each response against the defined success criteria rules. If you enable this setting when running performance tests there will be additional overhead imposed by having to parse and analyze each message response. Enabling this setting may also impact the throughput numbers due to the overhead of storing the data.

Tracing Analysis Options:

- Store Errors Only
 - Stores transactions when they fail according to the defined success criteria rules. Statistics will be captured only for those tests that fail.
- Store All Transactions
 - Stores every single request and response used during the performance testing run and all transaction statistics for each test
- Store Transaction Statistics Only
 - Disables storage of request and response data and only stores the transaction statistics for each iteration
- Store Transaction Statistics + Trace Errors
 - Stores transactions when they fail according to the defined success criteria rules, but enables all statistics to be captured regardless of Pass or Fail

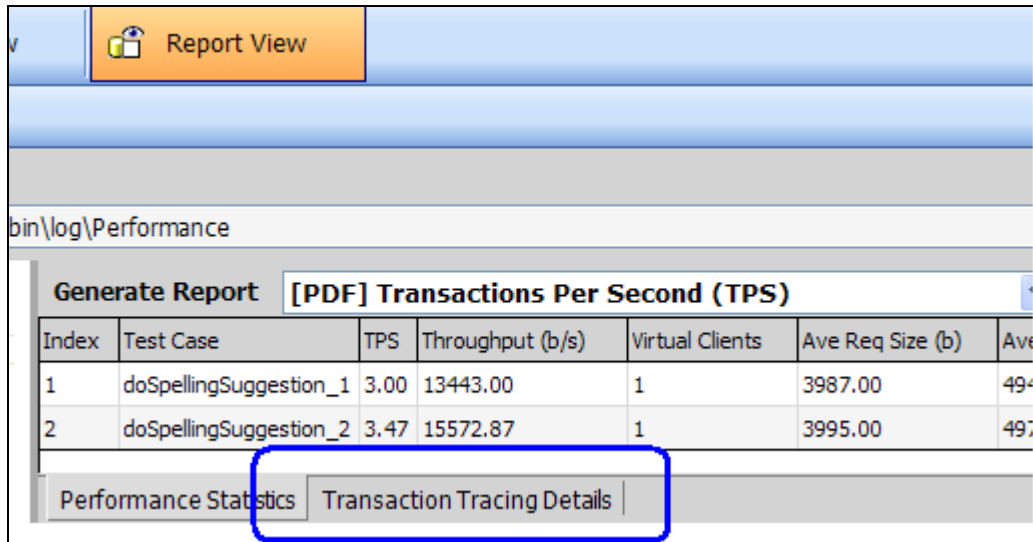
Storage Options:

- Write Data to External Files (Recommended)
 - Will write the request/response information to a subdirectory named the same as the log file in the target results directory. The statistics are also written to this directory in a CSV (Comma Delimited) file. This setting is the recommended setting to avoid too much overhead of memory storing results. This setting has no limits on the amount of information that can be stored.
- Embed Data in XML Log
 - This setting remains for legacy purposes but is not recommended to use due to the memory overhead and parsing time imposed by storing all the data within an XML file. If this setting is used, the amount of data that is stored is limited to the instance defined settings under the File->Settings and Preferences->Suite Settings section.

Note: The recommended usage of the Details Transaction Analysis feature is for diagnostic purposes. When running performance tests for pure throughput analysis, this setting should remain disabled since enabling this setting will degrade the loading capability of the SOAPSonar instance due to the overhead imposed by the success criteria rule framework analysis engine and the storage of request, response, and statistics information for every request. If you do want to enable this setting for unlimited data capture,

choose the Write to External Files method. If you choose Embed Data in XML Log, the limit imposed for capture can be configured under the File->Settings and Preferences->Suite Settings section.

When this setting is enabled, you can view the Transaction Tracing Details from the Report View screen under the tab "Transaction Tracing Details".



Disable Dependency Checking (Run Primary Tests Only)

This setting will disable the automatic variable dependency tests and run only the primary top level tests in the test suite, without running any of the dependent tests.

Segment Data Source Rows Uniquely Among Virtual Clients

For tests using data sources for data, this setting will divide the data source evenly among the allocated virtual clients such that each virtual client works from a unique section of the data source. If the number of rows in the data source is not equal or greater than the number of specified virtual clients, SOAPSonar will throw an error.

Run all Test Groups Asynchronously at the Same Time

When this option is not checked, each test group is run sequentially and each test case performance statistics are isolated and run individually. You can choose to run all test groups at the same time for performance profiles setting up several tests to run at the same time. This is useful when testing performance scenarios involving different test endpoints, or different web services operations with weighting of rates and the number of virtual clients.

Consider for example a scenario where there are 2 web services operations, CheckBalance and ModifyAccount. The CheckBalance call is called 10% of the time, where the ModifyAccount is called 90% of the time. You can tell SOAPSonar to build the performance profile for this by setting up 2 test groups, adding a test for CheckBalance to the first group, and a test for ModifyAccount in the second group. Then, set the asynchronous option to run both of these load profiles simultaneously.

Performance Test Group Property Settings

Test Suite created on Wednesday, February 13, 2013 at 1:22:19 PM

Performance Testing Mode Synchronous (Run each Group in Sequence)

Suite Settings | Group Performance Settings | Performance Loading Agents

Group Definition Mode: Manual Always Show Group Settings Expanded

EchoGroup Comments

Concurrent Loading Clients

Virtual Clients 3 Ramp Strategy Linear Ramp Up (sec) 0 Ramp Down (sec) 0 Link Time (ms) 0 IP (Default)

Interval Setting

Duration Type Duration Test Duration (seconds per test) 120

SLA Rate Throttling

Throttle 1 Requests Per

SplitValuesGroup Comments

Concurrent Loading Clients

Virtual Clients 5 Ramp Strategy Linear Ramp Up (sec) 0 Ramp Down (sec) 0 Link Time (ms) 0 IP (Default)

Interval Setting

Duration Type Duration Test Duration (seconds per test) 120

SLA Rate Throttling

Throttle 1 Requests Per

There are 3 type of interval settings to run performance tests, Duration, Iteration, and Data Source Iteration.

Iteration – Runs the tests for the specified number of iterations (request/response pairs)

Duration – Runs the tests for the specified duration (in seconds)

Data Source Iteration – Repeat all substitutions in the allocated data set(s) the number of times

Other properties defined later in this section include the number of concurrent virtual client engines to be used, the duration of the test, or the number of specified iterations, whether to throttle the loading throughput to the specified levels, and whether to capture error diagnostics in the event that errors are detected under load.

The term “concurrent clients” means that for each virtual client allocated for the test, there is an asynchronous engine allocated to run that client. Effectively, this means that all allocated clients are simultaneously making requests and gathering statistics on the responses, independent of each other.

Virtual Clients

The number of concurrent clients that will be allocated for simultaneous independent loading of the test group. Virtual clients are specified for each test group in the test suite and apply to all test cases in the test group.

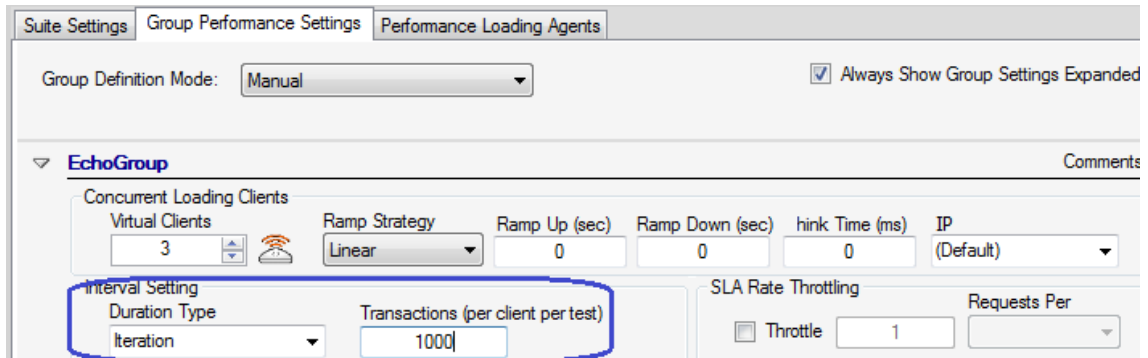
Test Duration

The duration of a test can be specified over a time interval, or a specified number of test iterations. Duration Mode is the test mode for time based testing where the virtual clients run for the specified time span. Iteration Mode is the test mode for running a specific number of test iterations per virtual client

where the duration of the test is dictated by the time it takes each virtual client to complete the specified number of transactions.

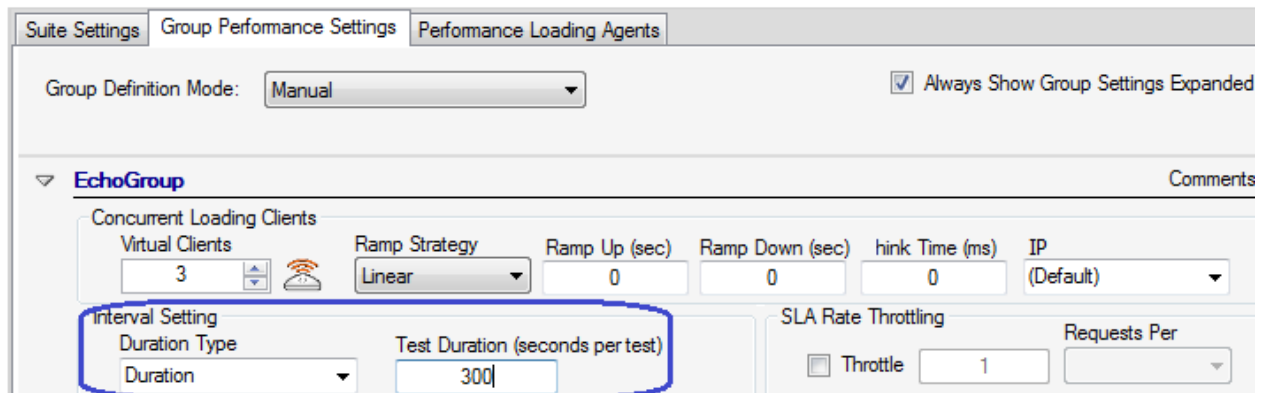
Iteration Mode

In iteration mode, you define a specified number of iterations that each virtual client will send for each selected test case. Thus, if you have for example 10 clients with a client iteration value of 100, this will result in 100 requests per client, or a total of 1000 iterations for each selected test in the Test Group. Once all tests have been processed, the statistical results will be tabulated and provided on the logging and reporting screen.



Duration Mode

In duration mode, you specify the duration for each selected test in the Test Group to run. When the duration has expired, all allocated virtual clients will stop running the current test, collect results, and move on to the next selected test. Once all tests have been processed, the statistical results will be tabulated and provided on the logging and reporting screen.



Duration mode is also commonly used for endurance testing where you may want to run the tests for periods of hours instead of minutes to assess the capabilities of the back-end service to handle extended periods of load.

Data Source Iteration Mode

Data Source Iteration mode will use allocated data sources to the test request as the actual test iteration value. The additional parameter for this mode is the number of times to repeat the data source row sets. A value of 1 will iterate only once through the data source row set. The number of data source row iterations is determined by the number of rows in the largest data source table allocated.

The screenshot shows the configuration interface for Performance Loading Agents. The 'EchoGroup' section is expanded, showing 'Concurrent Loading Clients' with 'Virtual Clients' set to 3 and 'Ramp Strategy' set to 'Linear'. Below this, the 'Interval Setting' section is highlighted with a blue box, showing 'Duration Type' set to 'Data Source Iteration' and 'Data source loops (per client per test)' set to 2. To the right, the 'SLA Rate Throttling' section shows 'Throttle' checked and 'Requests Per' set to 1.

SLA Rate Throttling

When running performance tests, you can choose to have the allocated virtual clients send requests as fast as the client machine will allow, or you can set an SLA Rate in which accesses will occur. By setting a Throttle rate, the collection of virtual clients per Performance Agent will adhere to the setting such that the overall rate of data does not exceed the Throttle rate specified.

The close-up screenshot shows the 'SLA Rate Throttling' section. It includes a checkbox labeled 'Throttle' which is checked, a text input field containing the number '1', and a dropdown menu labeled 'Requests Per'.

Throttle rates can be specified in intervals of Second, Minute and Hour. SOAPSonar will intelligently normalize the requests across all virtual clients to provide a uniform load distribution for each time period specified.

Note: Throttle rates are calculated per Agent. Thus, if you have the embedded "(local)" agent allocated, and a distributed loading Agent then the SLA rate settings will be per agent. Thus, the SLA rates would actually be calculated by The formula: Total Rate = SLA Rate * Agents.

Ramping Virtual Clients

Before running each test group in performance mode the virtual clients can be configured to warm up over a specified time interval called "ramp up" time. The clients will be started over the ramp up interval according to the strategy specified. For linear strategy the clients will be started in linear, equal time offsets across the ramp up period. For random strategy the clients will be started in random intervals across the ramp up period. Performance measurements and statistics are not calculated during the ramp up period, and are started once the ramp time is complete and the test begins. A ramp up time of 0 (default) will start all virtual clients immediately. The ramp down interval is the time allowed for the virtual clients to end gracefully upon completion of the test. Performance measurements and statistics are not calculated during the ramp down period and the beginning of the ramp down period marks the end of the performance test for the current set of statistics.

Ramp Up

The Ramp Up time defines the warm up duration for the allocated virtual clients whereupon the clients are started and begin the testing sequence. Ramp Up time is outside of the actual test time and no statistics are gathered during the Ramp Up period. Upon completion of the Ramp Up period, the define test duration begins and statistics are gathered.

Ramp Down

The Ramp Down time defines the wind down duration for the allocated virtual clients whereupon the clients are provided a period of time defined by the Ramp Down period to terminate gracefully. Ramp Down time is outside of the actual test time and no statistics are gathered since the Ramp Down period occurs after test duration completion.

Ramp Strategy

The Ramp Strategy determines the intervals within the Ramp Up period where the virtual clients are started. A Linear ramp strategy takes equal time segments across the Ramp Up duration and starts the virtual clients at equal intervals across the time span. A Random ramp strategy takes equal time segments across the Ramp Up duration and randomizes the virtual client start sequence within each of these intervals.

Think Time

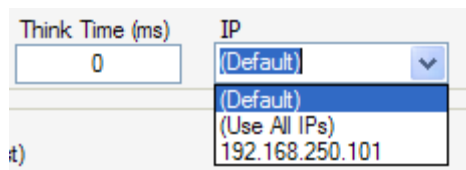
Specifies the programmatically applied time delay that is observed for each virtual user for each test iteration. Think time is a statistical emulation of the amount of time actual users spend reading or thinking about an input before performing the action.

Concurrent Load

The number of virtual clients specified will result in separate loading engines created to simultaneously load the back-end server and gather profile statistics. With the addition of a [Performance Agent Pack](#), you can extend the base capabilities of the loading with additional sets of 50 concurrent clients. Agents can be distributed among the embedded loading agent “(local)” or across any number of SOAPSonar Loading Agents, which can be installed on separate machine and managed with a SOAPSonar Enterprise Server Edition console.

Multiple Source Client IPs

If you have a multi-homed NIC with multiple IP addresses associated, you can set SOAPSonar performance mode to distribute the load across the bound IPs. This setting can be found under the Group performance setting.



Performance Mode Test Variables

Test case variable replacement occurs in Performance mode similar to QA test mode.

In Performance mode, ADF and ADS variable references which have multiple iterations of data variable replacements are handled in round-robin fashion when running performance tests either in duration or iteration mode. In other words, a variable reference that has 3 value replacements of InputA, InputB, and InputC would have this sequence repeat (InputA, InputB, InputC, InputA, InputB, InputC, ...) until the test duration has completed.

Runtime Variable references that create dependency invocations will be handled by invoking the entire test dependency sequence iteratively for the duration of the test. In other words if we have 2 test cases, MainTestCase and DependentTestCase, The sequence for the performance test would be (DependentTestCase, MainTestCase, DependentTestCase, MainTestCase, ...) until the test duration has completed.

How Statistics are Calculated

Performance runs will result in a number of metrics about the statistics for the test case run. These statistic values are defined below.

Response Time – Time from first byte of message sent to receipt of response and reading all response data from the network.

Min Response Time – Minimum response time value for each data set

Max Response Time – Maximum response time value for each data set

Ave Response Time – Average response time value for each data set

Standard Deviation Response Time – Standard Deviation calculated as follows:

- a. Calculate the mean m .
- b. Find the difference between each data point and the mean $(x-m)$.
- c. Take the square of each $(x-m)$
- d. Add all $\text{sqr}(x-m)$ for all data points.
- e. Divide it by total points $N - 1$.
- f. This gives you the variance.
- g. The square-root of the variance is the standard deviation

90 Percentile Response Time – Calculated as follows

$$90 \text{ Percentile} = \text{Mean} + (1.2815 * \text{StdDev})$$

The 90th percentile is that value x such that:

$$90\% \text{ of the values are } \leq x \text{ and } (100-90)\% \text{ are } \geq x.$$

The median is an example of a percentile measurement that represents the 50th percentile

Transactions Per Second – TPS is the number of request/response messages processed per second.

This is measured by the total amount of time the test ran for divided by the number of actual request/responses pairs that were made.

Distributed Loading Using SOAPSonar Agents


Performance tests can use SOAPSonar Agents installed on other machines to participate in the loading tests. To allocate Performance Agents you will need to have a SOAPSonar Enterprise Server Edition license, and then the following steps:





- 1) Download the SOAPSonar Agent using the Agents->Download SOAPSonar Agent Installer menu item
- 2) Install the Agent on the machines you want to use for loading. Ensure that the SOAPSonar Agent service is running and that Port 9570 is accessible from external systems.
- 3) (Optional): Go to Agents->Configure Performance Agents to set up the default Agents that are eligible for association with the Test Suite. This is not required since you can configure agents directly from the Test Suite as well.
- 4) Go to the Test Suite, and click on the Performance Loading Agents tab. Allocate the agents using the button options provided.

Test Suite created on Wednesday, February 13, 2013 at 1:22:19 PM

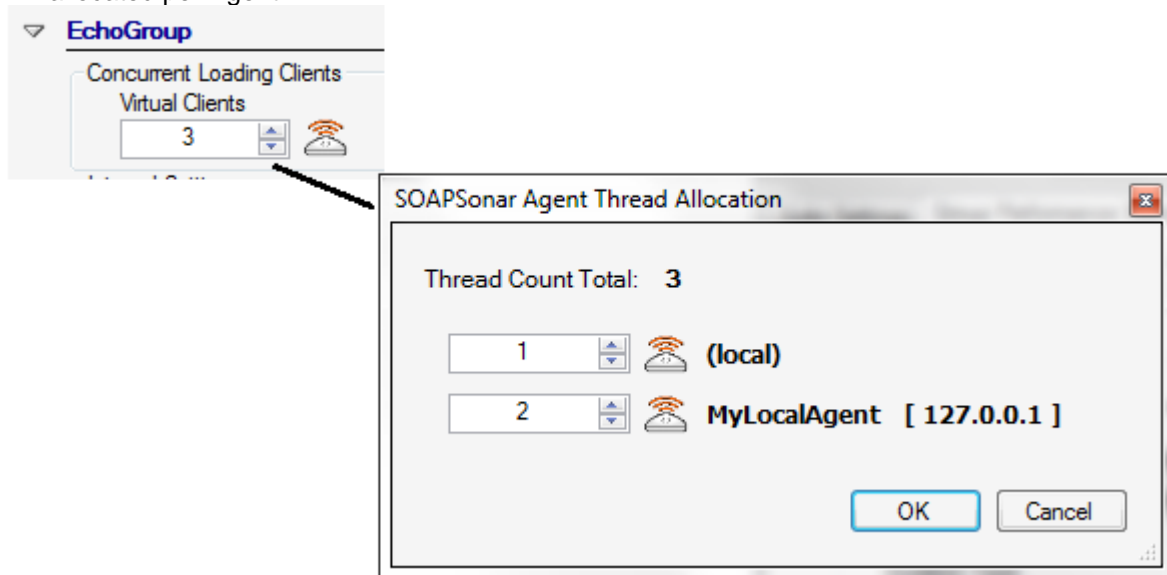
Performance Testing Mode

Suite Settings | Group Performance Settings | Performance Loading Agents



Enabled	Agent Display Name	Agent IP Address	Comm Port
	 Local Agent		
	<input type="text" value="My-VMWare-Agent"/>	<input type="text" value="10.5.1.3"/>	<input type="text" value="9570"/> 

- 5) Go to the Group Performance Settings screen and click on the **Concurrent Loading Clients** setting, or the SOAPSonar Agent icon, to allocate the individual loading clients that will be allocated per Agent.



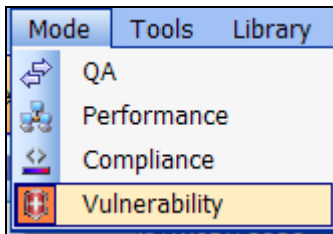
VULNERABILITY MODE

Vulnerability run mode provides patented XSD-Mutation technology that dynamically builds error boundary condition tests for your APIs using intelligent mutation of the WSDL schema in order to alter the message characteristics to assess the back-end server response characteristics. This technology can be used both to assess exposures from a risk mitigation perspective, and also to automatically evaluate the robustness of your back-end server application code logic's exception handling.

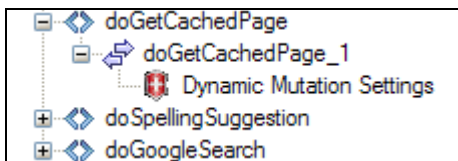
Subtopics in this section include:

- [Static Attack Vectors](#)
- [Dynamic Attack Vectors](#)
- [Automatic Vulnerability Discovery Libraries](#)
- [Scanning Responses using AVD Definitions](#)
- [Risk Score and Difficulty](#)

To switch to vulnerability mode, select the mode menu and choose Vulnerability.



When running in vulnerability mode, a new node named “Dynamic Mutation Settings” will appear as a child of each defined test case node in the project tree. This node appears as a child because it is dynamically generated from the test case, known also as the base test case.

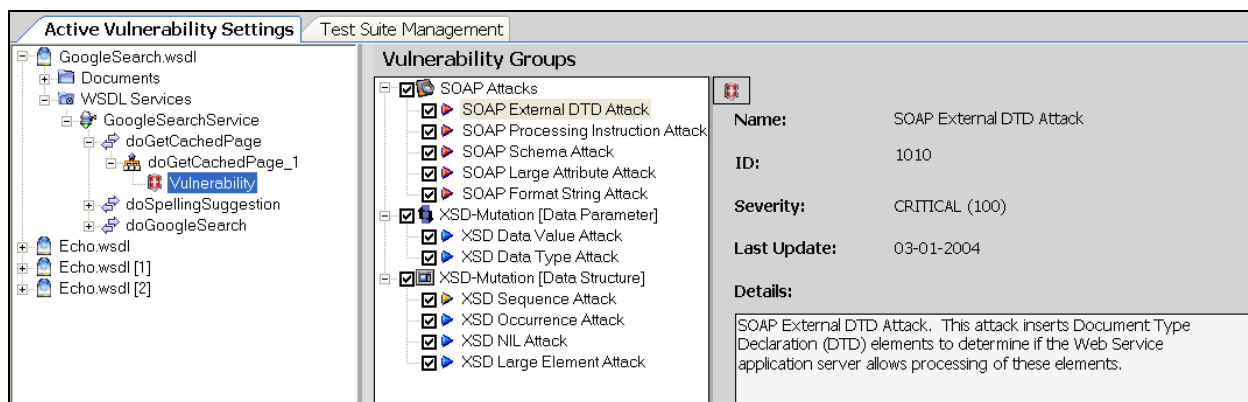


The unique ability of the patent-pending XSD-Mutation technology to dynamically mutate each instance of a base test case based on the schema provides the first and only solution on the market which can create penetration attack vectors dynamically based on message schema.

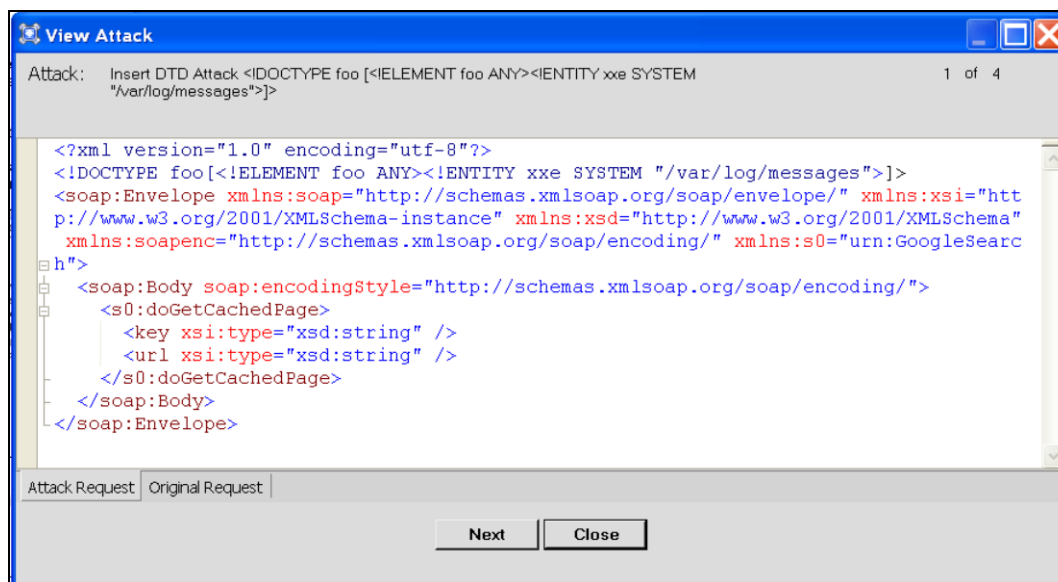
Static Attack Vectors

Certain attack vectors are considered static attack vectors as they are designed to inject and modify the base request in known ways to induce responses from the back-end service that can be evaluated to determine the robust error handling of disparate inputs.

Static Attack vectors include injection of external DTD references, injection of XML Processing Instructions, breaking of the SOAP Envelope structure per the SOAP specification schema, etc. These attack vectors are known as static because they effectively do not change based on the SOAP request input. To view the static attack vectors that are generated for each test case, click on the “Vulnerability” node which appears as a child node under the test case. Click on each item to see the properties and details for the item on the right side of the screen.



The static vector attacks appear under the “SOAP Attacks” section. Each attack vector can be viewed to see how the message will be dynamically modified when the test is run simply by clicking on the shield icon on the right or double-clicking the entry itself. This will bring up a viewer as shown below which shows the number of attack vector iterations which will be created and allow you to shift between the Attack Request and the Original Request to see what changes are going to be performed to the XML before it is submitted to the back-end service.

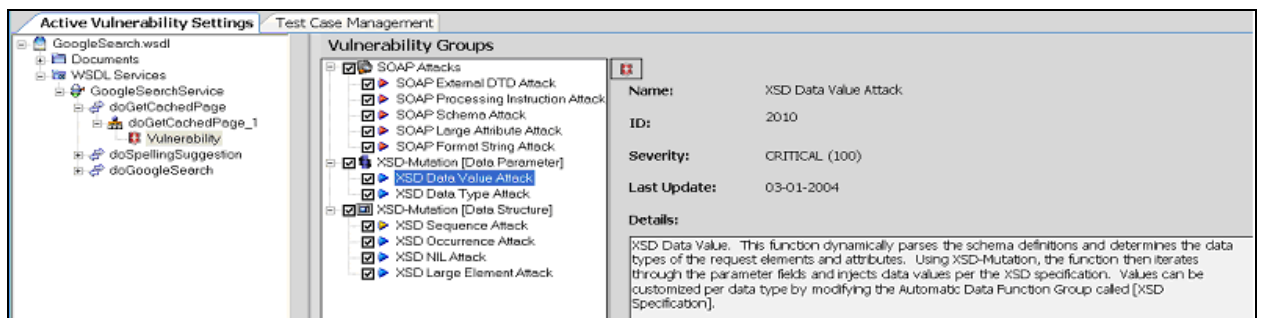


Dynamic Attack Vectors

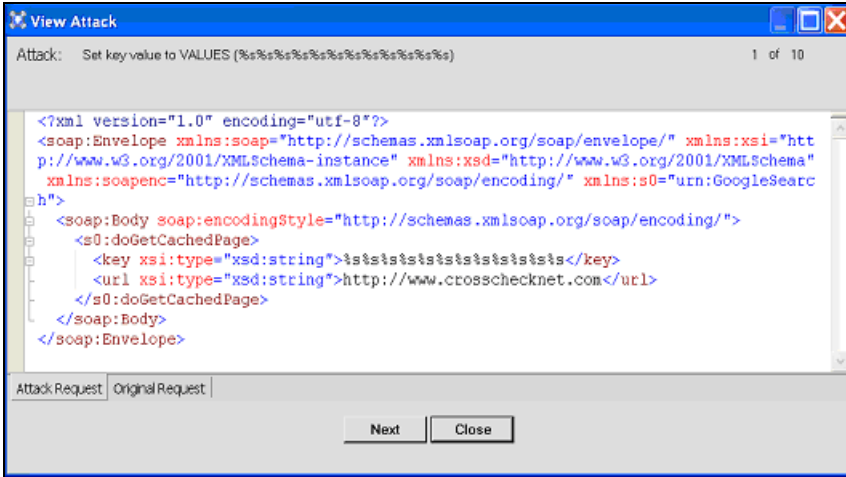
Attack vectors based on Crosscheck Networks patent-pending XSD-Mutation technology are known as dynamic attack vectors. This type of attack vector is the first and only dynamic penetration testing engine available for SOAP/XML data. The key to dynamic attack vectors is that they use the WSDL document schema in order to build a vulnerability model of XSD-Mutations. Why this is essential is because web services SOAP and XML provide the logical entities serialized on the wire to be interpreted by the service container and application layer. In order to validate the application code paths, the injection attacks must provide inputs that isolate coding paths intelligently. This is accomplished through the patent-pending XSD-Mutation technology which uses the schema to learn about the logic constructs of the application code of the service itself (also known as Grey Box Testing). The engine then takes a valid base request and intelligently mutates the request to isolate various decision points in the application layer for error handling robustness.

In conjunction with the XSD-Mutation, is the risk assessment and risk mediation components. These components called Automatic Vulnerability Detection libraries provide an extensible analysis harness for the results of the automated boundary condition testing to determine the risk exposure of governance violations, data leakage, and logic errors.

Dynamic Attack vectors include XSD Data Value mutations, XSD Data Type mutations, XSD Sequence mutations, XSD Occurrence mutations, XSD nil attribute injection, and XSD Large Element injection. To view the dynamic attack vectors that are generated for each test case, click on the “Vulnerability” node which appears as a child node under the test case. Click on each item to see the properties and details for the item on the right side of the screen.



The dynamic vector attacks appear under the “XSD-Mutation” sections. Each attack vector can be viewed to see how the message will be dynamically modified when the test is run simply by clicking on the shield icon on the right or double-clicking the entry itself. This will bring up a viewer as shown below which shows the number of attack vector iterations which will be created and allow you to shift between the Attack Request and the Original Request to see what changes are going to be performed to the XML before it is submitted to the back-end service.

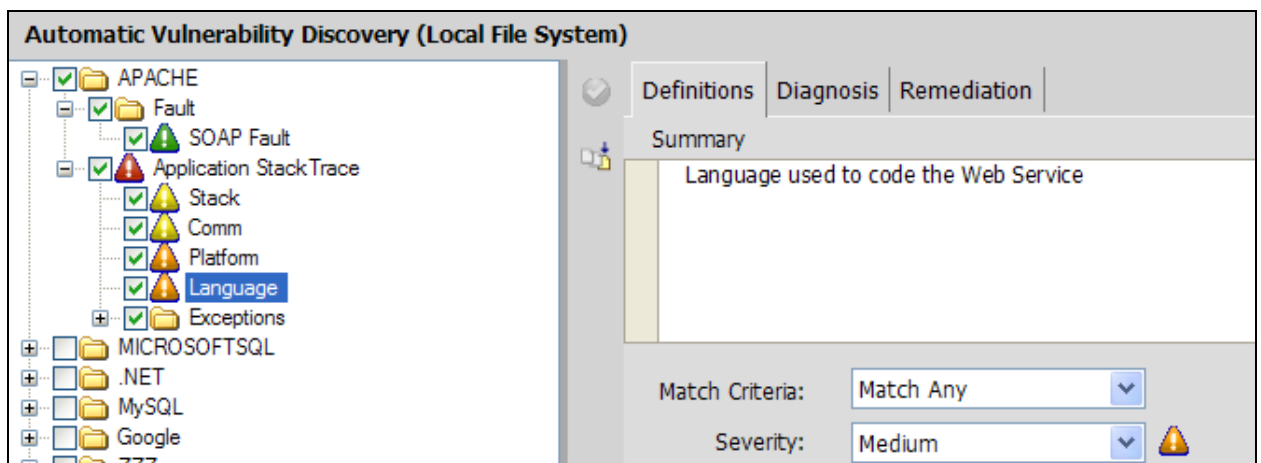


Automatic Vulnerability Discovery Libraries

Risk mitigation and risk mediation tools are provided for response analysis of the request values generated in Vulnerability mode. What constitutes a risk depends on many factors and can vary from one deployment to another. For example, consider if the web service is leaking stack trace information about a database query or about vendor specific implementation details of the web service container. In deployment where clients and servers only communicate internally within a secure network, this type of information leak may not consider this to be a risk. However, as most services are, a web service available to external trading partners where this information is being transmitted across the corporate network boundary is another case altogether.

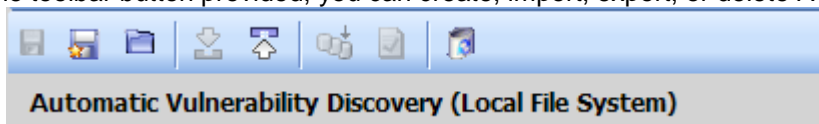
To access the AVD libraries, go to the Library menu and select “Automatic Vulnerability Definitions”.

An AVD library is a list of evaluation criteria that scans all the responses returned as a result of running attack vectors to look for anomalies, governance violations, proprietary data leakage, or any other analysis criteria you establish as representative of a vulnerability risk.



With AVD libraries, you can create a set of governance policies that automatically analyze the responses to highlight and report on matched criteria. Each item in the library can be assigned a priority, a summary description, detailed description, and risk mediation steps to help identify how to prevent the anomaly from occurring, or mitigating the anomaly exposure. SOAPSonar™ provides some default detection libraries to show examples of how to build assessment criteria for evaluation of the responses.

Using the toolbar button provided, you can create, import, export, or delete AVD library criteria.



AVD Criteria

AVD Criteria leverages the success criteria evaluation functions to analyze responses for match criteria. In the context of vulnerability assessment, the success criteria functions are used to trigger a match on the AVD category item. SOAPSonar provides some pre-populated AVD categories you can use as samples to correlate to your own environment's risk requirements and governance policies.

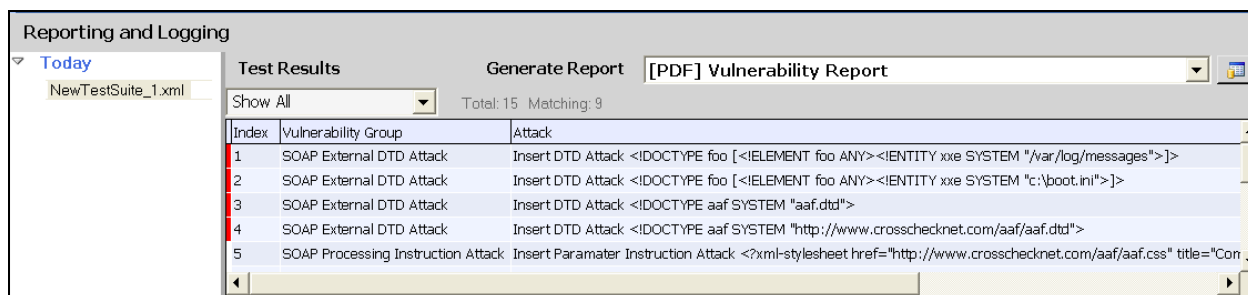
The AVD criteria items are comprised of the reporting criteria which show up in the vulnerability report when the specified AVD match criteria is triggered. The Definition text is a summary of the criteria category, the Diagnosis is a more descriptive statement of what it means that the AVD criteria specified triggered a match, and the Remediation provides information that can provide guidance as to how to mitigate the types of response messages from occurring in the first place.

The criteria items themselves function in the same manner as Success Criteria specified for the test cases in QA and Performance modes. Please refer to the [Success Criteria Rules](#) section in this document for detailed information about how to build evaluation criteria for match detection.

Scanning Responses using AVD Definitions

You can scan a result set using the enabled AVD definition items. When you view a log file from a vulnerability test run, it will scan the result set using the enabled (checked) items from the currently loaded AVD library. To scan the log file with different AVD criteria, simply load the criteria library and enable (check) the AVD criteria items to be active when scanning log results.

Responses are stored in the log files accessible from the reporting and logging section. Click on a log file to perform an active scan of the result set to detect any AVD criteria matches. Matches will appear with a red indicator in the left column and the Vulnerability report will show each AVD criteria that triggered with the Definition, Diagnosis, and Remediation text defined.



The screenshot shows the 'Reporting and Logging' section of the SOAPSonar interface. It displays a table of test results for 'NewTestSuite_1.xml'. The table has three columns: 'Index', 'Vulnerability Group', and 'Attack'. There are 5 rows of data, with the first three rows highlighted in red, indicating matches. The 'Generate Report' button is set to '[PDF] Vulnerability Report'. The table shows the following data:

Index	Vulnerability Group	Attack
1	SOAP External DTD Attack	Insert DTD Attack <IDOCTYPE foo [<ELEMENT foo ANY><ENTITY xxe SYSTEM "/var/log/messages">]>
2	SOAP External DTD Attack	Insert DTD Attack <IDOCTYPE foo [<ELEMENT foo ANY><ENTITY xxe SYSTEM "c:\boot.ini">]>
3	SOAP External DTD Attack	Insert DTD Attack <IDOCTYPE aaf SYSTEM "aaf.dtd">
4	SOAP External DTD Attack	Insert DTD Attack <IDOCTYPE aaf SYSTEM "http://www.crosschecknet.com/aaf/aaf.dtd">
5	SOAP Processing Instruction Attack	Insert Paramater Instruction Attack <?xml-stylesheet href="http://www.crosschecknet.com/aaf/aaf.css" title="Con

Risk Score

SOAPSonar provides a summarized report of the AVD criteria that matches against the responses returned during a vulnerability vector attack run. In this summarized report is scoring value called Risk Score. Risk Score represents the weighted score of triggered AVD criteria across the population of requests made to the back-end system. Weightings are determined as percentages based on severity where Critical=1.0, High=0.75, Medium=0.50, Low=0.25, and Info=0.10. If multiple rules trigger on the same request, the highest risk item will be the one that is weighted. Then the risk score is computed as follows:

$$\text{RISK} = \left(\frac{\text{SUM of Weighted Triggers}}{\text{Number of Total Requests}} \right) * 100$$

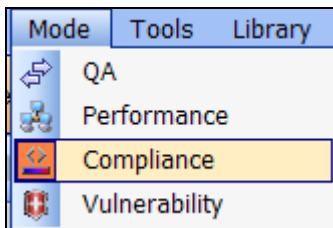
COMPLIANCE MODE

Compliance run mode provides access to the patent-pending XSD-Mutation technology that can dynamically exercise your Web Service APIs intelligently based on the compliance schema in order to alter the message characteristics and assess the back-end server conformance to the compliance specification or schema. This mode can be used both to assess exposures from a risk mitigation perspective, determine the level of active compliance with regard to the defined specification, and also to automatically evaluate the robustness of application level code path exception handling for messages falling outside of the expected boundaries.

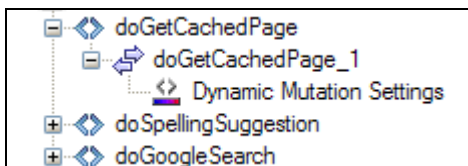
Subtopics in this section include:

- [Design Time WS-I Basic Profile WSDL Analysis](#)
- [Active Run-Time WS-Basic Profile SOAP Analysis](#)

To switch to compliance mode, select the mode menu and choose Compliance.



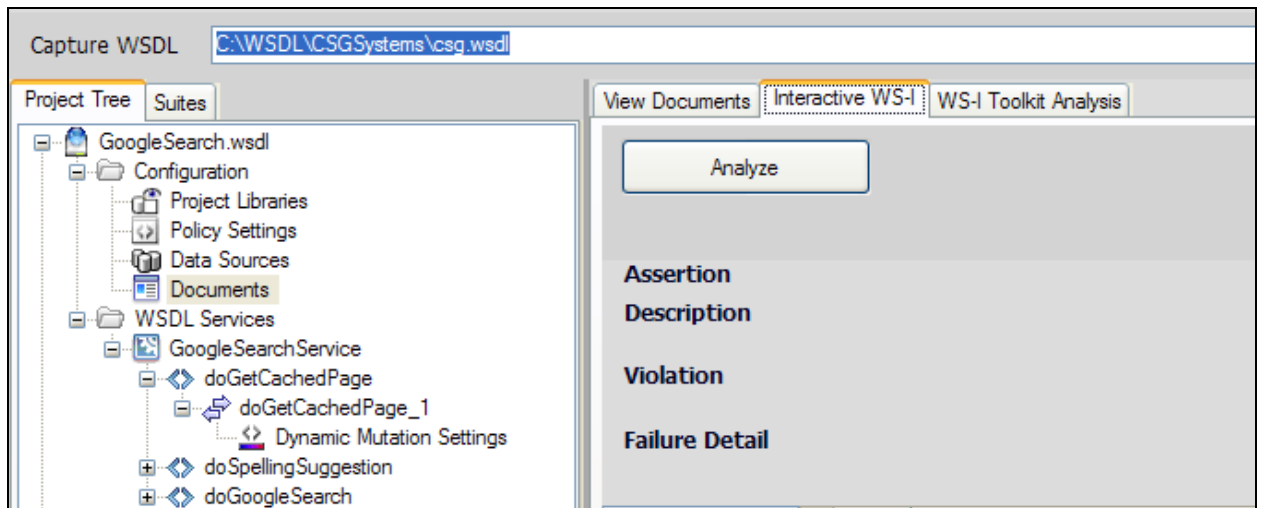
When running in compliance mode, a new node named “Compliance” will appear as a child of each defined test case node in the project tree. This node appears as a child because it is dynamically generated from the test case (referred to also as the base test case).



The unique ability of the patent-pending XSD-Mutation technology to dynamically mutate each instance of a base test case based on the WSDL schema provides the first and only product on the market which can create active compliance assessment dynamically based on each WSDL schema.

Design Time WS-I Basic Profile 1.1 WSDL Analysis

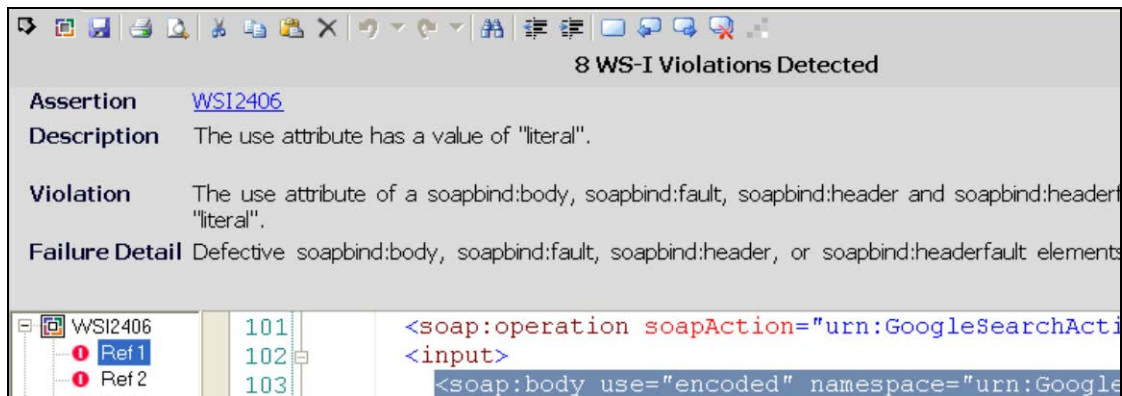
Crosscheck Networks is a member of the WS-I Organization and provides integrated design time WSDL analysis scanning using the WS-I Basic Profile 1.1 assertion engine. This analysis looks for assertion violations of the WS-I Basic Profile 1.1 assertions for the structure and content of the WSDL document. SOAPSonar Enterprise extends this further by providing WSDL highlighting for each failed assertion to quickly see what areas of the WSDL as failing the profile assertions.



When capturing a WSDL document, you can configure via the SOAPSonar™ settings to automatically perform a WS-I analysis scan in conjunction with the WSDL capture. Since this can sometimes be a time consuming process, you can also choose to run the WSDL WS-I Design-time analysis at any time manually by going to the WSDL document view and clicking on the WS-I analyze button.

Once the analysis is complete, you will see the number of failed assertions that were detected as well as a tree view of the assertion groups which you can click to go to the highlighted section of the WSDL that

triggered the assertion (if applicable). Some assertions are general assertions about the WSDL and therefore can not be specifically highlighted as a particular text block in the WSDL.

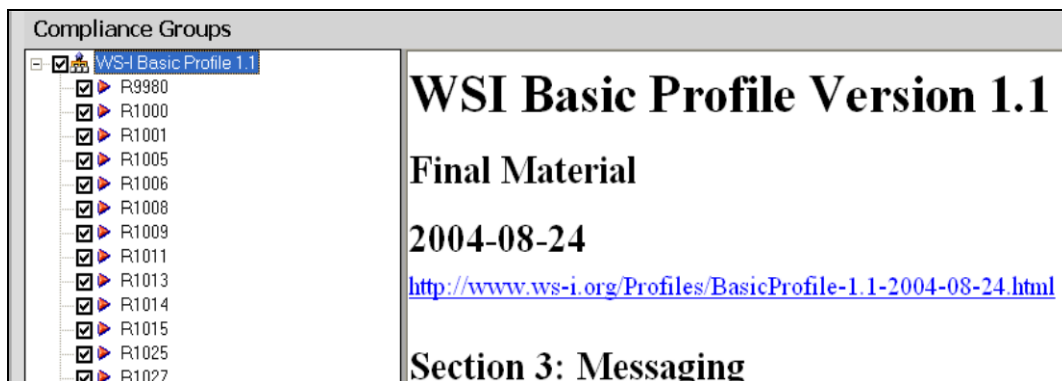


Active WS-I Basic Profile 1.1 SOAP Analysis

While many vendors provide design-time WSDL Basic Profile 1.1 analysis, this is only part of the equation to assess the interoperability posture of the Web Service. Design-time WSDL analysis provides only the assurance that the WSDL document itself is defined in a WS-I compliant manner. It does nothing to measure actual Web Service messages interoperability. Active WS-I Basic Profile 1.1 testing is an exclusive feature to SOAPSonar™ and leverages Crosscheck Networks patent-pending XSD-Mutation technology to actively send messages to your Web Service that intelligently mutate the base SOAP requests to isolate and test each requirement in the WS-I Basic Profile specification for SOAP messaging.

The Active WS-I Basic Profile compliance group is shown when clicking on any “Compliance” node that appears under the test case in Compliance run mode. This group extracts each requirement from the WS-I Basic Profile 1.1 Section 3 Messaging and builds dynamic request profiles that extend the base test case defined to exercise each stated requirement in the profile specification.

To see all of the requirements set forth in the specification click on the top level “WS-I Basic Profile 1.1” node. The document that appears in the right pane is hyperlinked with the active XSD-Mutation functions that will activate when the test case is run.

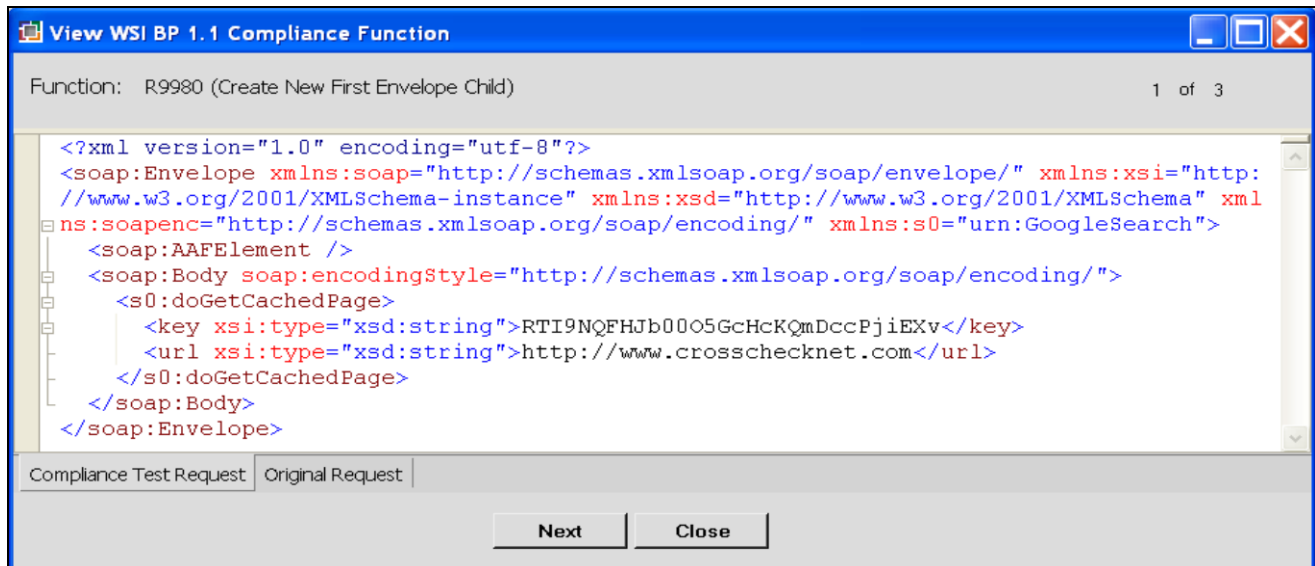


To see what modifications are going to be made on the defined base test request, you can click the document links.

3.1.1 SOAP Envelope Structure

[R9980](#) An ENVELOPE MUST conform to the structure specified in SOAP 1.1 Section 4, "SOAP Envelope"

This will bring up a viewer as shown below which shows the number of iterations which will be created and allow you to shift between the Attack Request and the Original Request to see what changes are going to be performed to the XML before it is submitted to the back-end service. Based on the profile, some mutations may be XML based, others may be HTTP protocol or header based.



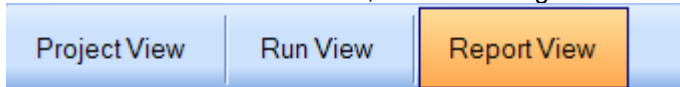
LOGGING AND REPORTING

Test suites store results in serialized XML files called log files. Each run mode stores files to the specified active directory. Viewing log files from test suite runs will provide access to report diagnostics specified to the active run mode. For example, when viewing logs and reports in QA mode, the reports are success based to assess the functional success or failure of test cases. In Performance mode, the reports provide analysis criteria for performance profile measurement statistics. In Compliance mode, the reports show which messages violated the WS-I Basic Profile 1.1 SOAP Messaging requirements. In Vulnerability mode, the report shows the AVD criteria that matched the response data set and the risk posture based on the matched criteria items.

Subtopics in this section include:

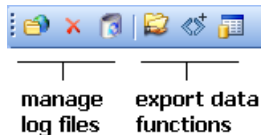
- [Generating Reports](#)
- [Report Export Formats](#)
- [Exporting Raw Log Data to File](#)
- [Exporting Log View Grid to Excel or HTML Table](#)
- [HTTP Response Error Code](#)

To view the results of the test suite run, click the navigation icon to go to the Logging and Reporting area.



The log files appear on the left pane organized by date. To view the information in a log file, select the entry. Contents will appear on the right screen based on the type of log file and run mode that was used to create the result set. Within each run mode there is a set of reports that can be generated from the log results.

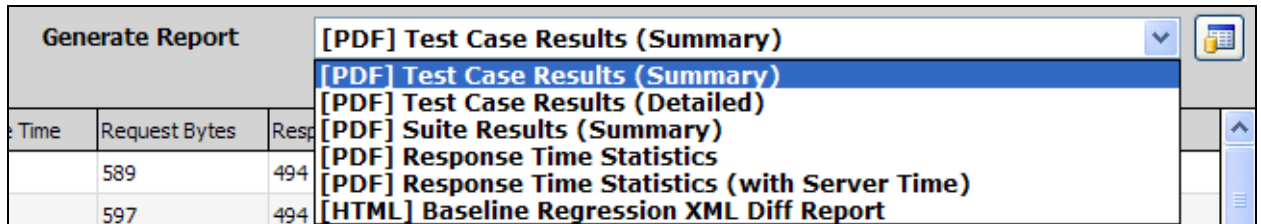
Result Log Toolbar



The log toolbar provides options for changing the active log folder for the current run mode, deleting the log file, or deleting a set of log files. There are also several export functions that allow you to extract all request and response data as individual files into a directory, export the log files as a normalized (no hashes) result set, and generate an HTML baseline XML diff report (exclusive to QA mode).

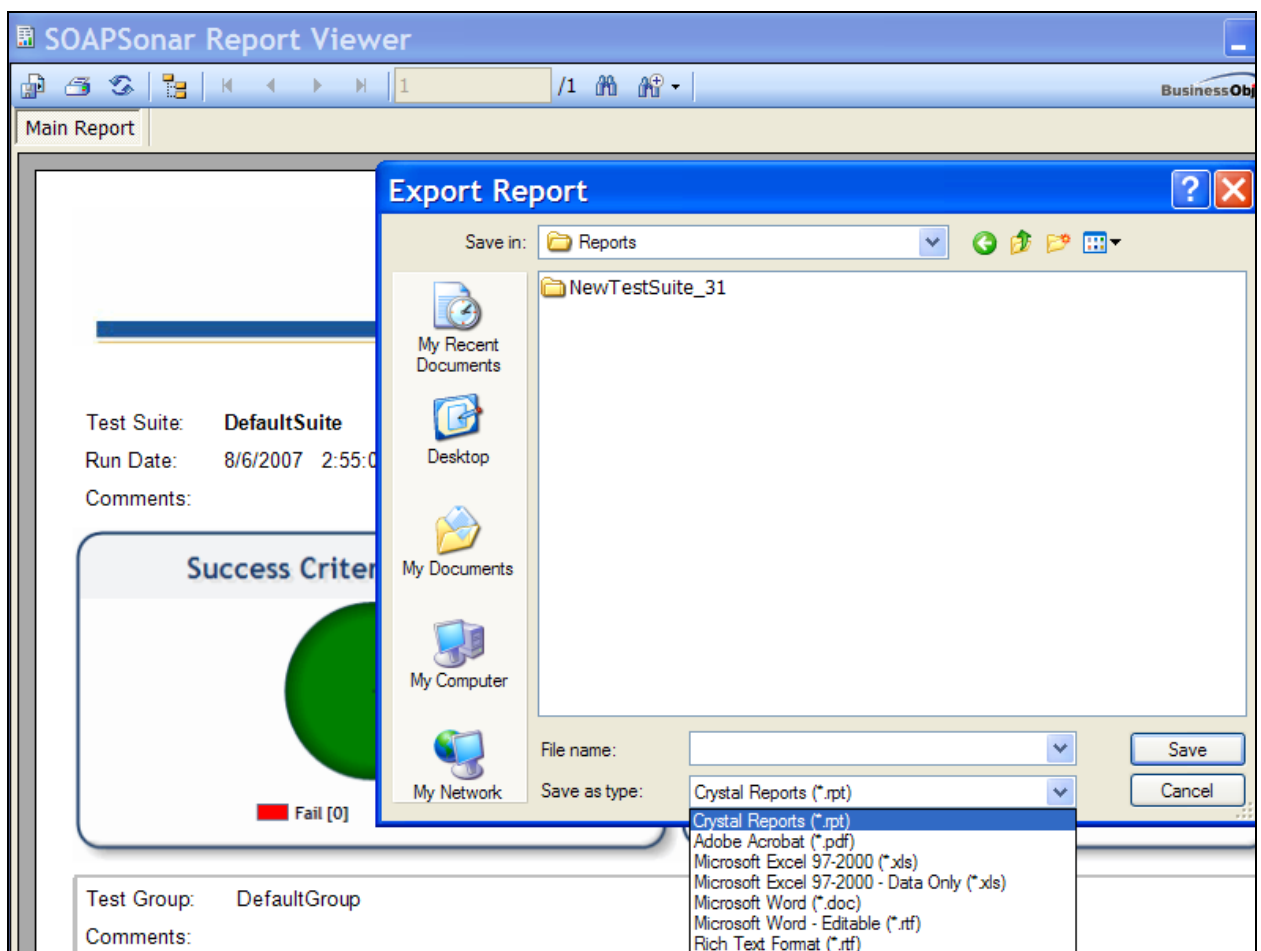
Generating Reports

When viewing log results reports can be generated in the report viewer. SOAPSonar uses Crystal Reports for rendering of reports in the report viewer which provide navigation among report pages as well as export formats for report data. To generate a report, click on the reports to select the report type and then press the Generate Report button to create the selected report and open the report viewer.



Report Export Formats

There are several formats for exporting reports including .pdf, .xls, .doc, .rtf, and .rpt. To export the report click on the top left button in the report viewer and then select the type of document to export the report to.



Exporting Raw Request and Response Data to File

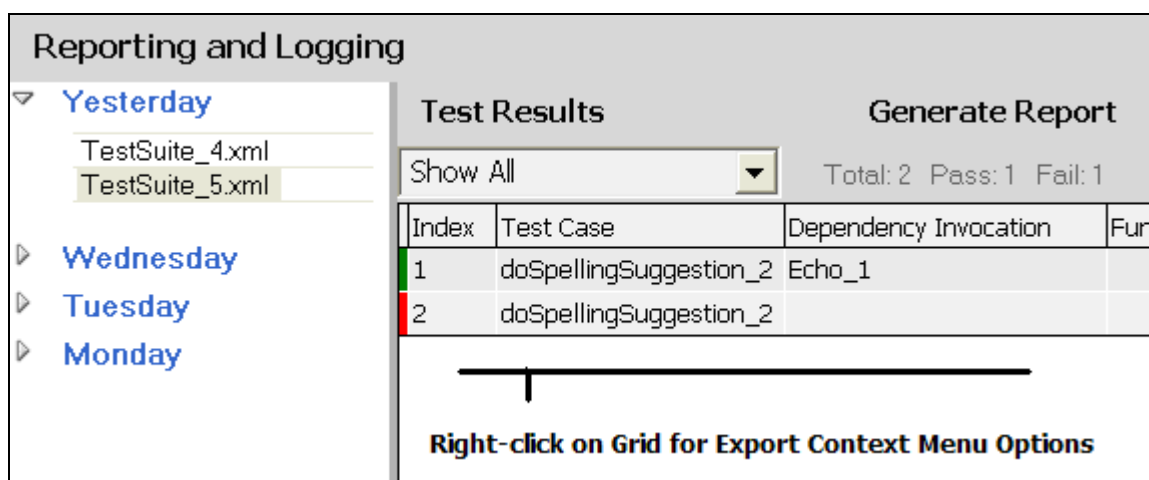
This option will extract each request and response message that was sent and received into the selected directory. A file will be created for each request and each response. You can choose whether to include the HTTP header information that was sent and received in the output files. Select this option from the log toolbar, or right-click on the log filename.

Exporting Raw Log Results to Normalized XML

By default, the log file results are stored in relational XML format to with hash mapping indexes. To export the result data in normalized (flat) format, select this option from the log toolbar, or right-click on the log filename. The exported results will match the current entries displayed in the summary grid, which are based on the Filter settings (“Show All”, “Show Matching”, and “Show Non-Matching”).

Exporting Log Data to CSV, Excel, or HTML Format

To export the log information as shown in the grid views, simply right-click and select the export option from the context menu of “CSV”, Excel, or HTML.



The screenshot shows the 'Reporting and Logging' window. On the left, there is a tree view with 'Yesterday' expanded, showing 'TestSuite_4.xml' and 'TestSuite_5.xml'. Below it are 'Wednesday', 'Tuesday', and 'Monday'. The main area is titled 'Test Results' and contains a 'Generate Report' button. Below the button is a dropdown menu set to 'Show All' and a summary 'Total: 2 Pass: 1 Fail: 1'. A table with the following data is displayed:

Index	Test Case	Dependency Invocation	Fun
1	doSpellingSuggestion_2	Echo_1	
2	doSpellingSuggestion_2		

Below the table is a horizontal line with a vertical tick mark pointing to it, and the text 'Right-click on Grid for Export Context Menu Options'.

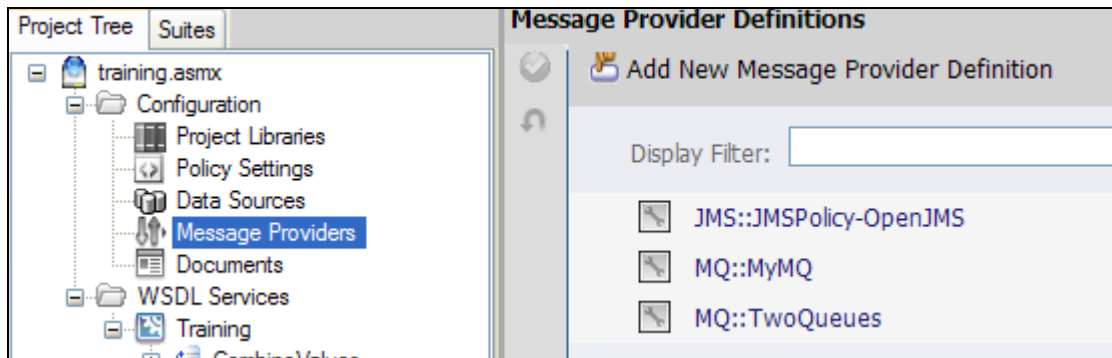
HTTP Response Error Codes

Requests failing for reasons outside standard HTTP response error codes (200-599) are reported in the log view and report views with the following HTTP response codes:

Code	Description
1	The name resolver service could not resolve the host name
2	The remote service point could not be contacted at the transport level
3	A complete response was not received from the remote server
4	A complete request could not be sent to the remote server
5	The request was submitted as a pipeline request, but the connection was dropped before the respective response was received
6	The request was aborted
7	The response received from the server was complete but indicated a protocol-level error
8	The connection was prematurely closed
9	A server certificate could not be validated
10	An error occurred in a secure channel link
11	The server response was not a valid HTTP response
12	The connection for a request that specifies the Keep-alive header was closed unexpectedly
13	An internal asynchronous request is pending
14	No response was received during the timeout period for a request

IBM MQ, Weblogic JMS, Tibco EMS, and Native JMS MESSAGE PROVIDERS

For IBM MQ, Weblogic JMS, Tibco EMS, or native JMS messaging, the policy configuration is located on the Message Providers section of the Configuration area.

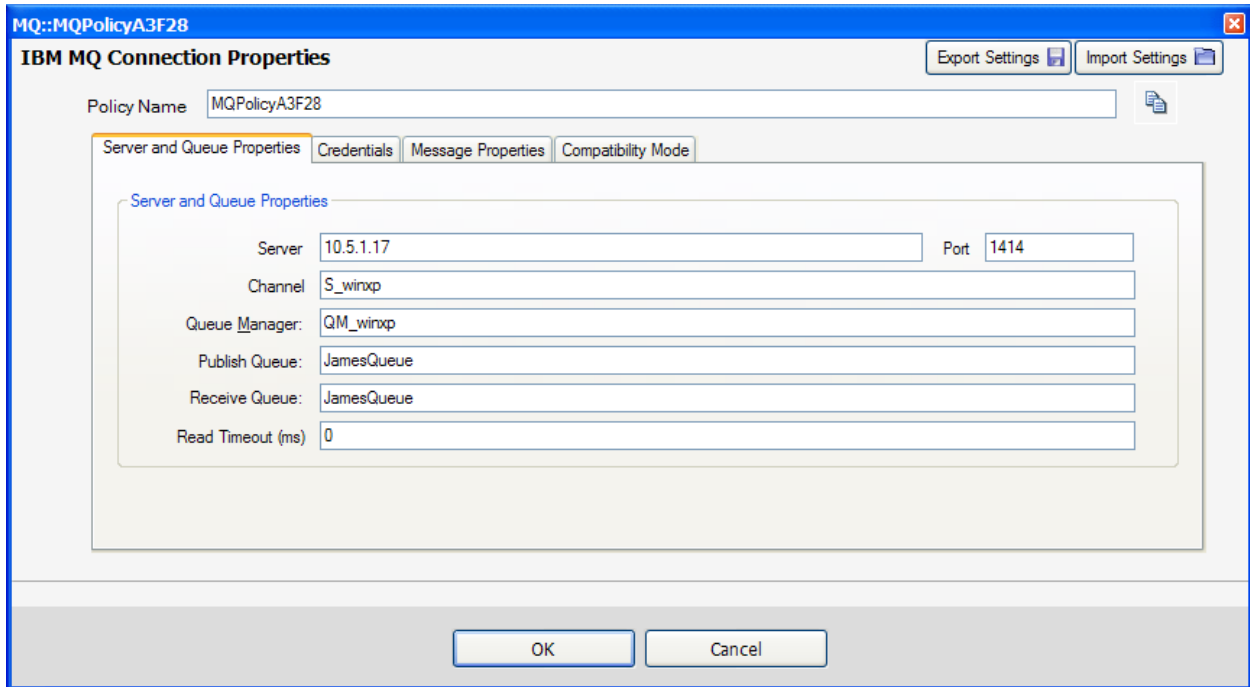


To create a new message provider policy, click on the “Add a New Message Provider Definition” menu and choose either “MQ Message Provider” or “JMS Message Provider”.

MQ Message Provider

The MQ message provider allows for native MQ protocol testing using the IBM MQ Client libraries. In order to send MQ messages, you must first install the IBM MQ Client on the machine where SOAPSonar is running. The IBM MQ client is freely available from the IBM web site.

Once you have the IBM MQ Client installed, you can create MQ policies to send and receive messages from MQ queues. The settings for an MQ policy are described below.



Policy Name:

The name used to reference the policy from the URI field in a test case

Server:

The host name of the MQ Queue Manager

Channel:

The MQ Channel where the Queue Manager is defined

Queue Manager:

The name of the Queue Manager where the queues are defined

Publish Queue:

The name of the Queue to send message to. Used if 2-way or 1-way publish options are selected.

Receive Queue;

The name of the Queue to receive messages from. Used if 2-way or 1-way receive options are selected

Read Timeout:

Amount of time to wait while listening for a response message on the receive Queue

Username:

If credentials are required, the username

Password:

If credentials are required, the password

Message Expiry:

The message property indicating whether the message has an expiry

Use Correlation ID:

When set, expects the response message correlation ID to match the originating message ID

Persistence

The persistence setting for the published message

Compatibility Mode

Version of the API to invoke

Oracle Weblogic JMS Message Provider (WLS)

The Oracle Weblogic JMS message provider allows for Weblogic JMS protocol testing using the native Weblogic API. In order to send WLS messages, you must first install the Weblogic .NET Client `WebLogic.Messaging.dll` on the machine where SOAPSonar is running. The Weblogic client is freely available from the Oracle web site, and can be obtained from a Weblogic installation.

Once you have the Weblogic Client installed, you can create WLS policies to send and receive messages from Weblogic queues. The settings for a Weblogic policy are described below.

The screenshot shows a Windows-style dialog box titled "WLS::WLSPolicy2CA34" with a subtitle "Weblogic WLS Connection Settings". At the top right, there are "Export Settings" and "Import Settings" buttons. Below the title bar, the "Policy Name" is set to "WLSPolicy2CA34". There are four tabs: "JNDI and Queue Properties" (selected), "Credentials", "Message Properties", and "Compatibility Mode". The "JNDI and Queue Properties" section contains several text boxes and a dropdown menu: "java.naming.factory.initial" is "weblogic.jndi.WLInitialContextFactory", "java.naming.provider.url" is "t3://127.0.0.1:7001", "Connection Factory:" is "CF_MACY", "Factory Type:" is a dropdown menu set to "Queue", "Publish Queue:" is empty, "Receive Queue:" is "JMS_DQ", and "Read Timeout (ms)" is "1000". At the bottom, there are "OK" and "Cancel" buttons.

Policy Name:

The name used to reference the policy from the URI field in a test case

Java.Naming.Factory.Initial:

This is the context factory to be used to establish the `InitialContext()` call

Java.Naming.Provider.Url:

The endpoint listener which provides the JNDI information

ConnectionFactory:

The name of the Connection Factory class

Factory Type:

The type of messaging exchange to be performed

Publish Queue:

The name of the Queue to send message to. Used if 2-way or 1-way publish options are selected.

Receive Queue;

The name of the Queue to receive messages from. Used if 2-way or 1-way receive options are selected

Username

Username for JNDI Context bind and Queue access

Password

Password for JNDI Context bind and Queue access

Read Timeout:

Amount of time to wait while listening for a response message on the receive Queue

Message Expiry:

The message property indicating whether the message has an expiry

Use Correlation ID:

When set, expects the response message correlation ID to match the originating message ID

Persistence

The persistence setting for the published message

Format

Choice of JMS Map Message format or JMS Text Message format

Compatibility Mode

Version of the API to invoke

Tibco EMS Message Provider

The Tibco EMS message provider allows for Tibco EMS protocol testing using the native Tibco API. In order to send EMS messages, you must first install the Tibco .NET Client `TIBCO.EMS.dll` on the machine where SOAPSonar is running. The Tibco .NET client is freely available from the Tibco web site, and can be obtained from a Tibco EMS installation.

Once you have the Tibco .NET Client installed, you can create EMS policies to send and receive messages from Tibco EMS queues. The settings for a Tibco policy are described below.

The screenshot shows a dialog box titled "EMS::EMSPolicyBFF1F" with the main heading "Tibco EMS Connection Settings". It features "Export Settings" and "Import Settings" buttons in the top right. The "Policy Name" field is filled with "EMSPolicyBFF1F". Below this are four tabs: "JNDI and Queue Properties", "Credentials", "Message Properties", and "Compatibility Mode". The "JNDI and Queue Properties" tab is selected, revealing a sub-section with the following fields: "java.naming.factory.initial" (empty), "java.naming.provider.url" (filled with "tcp://localhost:7222"), "Connection Factory:" (empty), "Factory Type:" (a dropdown menu set to "Queue"), "Publish Queue:" (empty), "Receive Queue:" (empty), and "Read Timeout (ms)" (empty). At the bottom of the dialog are "OK" and "Cancel" buttons.

Policy Name:

The name used to reference the policy from the URI field in a test case

Java.Naming.Factory.Initial:

This is the context factory to be used to establish the InitialContext() call

Java.Naming.Provider.Url:

The endpoint listener which provides the JNDI information

ConnectionFactory:

The name of the Connection Factory class

Factory Type:

The type of messaging exchange to be performed

Publish Queue:

The name of the Queue to send message to. Used if 2-way or 1-way publish options are selected.

Receive Queue;

The name of the Queue to receive messages from. Used if 2-way or 1-way receive options are selected

Username

Username for JNDI Context bind and Queue access

Password

Password for JNDI Context bind and Queue access

Read Timeout:

Amount of time to wait while listening for a response message on the receive Queue

Message Expiry:

The message property indicating whether the message has an expiry

Use Correlation ID:

When set, expects the response message correlation ID to match the originating message ID

Persistence

The persistence setting for the published message

Format

Choice of JMS Map Message format or JMS Text Message format

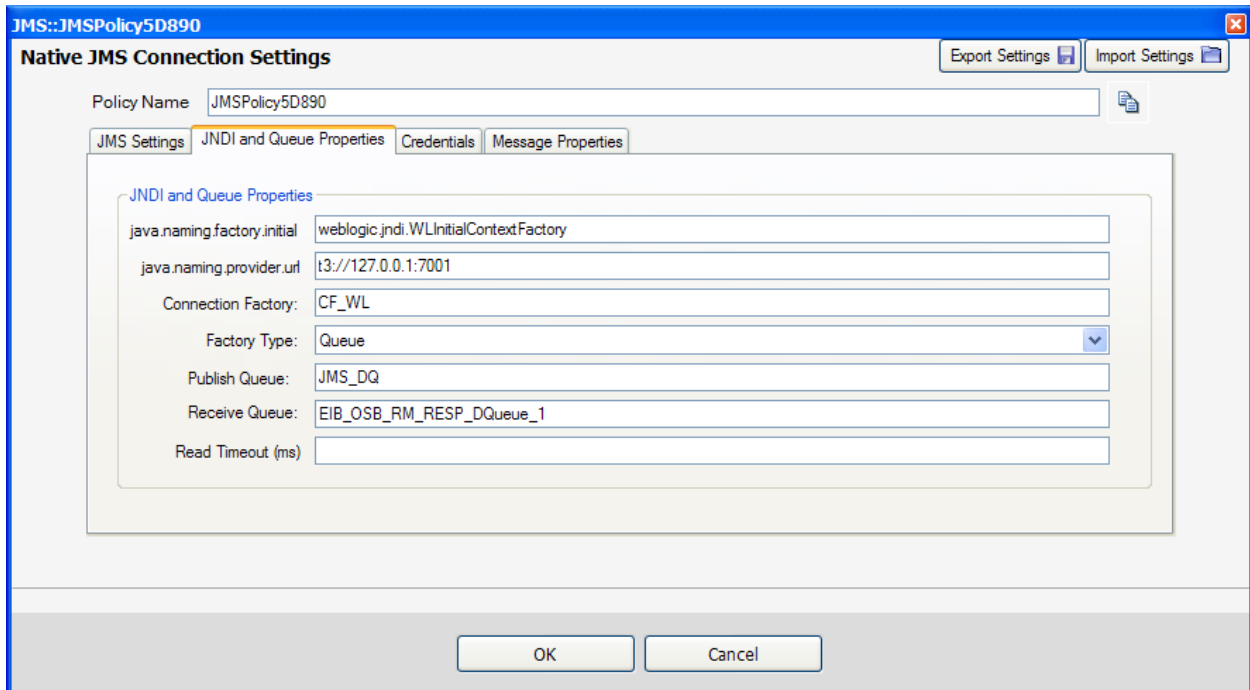
Compatibility Mode

Version of the API to invoke

JMS Message Provider

The JMS message provider allows for native JMS protocol testing using the JMS provider of your choice. In order to send JMS messages for your selected JMS provider, you must first install the provider JAR files on the machine where SOAPSonar is running. The JMS message provider uses JNDI to define the classes and connection information.

The settings for configuring a JMS message provider policy are defined below.



Policy Name:

The name used to reference the policy from the URI field in a test case

JVM Path:

Choose "Use Default" to use the active JVM based on the JAVA_HOME environment variable. Otherwise, choose "Specify" and enter the path to the jvm.dll directly.

JVM Provider Classpath:

The classpath for the chosen JMS provider JAR files. If multiple JAR files are required, separate them with a ";" delimiter. i.e. a.jar; b.jar; c.jar.

Enable Java Debugging:

Allows debug tracing of the Java Virtual Machine to diagnose connection or initialization issues

Java.Naming.Factory.Initial:

This is the context factory to be used to establish the InitialContext() call

Java.Naming.Provider.Url:

The endpoint listener which provides the JNDI information

ConnectionFactory:

The name of the Connection Factory class

Factory Type:

The type of messaging exchange to be performed

Publish Queue:

The name of the Queue to send message to. Used if 2-way or 1-way publish options are selected.

Receive Queue:

The name of the Queue to receive messages from. Used if 2-way or 1-way receive options are selected

Username

Username for JNDI Context bind and Queue access

Password

Password for JNDI Context bind and Queue access

Read Timeout:

Amount of time to wait while listening for a response message on the receive Queue

Message Expiry:

The message property indicating whether the message has an expiry

Use Correlation ID:

When set, expects the response message correlation ID to match the originating message ID

Persistence

The persistence setting for the published message

Format

Choice of JMS Map Message format or JMS Text Message format

COMMAND-LINE INTERFACE

Running test suites and generating reports can be performed using the SOAPSonar command-line interface utilities. Use of these command-line utilities provides the means to automate testing through integration with build environments or scheduling applications.

The command line interface is accessible from the installation directory and has the options shown below.

Running Test Suites From The Command-Line

To run a test suite from the command-line, use the runsuite.exe utility.

```
Usage: runsuite.exe [options]

options

List Suites
-l, --listsuites arg
    list the suites available in the referenced project file

Run Suite
-m, --runmode arg
    run mode (qa, performance, compliance, vulnerability)
-p, --projectfile arg
    project file
-t, --testsuite arg
    test suite to run
-g, --testgroup arg
    (optional) name of test group to run in specified test suite
-o, --logfile arg
    (optional) override the default result log filename
-u, --urloverride arg
    (optional) specify a new endpoint for all requests

Performance Mode Options
--clients arg
    (optional) override virtual client values
--duration arg
    (optional) override test duration
--iterations arg
    (optional) override test iterations
--rampup arg
    (optional) override test ramp up times (in seconds)
--rampdown arg
    (optional) override test ramp down times (in seconds)
--rampstrategy arg
    (optional) override test ramp strategy [linear, random]
--thinktime arg
    (optional) override test think times (in milliseconds)
--throttle arg
    (optional) override test throttle [on, off]
--throttlevalue arg
    (optional) override test throttle value
--throttlerate arg
    (optional) override test throttle rate [sec, min, hr]

Report Generation
```

```

-r, --reporttype arg
    (optional) generate the specified report
-f, --reportfile arg
    (optional) report output file
-e, --reportemail arg
    (optional) report email recipient
Automatic License Lease
-a, --autolease arg
    Type of License to Lease [S,A,P]
HP Quality Center Integration
--hqc arg
    (optional) write HP QC formatted results to specified file
--hqcreport true
    (optional) attach generated report to HP QC
--hqclog true
    (optional) attach xml result log to HP QC
--hqclogtask true
    (optional) attach tasklog to HP QC
JUnit Integration
--junit true
    (optional) product results in format parsed by junit harness (arg=true)

```

The following are a few sample commands:

```
runsuite.exe -l "C:\Projects\SSWebService.ssp"
```

List the suites available to be run in project file SSWebService.ssp

```
runsuite.exe -m qa -p "C:\Projects\SSWebService.ssp" -t TestSuite1
```

Run the test suite named "TestSuite1" in QA mode from project SSWebService.ssp

```
runsuite.exe -m qa -p "C:\Projects\SSWebService.ssp" -t TestSuite1 -u
http://10.10.1.1:8088/newservice.asmx
```

Run the test suite named "TestSuite1" in QA mode from project SSWebService.ssp and redirect all tests to the new endpoint <http://10.10.1.1:8088/newservice.asmx>

Generating Reports from the Command-Line

To generate a report from the command-line, use the `genreport.exe` utility.

```

Usage: genreport.exe [options]

options

-l, --listreports arg
    list the types of reports available for the result file

-r, --reporttype arg
    type of report
-o, --logfile arg
    test result file
-f, --reportfile arg
    report filename
-e, --emailreport arg
    email report to address

```

<p>-s, --emailsubject arg email subject for emailed report</p> <p>-b, --emailbody arg email body text for emailed report</p>
--

The following are a few sample commands:

genreport.exe -l "log/QA/TestSuite.xml"

list the set of reports allowed for the result file TestSuite.xml

genreport.exe -o "log/QA/EchoSuite_10.xml" -t testcasesummary -f "c:\zzz\summaryreport.pdf"

generate the qa mode test case summary report from the result file TestSuite.xml and write the results to c:\zzz\summaryreport.pdf

genreport.exe -o "log/Perf/abc.xml" -t tps -f "c:\zzz\tpsreport.pdf" -e myemail.company.com -s "Test Subject for Email" -b "Email Body Text"

generate the performance mode test case tps report from the result file abc.xml, write the results to c:\zzz\tpsreport.pdf, and email the results to myemail.company.com with the specified email title and subject

SMARTCARD INTEGRATION

SOAPSonar provides the ability to use keys from a SmartCard to perform digital signatures, encryption, decryption, and SSL X509 mutual authentication. SOAPSonar provides a native integration with A.E.T. SafeSign Client software to dynamically access the digital keying information on the card.

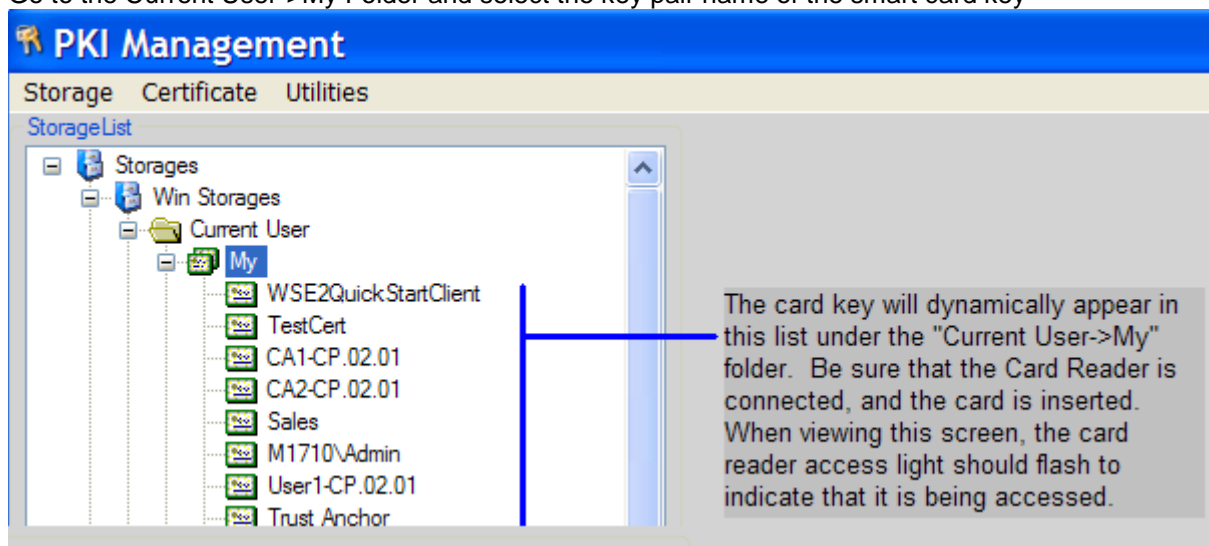
Setup for SmartCard Integration Support

SOAPSonar will recognize the existence of A.E.T. SafeSign Client on the machine where it is running and integrates natively through the API to provide seamless access to the smartcard key pair. The first requirement for SmartCard support is to install the A.E.T. SafeSign Client. For more information about how to obtain and install the A.E.T. SafeSign Client, please visit <http://www.aeteurope.com>.

Using a SmartCard Key for Signing, Encryption, or Decryption

Follow the steps below to use a SmartCard Key for digital signatures, encryption, or decryption:

- 1) Ensure A.E.T. SafeSign Client has been installed
- 2) Attach the card reader
- 3) Insert the card key
- 4) Go to the Test Case node in SOAPSonar and click on the Request Tasks tab
- 5) Create a new Signature, Encryption, or Decryption Task and click on the key icon to select the Key Pair
- 6) Go to the Current User->My Folder and select the key pair name of the smart card key

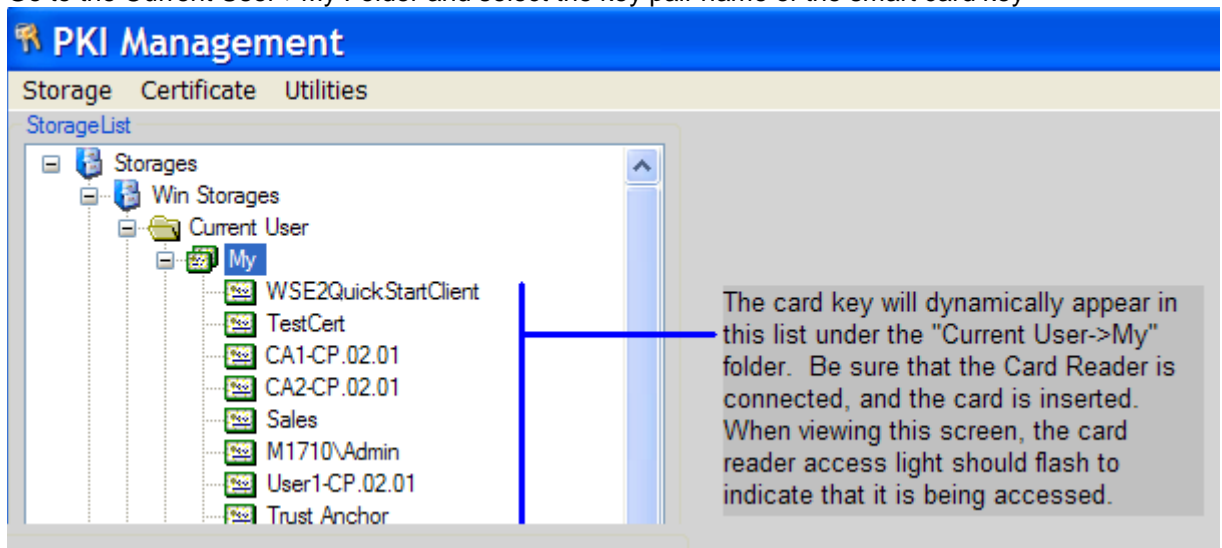


Using a SmartCard Key for SSL X509 Mutual Authentication

Follow the steps below to use a SmartCard Key for SSL X509 Mutual Authentication:

- 1) Ensure A.E.T. SafeSign Client has been installed
- 2) Attach the card reader
- 3) Insert the card key
- 4) Go to the Test Case node in SOAPSonar and click on the Authentication tab. If you want to create a global policy for authentication, instead click on the Policy node under the configuration tab and navigate to the Authentication tab.

- 5) Under the SSL authentication section, click on the key icon to select the Key Pair
- 6) Go to the Current User->My Folder and select the key pair name of the smart card key



HP QUALITY CENTER INTEGRATION : EMBEDDED

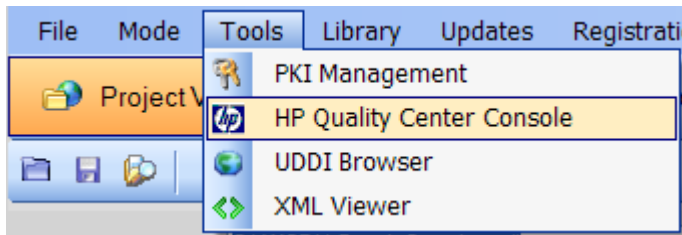
SOAPSonar provides native integration with HP Quality Center allowing test run information to be published directly into HP Quality Center and tests can be run from either within a Quality Center project, or separately from the SOAPSonar console or command-line interface. SOAPSonar also provides an HP Quality Center console allowing browsing Test Instances, Test Sets, Defects, and Attachments. SOAPSonar projects can now be managed and stored directly within an HP Quality Center project.

SOAPSonar Embedded Integration Features

The embedded HP Quality Center features provide the means to login to an HP Quality Center project on the project server and access the Test Instance, Test Plan, and Defect, and Attachment information. To use the embedded HP Quality Center features, you will first need to access the HP Quality Center server from a web browser on the machine where SOAPSonar is running. This process will ensure that the required OTAClient.dll library is copied and registered locally on the machine. Once you have accessed the HP Quality Center project, you will then need to copy the OTAClient.dll file into the SOAPSonar installation directory. After that you will have access to all of the embedded HP Quality Center features.

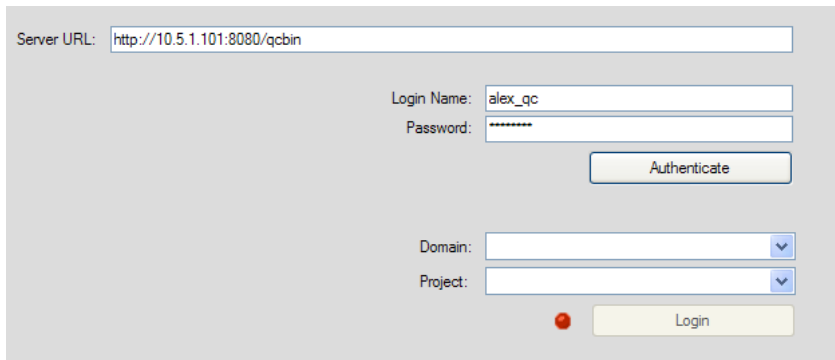
HP Quality Center Console

To access an HP Quality Center Project information, go to the **Tools->HP Quality Center Console**. This will allow you to login and view project information.



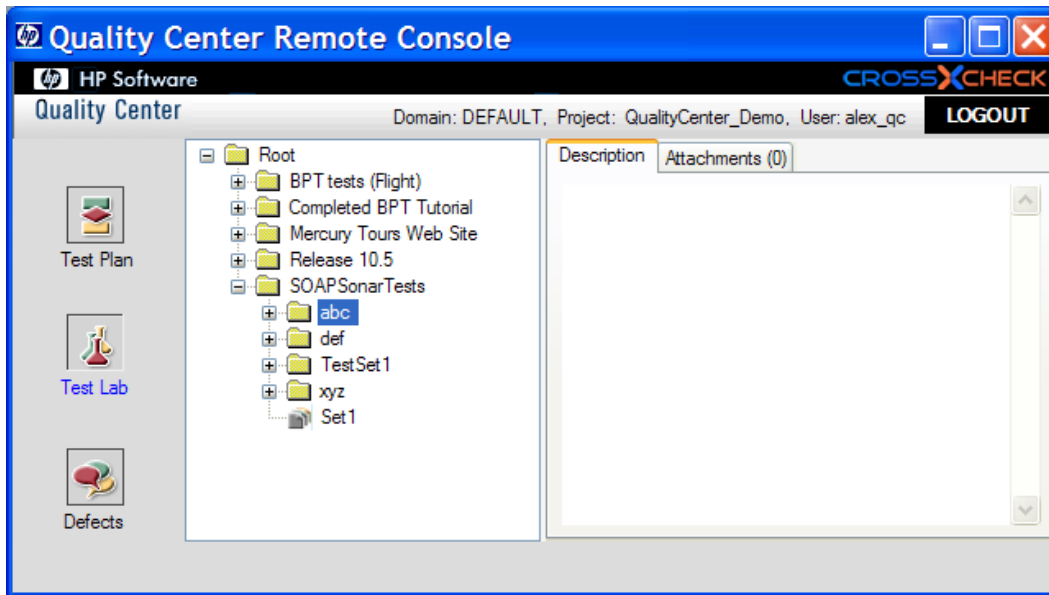
Login Screen

To access an HP Quality Center project, the Server URL, username, and password must be provided. The Server URL is the same HTTP endpoint that was used to access the HP Quality Center project via a web browser from the client machine. It is usually in the format **http://server:port/qcbin**.

A screenshot of the HP Quality Center Console login screen. It features a 'Server URL' field with the value 'http://10.5.1.101:8080/qcbin'. Below this are fields for 'Login Name' (containing 'alex_qc') and 'Password' (masked with asterisks). An 'Authenticate' button is positioned below the password field. At the bottom, there are 'Domain' and 'Project' dropdown menus, and a 'Login' button. A small red error icon is visible below the 'Login' button.

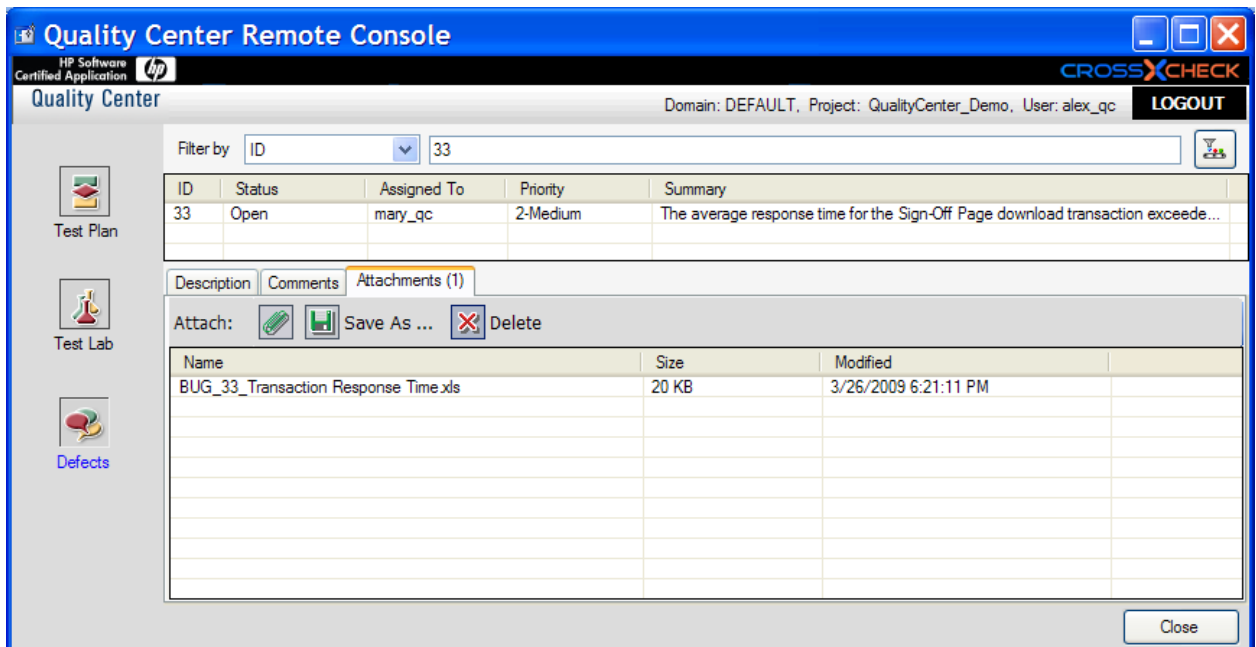
Project Viewer

Once the credentials have been provided and the corresponding Domain and Project selected, the project information will be shown for the Test Plan, Test Lab, and Defects.

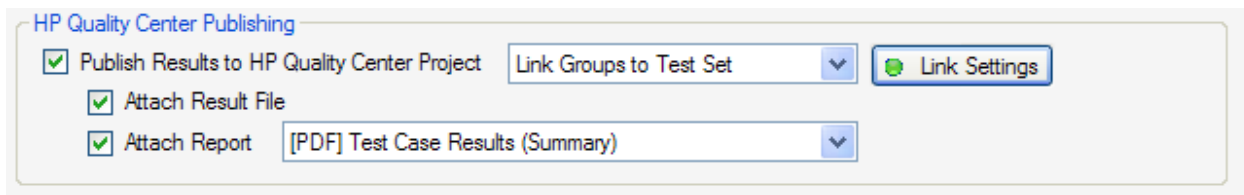


Uploading and Downloading Files Directly To HP Quality Center Server

Local files can be stored to the HP Quality Center project on the server. To upload files to the server, select on a project object that supports attachments. These include Test Instance, Test Folder, Test Set, and Defects. Each of these screens will show an attachment tab. Selecting the attachment tab will provide the menu options to upload or download attachments. Attachments can be any file type, including SOAPSonar project files or XML result log files.



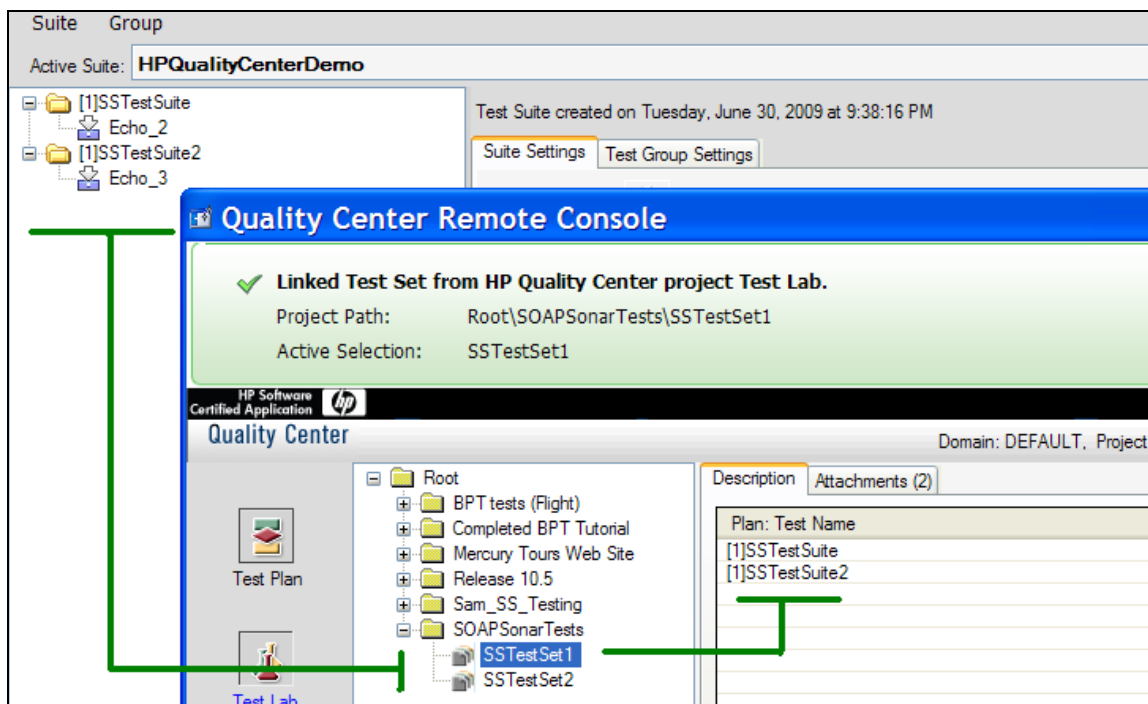
Publish Option #2: Link Groups to Test Set



With this option selection, test runs are segmented by Test Suite Group name and then mapped to each Test Instance associated with the target Test Set. To configure SOAPSonar to automatically publish test results to HP Quality Center as a new Test Run, then go to **Run View** and click on the **Publish Results to HP Quality Center** checkbox and then click on the **Link Settings** button. Enter the credentials for accessing the Quality Center project, then navigate to the **Test Lab**, and **Test Set** that corresponds to the Test Set where you want to map the Test Suite Group results to the Test Instances that have been defined for the Test Set.

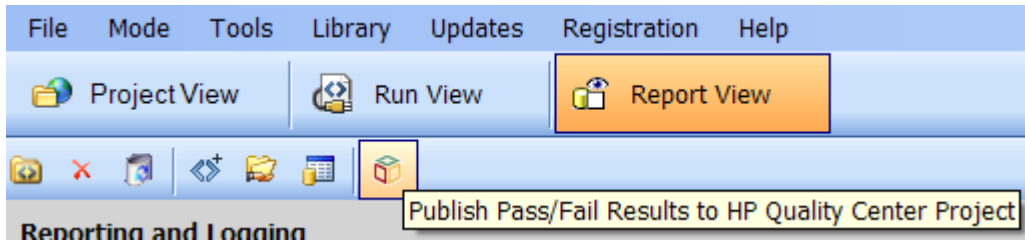
Optionally, you can also enable the result log to be attached to the HP QC TestSet object.

Optionally, you can also enable an automatically generated report to be attached to the HP QC TestSet object..



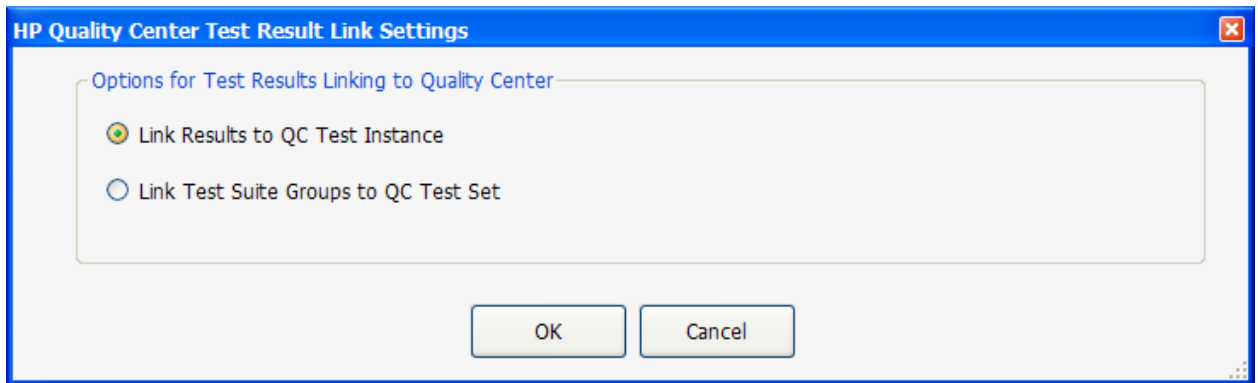
Manually Publish Test Results to HP Quality Center

SOAPSonar test results can be published to HP Quality Center as a test run manually from a previous run log file. To manually publish test results from a previous test run to Quality Center, go to Report View, click on a result file, then click the Publish to Quality Center button.

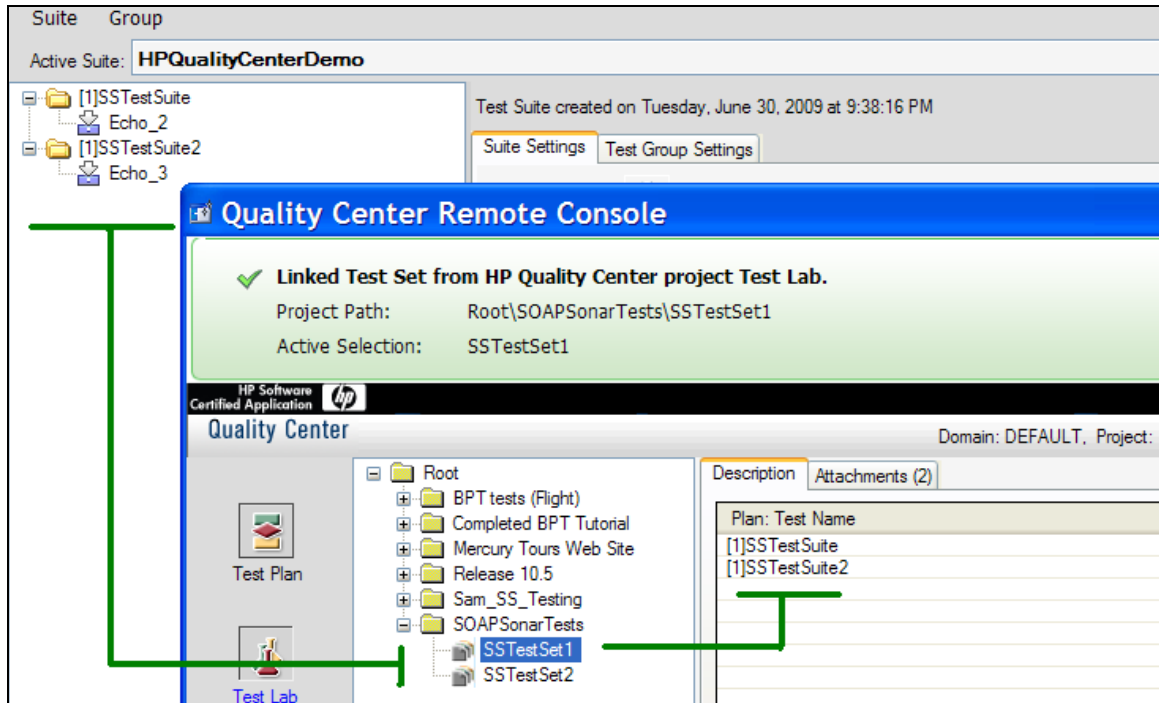


There are 2 settings you can choose for test run publishing. The first option links all test results to a selected Test Instance. The second option links test results based on the Suite Test Group to a target Test Set with associated Test Instances.

Publish Option #1: Link Results to Test instance



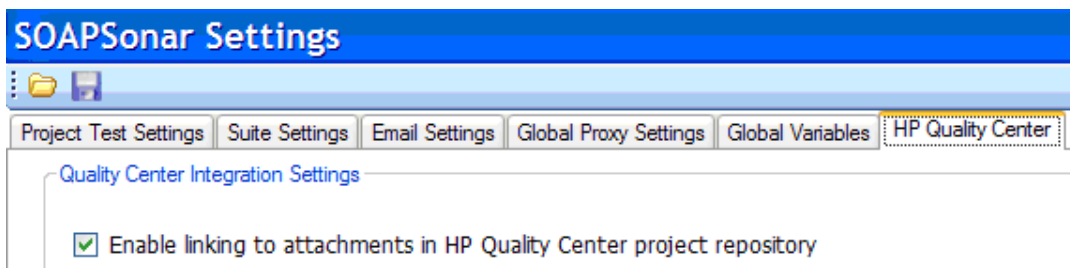
With this option selection, test runs are allocated to a Test Instance that has been associated with a Test Set. Enter the credentials for accessing the Quality Center project, then navigate to the **Test Lab, Test Set, and Test Instance** that corresponds to the Test Instance where you want to publish the test results as a new Test Run.



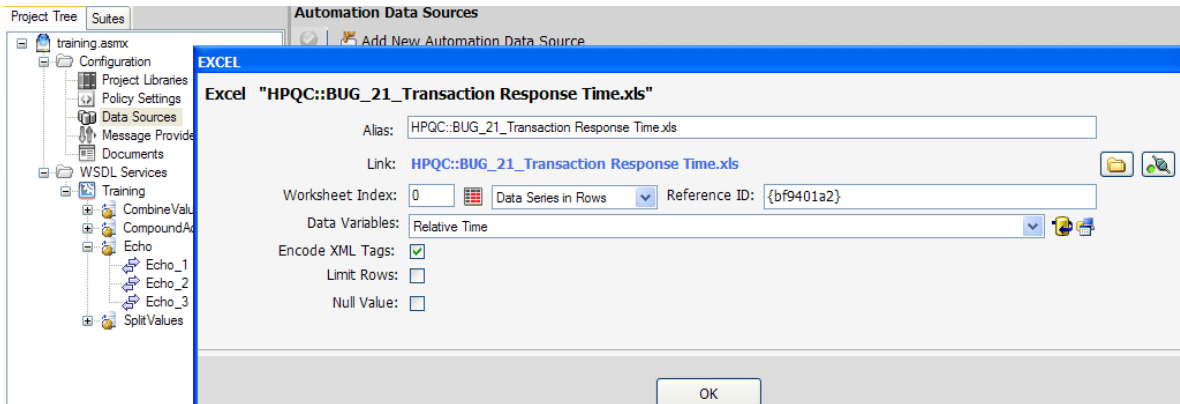
Dynamically Link to Quality Center Attachments for SOAPSonar Data Sources

SOAPSonar can be configured to dynamically link to a CSV or Excel file attachment in an HP Quality Center project for use as an Automation Data Source.

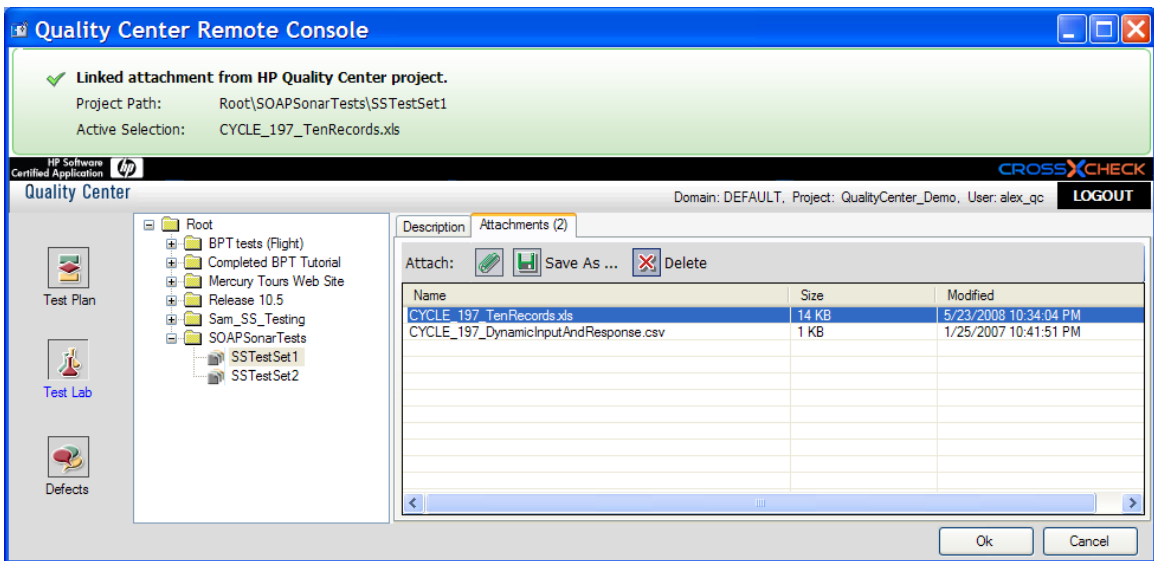
To enable the ability to perform dynamic file linking, go to the Settings and Preferences from the File menu and select the HP Quality Center tab. Click on the checkbox for attachment linking.



Once the setting above is enabled, when you are viewing file based data sources (CSV and Excel), the HP QC link button will appear next to the browse button. Click this button to browse an HP QC project for an attachment to link as the target data source.



Once an attachment is selected from either a Test Plan, Test Lab, or Defect object the link indicator at the top of the screen will appear with a check indicating the link has been established.



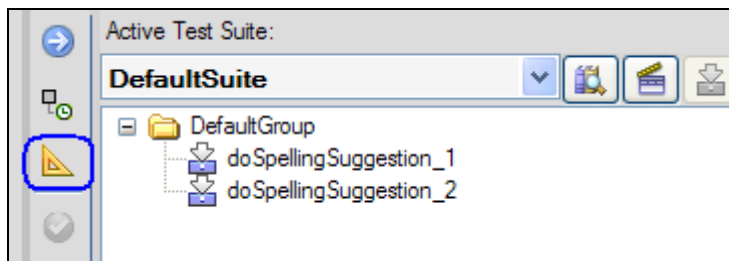
HP QUALITY CENTER INTEGRATION : VAPI-XP-TEST

SOAPSonar Automation and Platinum Editions provide the ability to integrate with HP Quality Center to run tests from within Quality Center and have test results reported back into Quality Center for bug tracking, workflow analysis, and other standard Quality Center features. HP Quality Center integration is supported for QA mode tests where results are in pass/fail format.

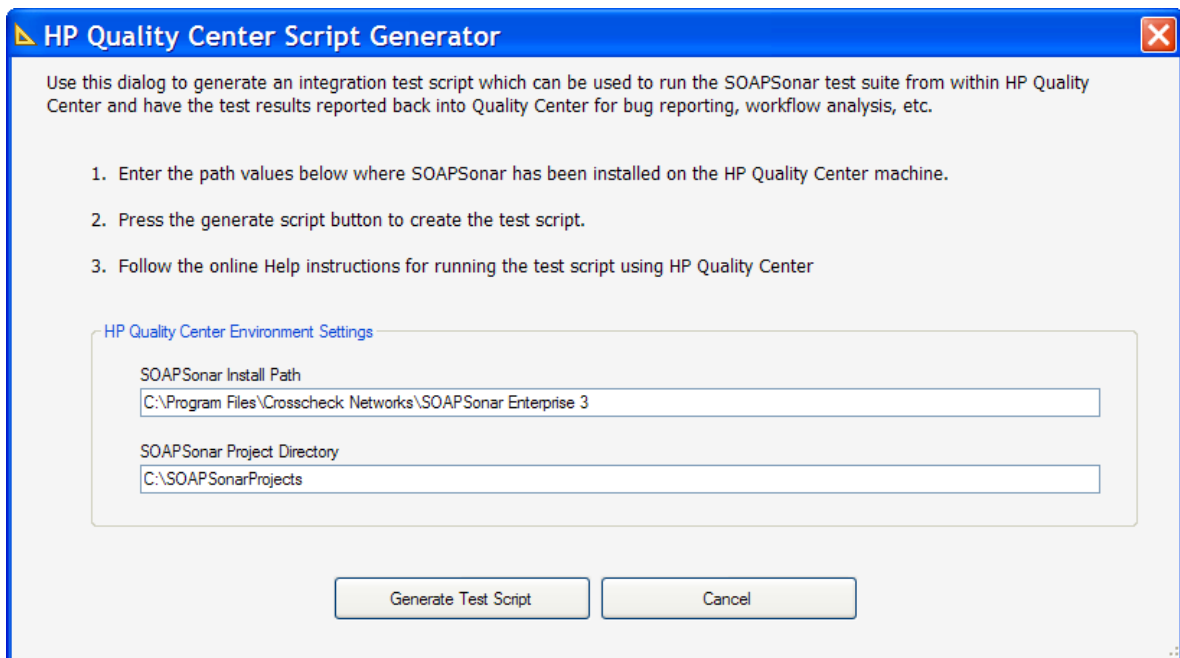
SOAPSonar Generated HP Quality Center Test Script

SOAPSonar automatically builds the HP Quality Center VAPI-XP-TEST VBScript for any test suite. This test script is then used by HP Quality Center to invoke SOAPSonar and run the test suite and report the test suite results as a test run.

To create the HP Quality Center test script from within the SOAPSonar interface, choose the active test suite to run and then click the icon to open the HP Quality Center script generator.

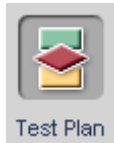


Scripts can be generated on SOAPSonar instances which are not running on the HP Quality Center machine, but the test themselves are run from the SOAPSonar instance installed on the Quality Center machine. To generate the test script with the applicable path settings, enter the SOAPSonar install path and directory where the SOAPSonar project file is stored for the HP Quality Center machine. If you are unsure, these values can be edited later directly in the script. Click the "Generate Test Script" button to proceed.

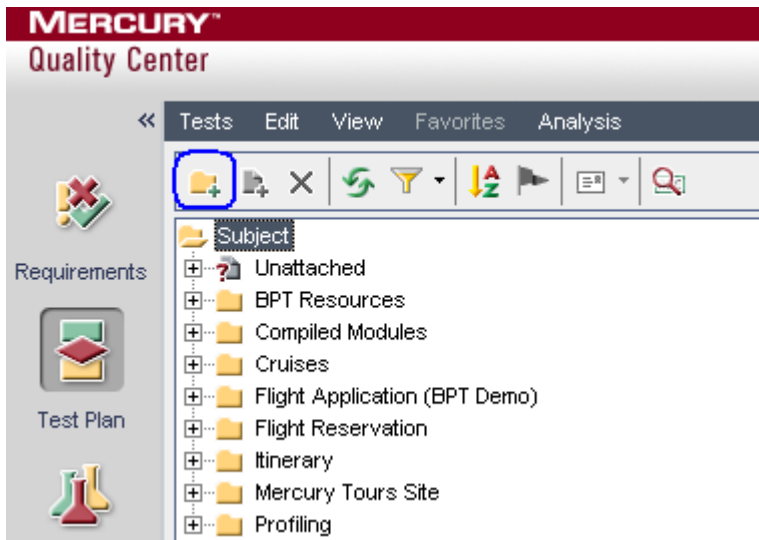


Integrate the SOAPSonar Test Script into HP Quality Center

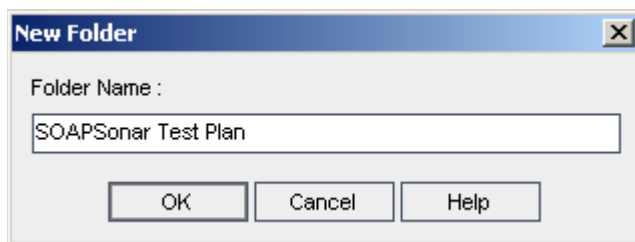
1. After logging in to HP Quality Center, click the Test Plan button from the left panel. The Test Plan area allows you to set up your automated test and custom scripts for Quality Center.



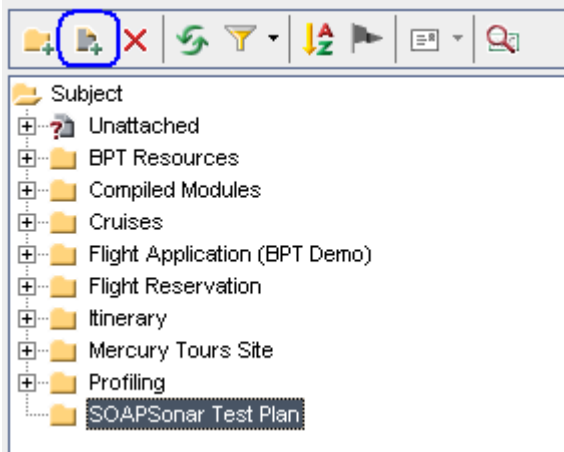
2. In the Test Plan Tree that appears, click the "New Folder" toolbar button.



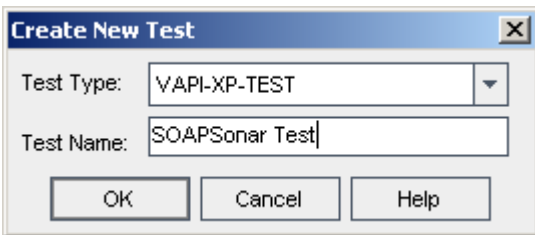
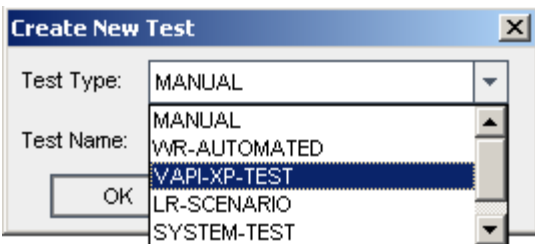
3. Enter a folder name. This example uses "SOAPSonar Test Plan" for the folder name.



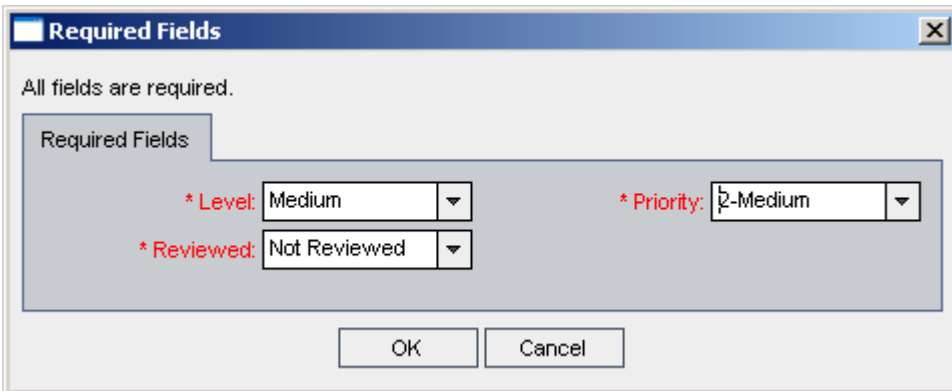
4. Highlight the newly created folder and then click the "New Test" toolbar button.



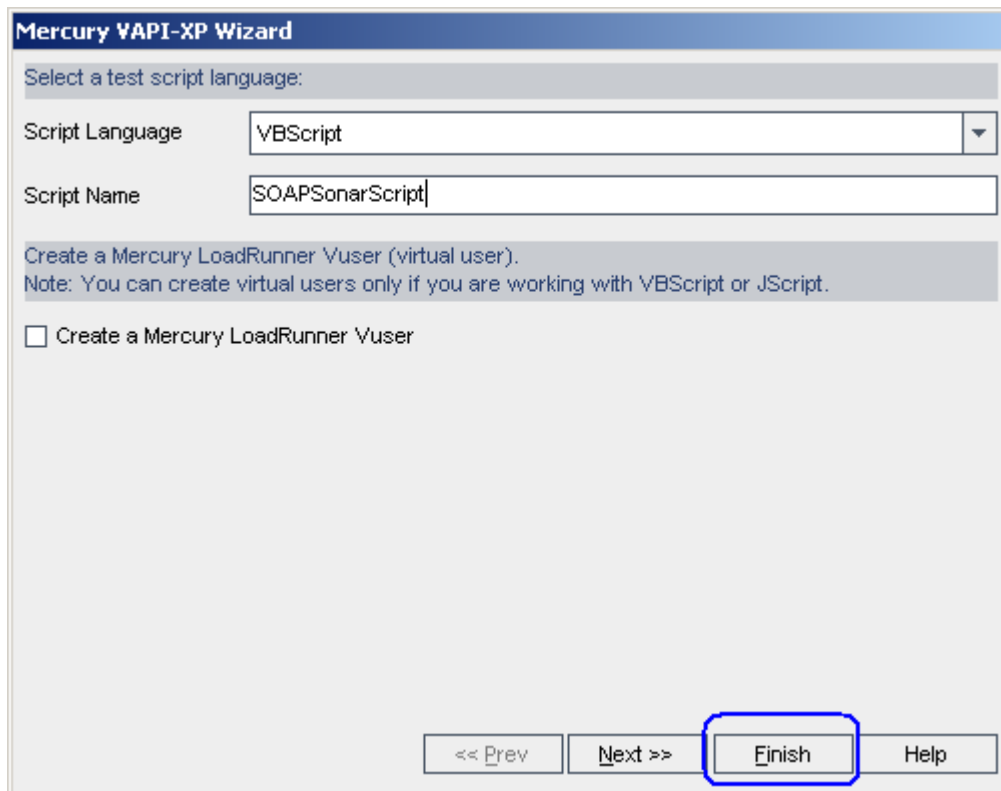
5. In the Create New Test dialog, select VAPI-XP-TEST as the test type and enter a test name. This example uses “SOAPSonar Test” as the test name. Press OK.



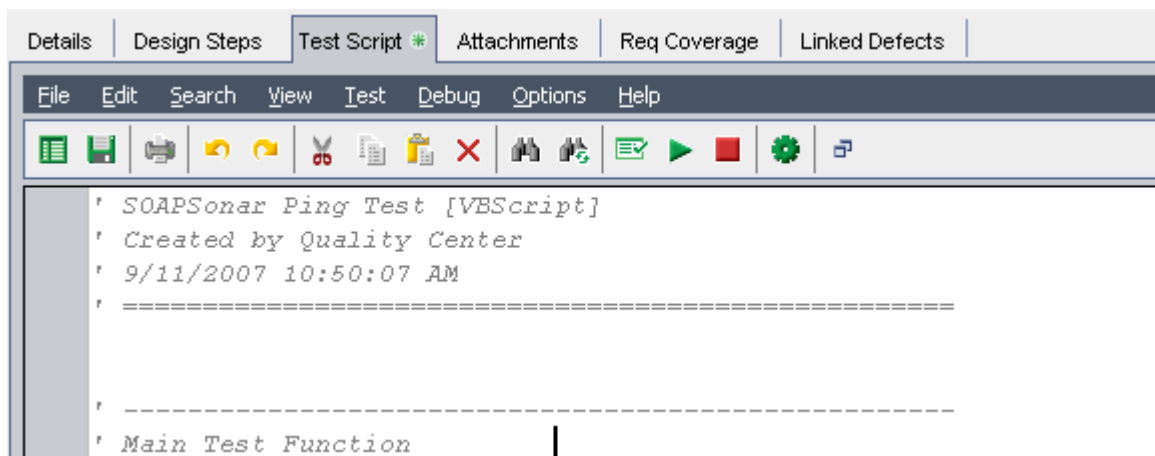
6. Choose the Level, Review Status, and Priority for this test from the available options. Press OK.



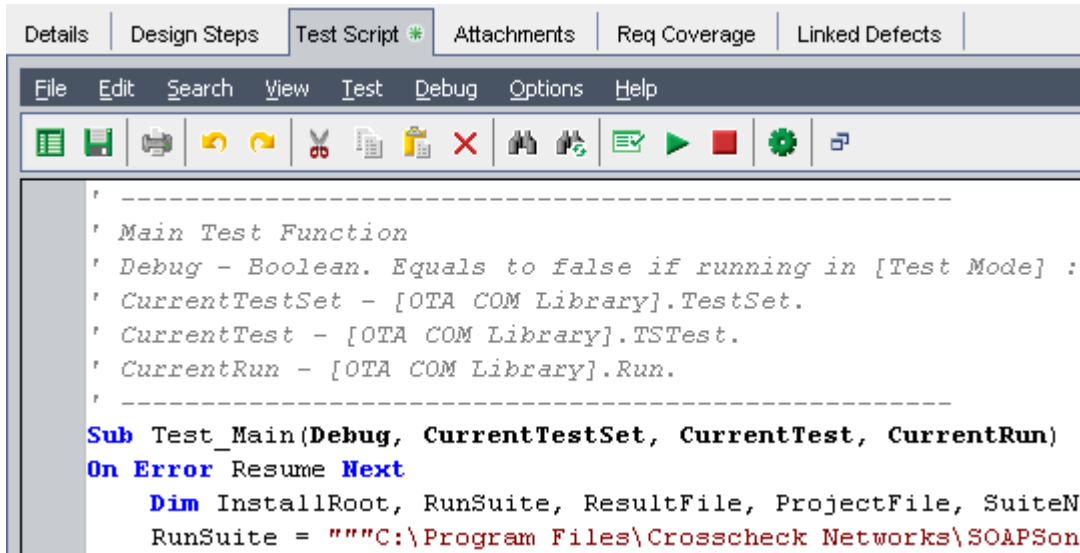
7. Choose VBScript as the script type and enter a script name. This example uses "SOAPSonarScript" for the script name. Click Finish



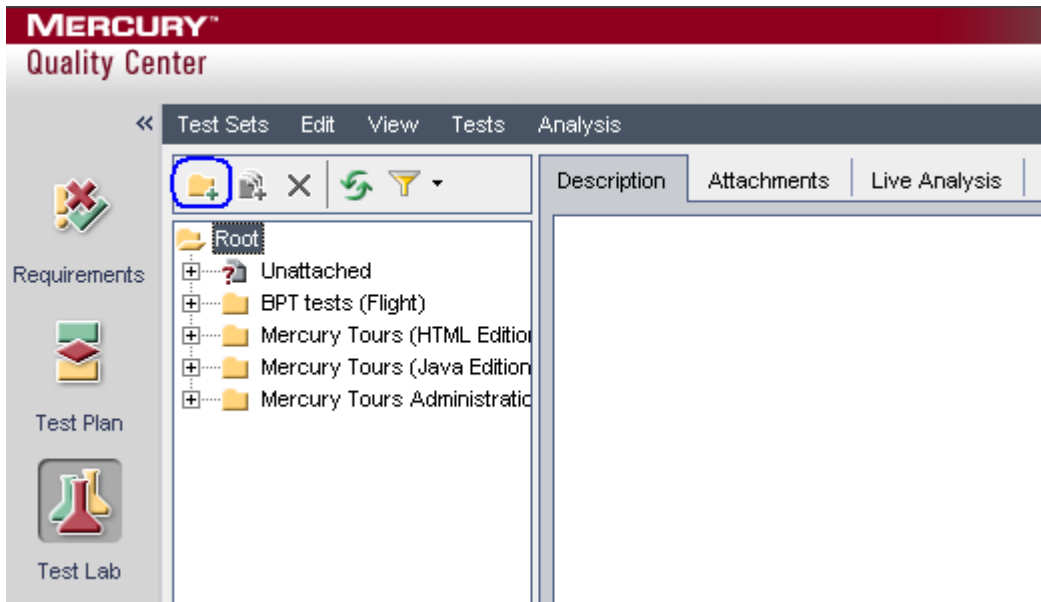
8. Select the Test Script tab. A default script template will have been created from the steps above. Press CTRL-A to select all of the text and then press the DELETE key to delete it.



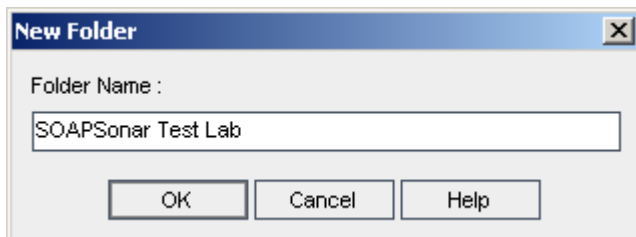
9. Open the SOAPSonar HP Quality Center test script saved from the [Generate Test Script](#) section above. Copy the contents of the SOAPSonar script file into the Test Script area.



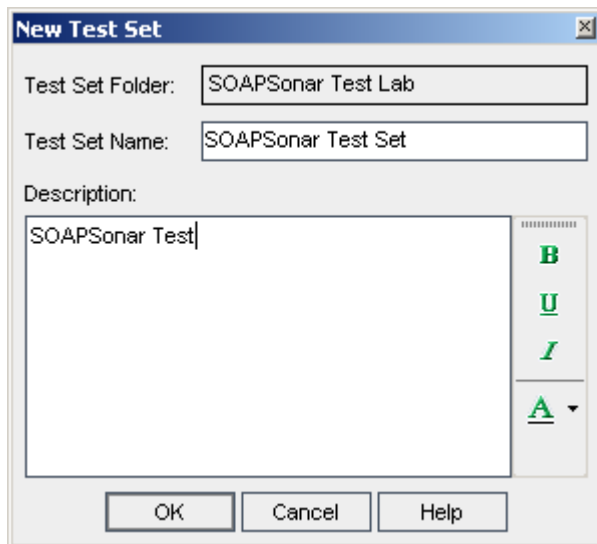
- Verify that all of the paths and filenames point to the proper locations for the current machine then click the Save button.
- Select the Test Lab button on the left. The Test Lab area allows you to combine the tests to run interactively.



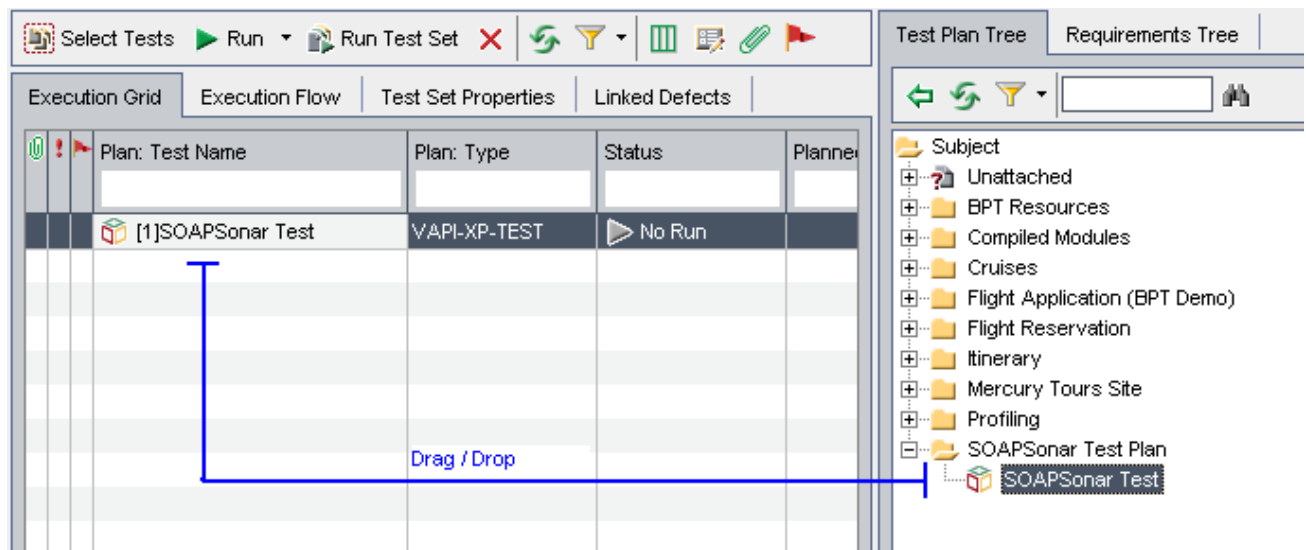
- Create a new Folder by clicking on the New Folder toolbar button and enter a folder name. This example uses "SOAPSonar Test Lab" for the folder name.



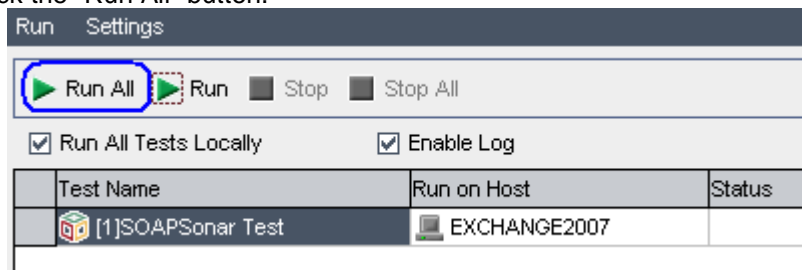
13. Click on the "New Test Set" toolbar button and enter a test set name a description for the test. This example uses "SOAPSonar Test Set" as the test set name.



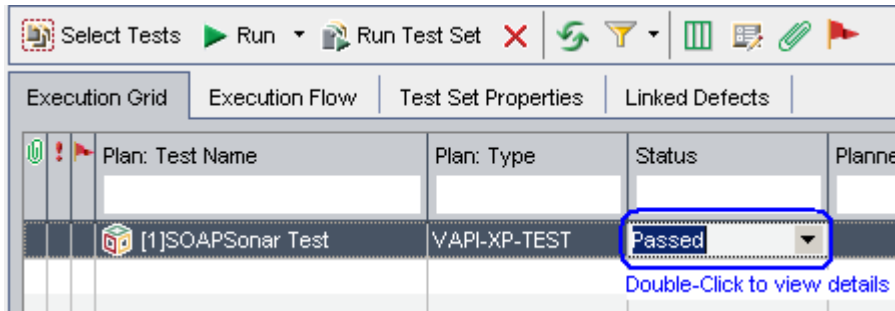
14. On the right side of the screen you will see the test plan folder created earlier. Within this folder is the newly created test. Drag this test to the Execution grid.



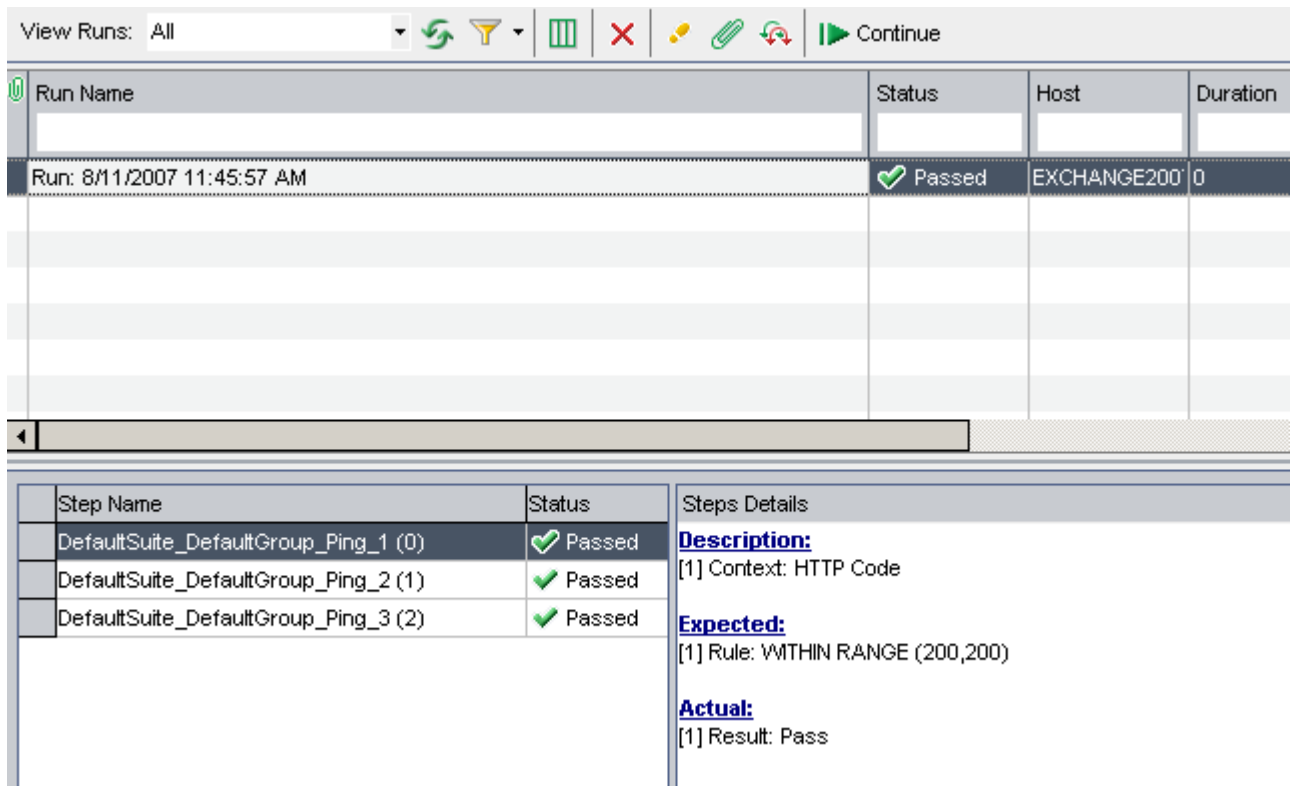
15. To run the test, click the Run button, check the "Run all Test locally" and "Enable Log" checkboxes, then click the "Run All" button.



16. Upon completion of the test there will be an overall Pass or Fail indication. Double click on the status to view the detailed test step results that were reported back from SOAPSonar into the Quality Center panel.



17. Each test step will be displayed with the result.



18. Double-click on the status column for a detailed view of the Description, Expected, and Actual results. SOAPSonar test results and criteria rule evaluation are reported back to Quality Center as follows:

Description	Success Criteria Evaluation Function
Expected	Parameters of Success Criteria Evaluation Function
Actual	Results of the Success Criteria Evaluation for this Test result

For tests which have multiple criteria rules, each criteria evaluated will be preceded by an [x] indicator where x=1, 2, ... allowing correlation of each rule with the rule evaluation results.

JUNIT INTEGRATION

SOAPSonar Automation and Platinum Editions provide the ability to integrate with Junit and Java to run tests from within the JUnit framework and have test results reported back in standard Junit format for tracking, workflow analysis, and other standard JUnit features. Standard Java programs can also be used to wrap the SOAPSonar classes for integrating and invoking SOAPSonar from within other Java classes.

SOAPSonar Junit Class Wrapper

Allows running SOAPSonar test suites from within java program or junit harness.

Files (located in the **junit** subfolder):

- RunSuite.java - Sample source code to show how to invoke a SOAPSonar test suite and set the SOAPSonar install directory
- RunSuiteTest.java - Sample source code which gets results of individual tests within the selected SOAPSonar test suite
- SoapSonarTestSuite.java - Source code of JUnit framework extension for SOAPSonar
- SOAPSonarTestCase.java - Source code Junit framework extension for SOAPSonar
- soapsonar.jar - Compiled Junit framework extensions to invoke SOAPSonar tests
- junit-4.4.jar - Junit framework

Usage:

```
java -jar soapsonar.jar: <projectfile> <testsuite>
```

SERVICE SIMULATION USING CLOUDPORT

Crosscheck Networks created the CloudPort product to address the full range of web services simulation and client diagnostics features to compress the deployment lifecycle and allow parallel service and client development. With point-and-click service simulation, client development can be performed in parallel to service development and thus reduce the total project time.

Additional capabilities provide the means to validate the functional quality of the client requests, ensure that requests follow corporate best practices. CloudPort provides the portable framework that allows service simulations to be created and shared among team members to help ensure consistent development practices and techniques.

CloudPort can be used as a means to reduce sharing of IT and network resources for trading partner integration testing or new business unit integration. Simulations are portable which allows for integration testing locally before requiring shared lab time, firewall accessibility, and ultimately production server access.

Launching CloudPort Runtime Simulation Player

Saved CloudPort simulations can be played on an unlimited number of machines using the free runtime player. This allows playing any simulation on any number of machines after the simulation project rules have been defined and stored in a re-distributable CloudPort simulation project file (.ssp) using CloudPort Platinum Edition, or CloudPort Standard Edition to author the simulation rules.

Canned simulations are provided for free for the following types of simulation:

- Echo Service
- Fault Service
- Canned Response Service

To launch the CloudPort Runtime Player, choose **Simulation->Launch CloudPort Service Simulator** from the main menu.

WSDL SCORING AND GOVERNANCE

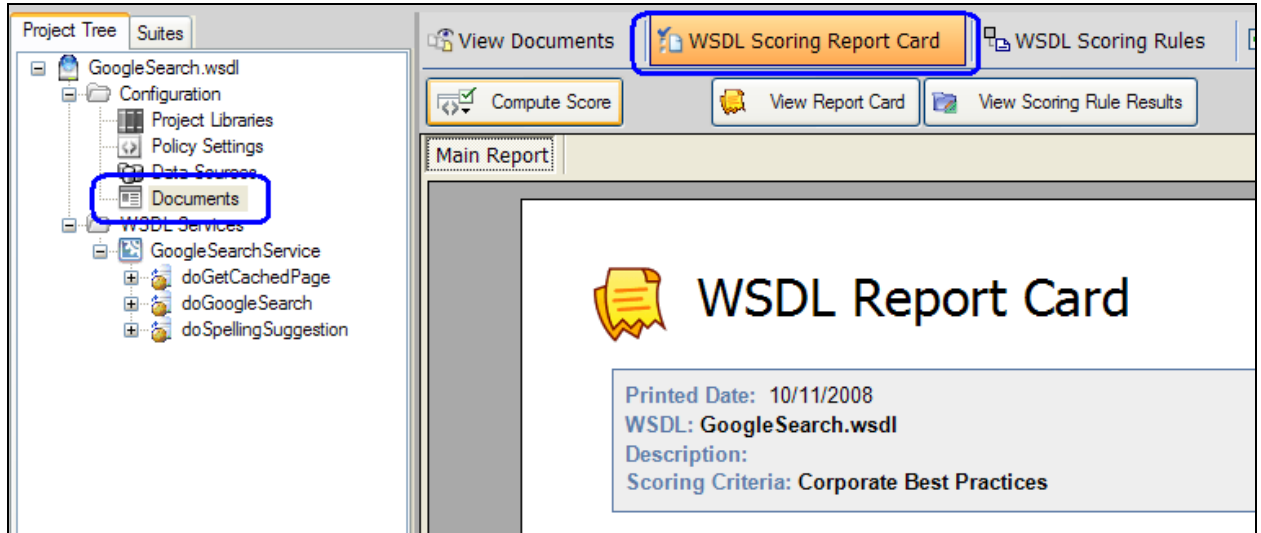
SOAPSonar provides a framework for WSDL analysis and governance to help improve interoperability and best practices adherence in a consistent manner across the enterprise. WSDL Scoring allows WSDL and corresponding Schema to be scored across several categories to provide a scorecard of the rule results. The WSDL scoring rule set can be shared across other SOAPSonar instances to provide a corporate scoring framework to improve the quality of WSDLs and maximize interoperability.

Subtopics in this section include

- [WSDL Scoring](#)
- [WSDL Scoring Rules](#)
- [WS-I Basic Profile Analysis](#)
- [WSDL and Schema Document Review](#)

WSDL Scoring

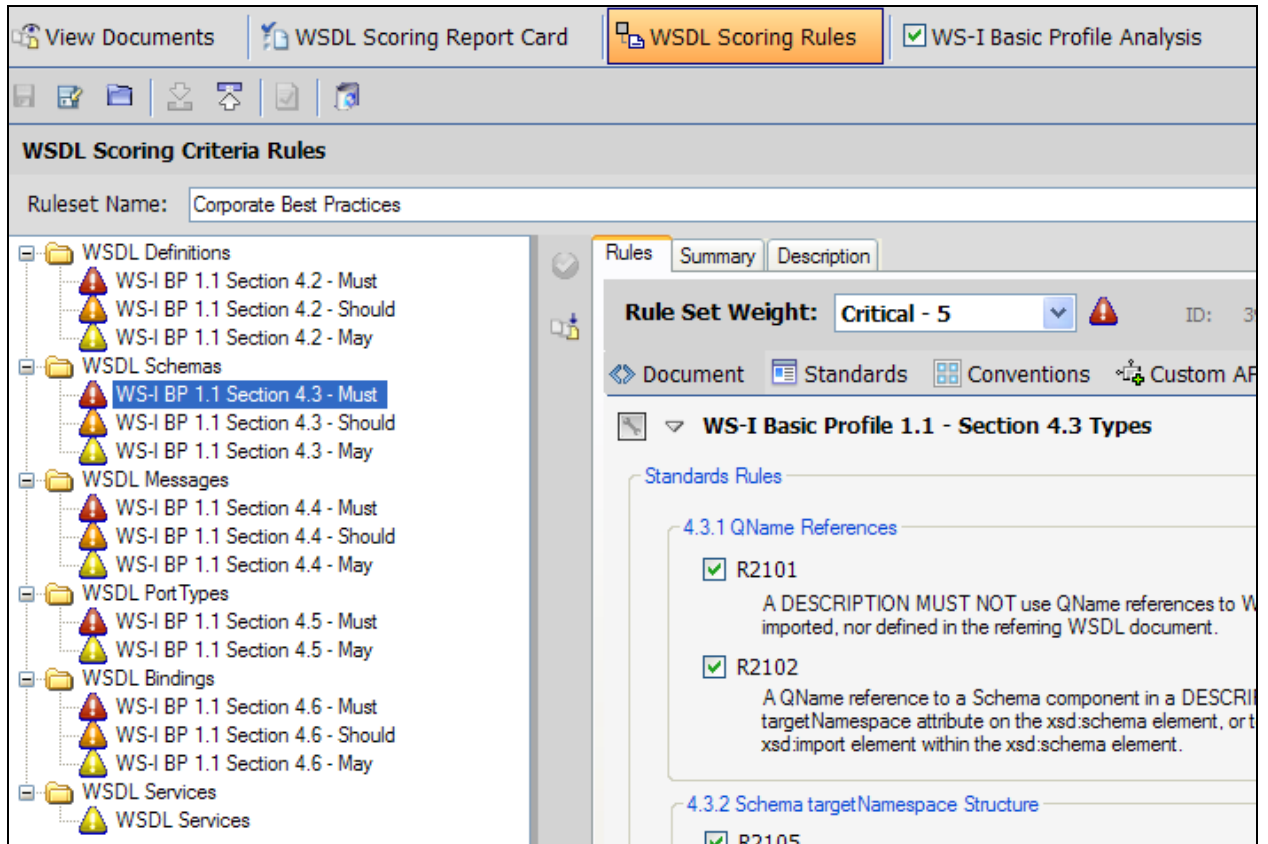
WSDL scoring evaluates the WSDL and Schema across the defined best-practices rules to build a scorecard report for the results. Once a WSDL is loaded into the project, the analysis options for the WSDL including generating a scorecard are found under the Documents node.



To compute the score of the WSDL, click on the “Compute Score” button. Once the scorecard is generated, the report can be viewed in a separate window by clicking on the “View Report Card” and “View Scoring Rule Results” buttons.

WSDL Scoring Rules

The rules for WSDL scoring are extensible and can be defined for the best practices based on industry standards, as well as specific to corporate governance and best practices. Rules are created with severity weights and each rule set has a summary and description which are used to describe the nature of the evaluation rules in the set. These descriptions appear in the scoring rule results report.



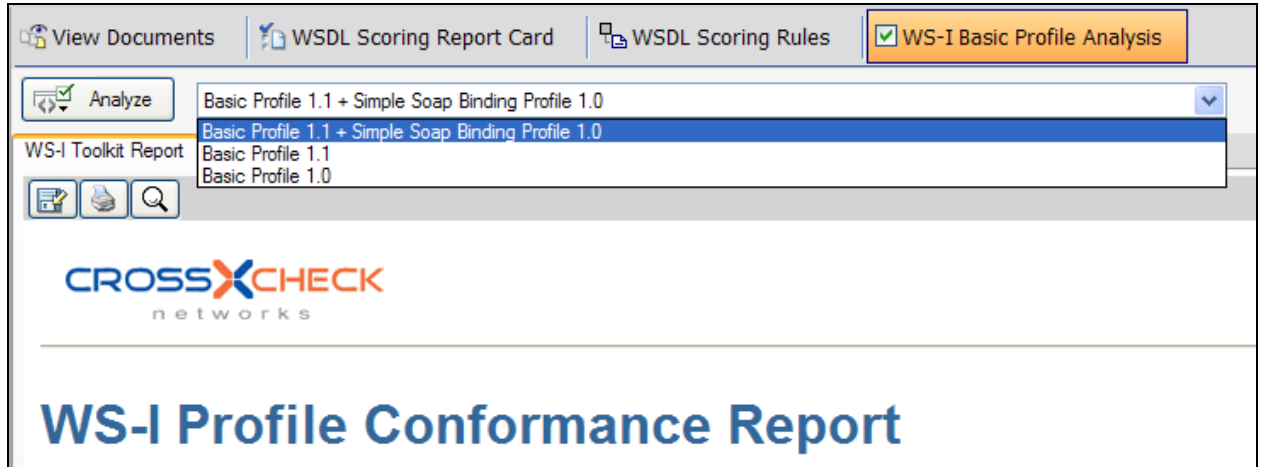
Rule Categories

Scoring rules can be defined for each component of the WSDL which are the WSDL Definitions, Schemas, Messages, PortTypes, Bindings, and Services sections of the WSDL. Within each of these categories, the rule sets include standards based rules such as the WS-I Basic Profile rule evaluation categories as well as extensible rules for naming conventions, and document based rules which provide full extensibility for isolating any part of the WSDL or schema. Rules also include extensible scripting of DLL plugin for even more customized analysis of each of the WSDL categories.

When configuring the rules that are based on WS-I category rules, each rule provides the ability to enable or disable the rules set forth by the WS-I Basic Profile 1.1. The set of enabled features on each rule will be evaluated and each will adhere to the severity weight specified. Severity weights vary from Info=1 through Critical=5.

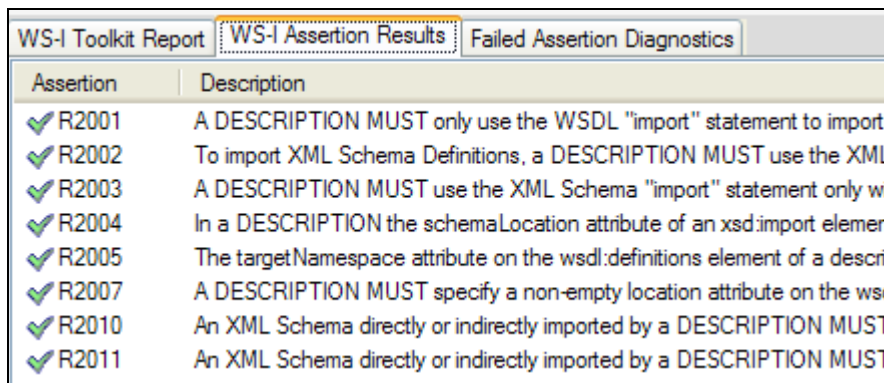
WS-I Basic Profile Analysis

To perform WS-I Basic Profile scanning of the WSDL and Schema, go to the WS-I Basic Profile Analysis option on the menu under the Documents node.



To generate reports, click on the Analyze button after selecting the WS-I Basic profile to use for the analysis.

To look at the results of each assertion, click on the “WS-I Assertion Results” tab



Assertion	Description
✓ R2001	A DESCRIPTION MUST only use the WSDL "import" statement to import a
✓ R2002	To import XML Schema Definitions, a DESCRIPTION MUST use the XML
✓ R2003	A DESCRIPTION MUST use the XML Schema "import" statement only wit
✓ R2004	In a DESCRIPTION the schemaLocation attribute of an xsd:import element
✓ R2005	The targetNamespace attribute on the wsdl:definitions element of a descrip
✓ R2007	A DESCRIPTION MUST specify a non-empty location attribute on the wsd
✓ R2010	An XML Schema directly or indirectly imported by a DESCRIPTION MUST
✓ R2011	An XML Schema directly or indirectly imported by a DESCRIPTION MUST

For failed assertions, if applicable based on the rule, you can click on the ‘Failed Assertion Diagnostics’ tab to review the highlighted sections of the WSDL which triggered the assertion rule failures.

WS-I Toolkit Report | WS-I Assertion Results | Failed Assertion Diagnostics

8 Interactive WS-I Violations Detected

Assertion [BP2406](#)

Description The use attribute has a value of "literal".

Violation The use attribute of a soapbind:body, soapbind:fault, soapbind:header and soapbind:headerfault

Failure Detail Defective soapbind:body, soapbind:fault, soapbind:header, or soapbind:headerfault elements.

<ul style="list-style-type: none"> BP2406 Ref 1 Ref 2 Ref 3 Ref 4 Ref 5 Ref 6 BP2108 	<p>93 <operation name="doGoogleSearch"></p> <p>94 <input message="s0:doGoogleSearch" /></p> <p>95 <output message="s0:doGoogleSearchResponse" /></p> <p>96 </operation></p> <p>97 </portType></p> <p>98 <binding name="GoogleSearchBinding" type="s0:GoogleSea</p> <p>99 <soap:binding transport="http://schemas.xmlsoap.org/</p> <p>100 <operation name="doGetCachedPage"></p> <p>101 <soap:operation soapAction="urn:GoogleSearchAction</p> <p>102 <input></p> <p>103 <soap:body use="encoded" namespace="urn:GoogleSe</p> <p>104 </input></p>
--	--

WSDL and Schema Document Review

Document review of any of the WSDL and Schema documents can be performed from the “View Documents” option menu under the Documents node.

View Documents | WSDL Scoring Report Card | WSDL Scoring Rules

Type	Filename	Namespace
Main WSDL	GoogleSearch.wsdl	urn:GoogleSearch
Embedded Schema		urn:GoogleSearch
Schema Import	soapencoding.xsd	http://schemas.xmlsoap.org/soap/encoding/

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <definitions xmlns:http="http://schemas.xmlsoap.org/ws
3 <types>
4 <s:schema targetNamespace="urn:GoogleSearch">

```

EXTENSIBLE PLUGIN API

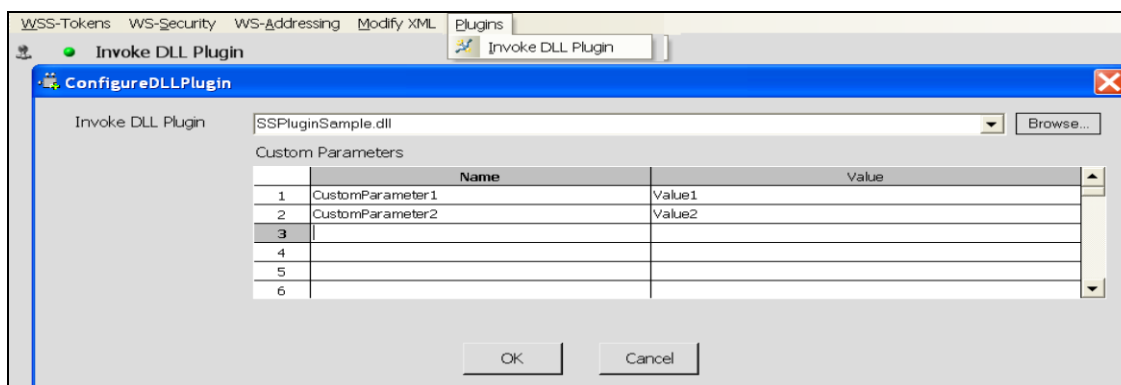
SOAPSonar provides a plug-in interface which allows you to write your own DLL plug-ins to be invoked during the request, response, and success criteria evaluation events. An interface document is provided in the installation subdirectory called **plugins**. The interface DLL is called SOAPSonar.IPlugin.dll and describes each function signature that must be implemented to be called for the events. The functions are documented in the interface document provided in the plugins subdirectory and are in VB.NET and C# format. Once you have created a DLL which implements the interfaces specified in SOAPSonar.IPlugin.dll, you can configure the functions to be invoked for any of 3 events: Request Event, Response Event, and Evaluate Response event.

Request Event

On the Request Tasks tab, the Invoke DLL Plugin menu item allows you to select your DLL to be invoked for each request. The request event triggers the RunCustomRequestTask function. Values returned from your function for the Request and Request Header will be used by SOAPSonar for the current request and header to be sent to the specified endpoint.

```
public bool RunCustomRequestTask(ref string Request, ref string RequestHeader, ref HashTable Variables);
```

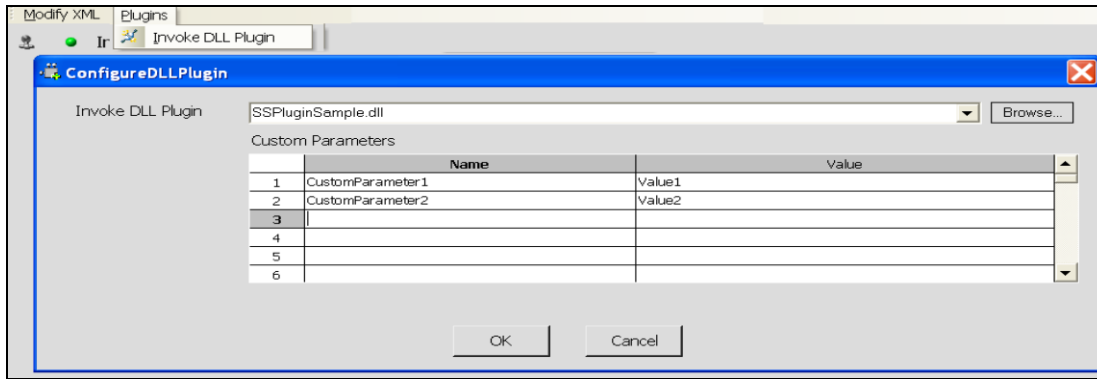
Custom parameters can be passed to your DLL function by specifying them in the table provided.



Response Event

On the Response Tasks tab, the Invoke DLL Plugin menu item allows you to select your DLL to be invoked for each response. The response event triggers the RunCustomResponseTask function. Values returned from your function for the Response and Response Header will be used by SOAPSonar for the success criteria evaluation and Runtime Variable capture.

```
public bool RunCustomResponseTask(ref string Response, ref string ResponseHeader, ref HashTable Variables);
```

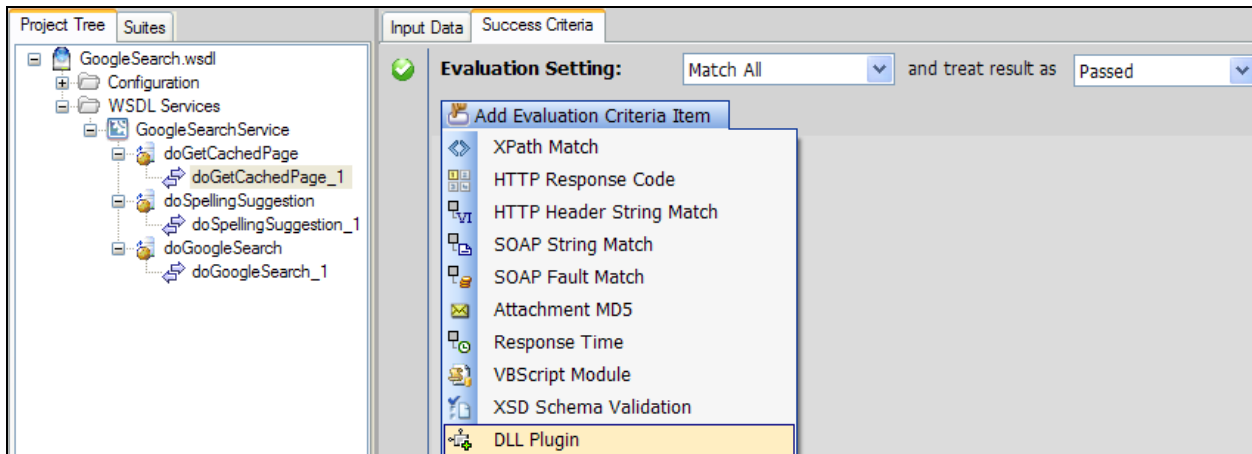


Evaluate Success Event

On the Success Criteria tab, the Invoke DLL Plugin menu item allows you to select your DLL plug-in to be invoked for success/failure assessment of the current response. The response event triggers the EvaluateItem function.

```
public bool EvaluateItem(ref string Request, ref string RequestHeader,
    ref string Response, ref string ResponseHeader,
    ref Hashtable Variables, ref ArrayList Attachments,
    ref string MatchString , ref string MatchContext ,
    ref string MatchExpression , ref string MatchCriteria);
```

Custom parameters can be passed to your DLL function by specifying them in the table provided.



Test Case Success Criteria

Match All and treat result as Passed

⋮ Add

▾ Invoke DLL Plugin

Custom Name Value Pairs to Pass to your EvaluateItem DLL Function

	Name	Value
1		
2		
3		
4		
5		
6		

Enterprise License Component – Automation Edition

SOAPSonar Enterprise Automation Module License Component (APC)

You can extend your standard Enterprise Console license to include extensibility features for testing automation. The Automated Processing Component (APC) is a license feature which enables the automation features listed below. SOAPSonar Automation Edition and SOAPSonar Platinum Edition include the APC component by default.

Automation Data Sources

Create test parameters and success criteria dynamically using externally defined data in CSV, Excel, SQL Server, Oracle, or any ODBC compliant database. This feature allows you to generate test iterations for each row in the data table, and use the data table values as dynamic test input parameters and also dynamic success criteria evaluation values.

Automated Baseline Regression Validation

Enable the ability to create a baseline response set for your Test suites which can be used for subsequent test runs to quickly determine if your tests are no longer adhering to your baseline criteria. This feature also provides the ability to publish an HTML XML Diff report of baseline vs. actual test request and responses.

Batch Processing

Enable the ability to send a set of files which reside in the selected directory for both functional evaluation, or for performance profiling. This allows you to build requests using separate tools, or just store requests from previous test runs to a directory, and then simply resubmit this list of files using the SOAPSonar testing architecture. You can configure authentication settings and dynamic tasks for the batch set.

Command-Line Interface

Enable the use of the command-line sscmd.exe utility where you can run test suites directly from the command line and generate html baseline reports. Leverage the command-line interface to extend your test automation to include test scheduling, custom build script integration, and more.

Extensibility Interfaces

Enables the SOAPSonar Plug-in API. With samples provided in the plugins subdirectory under the installation directory, see how you can easily create your own custom SOAPSonar plug-in DLL to get full access to the following events: Request Event, Response Event, and Success Criteria Event. The plug-in DLLs are invoked using reflection, so you can create the DLL using any language and your function with the matching function signature will be invoked. This is a powerful way to extend SOAPSonar to produce any kind of input, and process any kind of response based on your custom code.

Enterprise License Component Upgrade – Performance Agent Pack

SOAPSonar Enterprise Performance Agent Pack (PAP)

You can extend your Platinum Edition installation with additional virtual client loading power by adding Performance Agent Packs. Each PAP extends the number of available virtual clients by an additional 50.

Appendix A – Now() Context Function Date Formats

Supported date formats for the Now() context function to format the current date and time according to the format definitions within the parameters. By default the Now() and Now(<format>) function calculates time in UTC. If you want to calculate time in the Local Time zone, use Now(local), or Now(local=<format>) where format options are noted below.

The following are examples of user-defined date and time formats for

December 7, 1958, 8:50 PM, 35 seconds:

Function	Displays
\$fn:Now(M/d/YY)\$	12/7/58
\$fn:Now(d-MMM)\$	7-Dec
\$fn:Now(d-MMMM-YY)\$	7-December-58
\$fn:Now(d MMMM)\$	7 December
\$fn:Now(MMMM YY)\$	December 58
\$fn:Now(hh:mm tt)\$	08:50 PM
\$fn:Now(h:mm:ss t)\$	8:50:35 P
\$fn:Now(H:mm)\$	20:50
\$fn:Now(H:mm:ss)\$	20:50:35
\$fn:Now(M/d/YYYY H:mm)\$	12/7/1958 20:50

List of user-defined data and time format options

Character	Description
(:)	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system's LocaleID value.
(/)	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your locale.
(%)	Used to indicate that the following character should be read as a single-letter format without regard to any trailing letters. Also used to indicate that a single-letter format is read as a user-defined format. See below for further details
d	Displays the day as a number without a leading zero (for example, 1). Use %d if this is the only character in your user-defined numeric format.
dd	Displays the day as a number with a leading zero (for example, 01).
ddd	Displays the day as an abbreviation (for example, Sun).
dddd	Displays the day as a full name (for example, Sunday).
M	Displays the month as a number without a leading zero (for example, January is represented as 1). Use %M if this is the only character in your user-defined numeric format.
MM	Displays the month as a number with a leading zero (for example, 01/12/01).
MMM	Displays the month as an abbreviation (for example, Jan).
MMMM	Displays the month as a full month name (for example, January).

gg	Displays the period/era string (for example, A.D.)
h	Displays the hour as a number without leading zeros using the 12-hour clock (for example, 1:15:15 PM). Use %h if this is the only character in your user-defined numeric format.
hh	Displays the hour as a number with leading zeros using the 12-hour clock (for example, 01:15:15 PM).
H	Displays the hour as a number without leading zeros using the 24-hour clock (for example, 1:15:15). Use %H if this is the only character in your user-defined numeric format.
HH	Displays the hour as a number with leading zeros using the 24-hour clock (for example, 01:15:15).
m	Displays the minute as a number without leading zeros (for example, 12:1:15). Use %m if this is the only character in your user-defined numeric format.
mm	Displays the minute as a number with leading zeros (for example, 12:01:15).
s	Displays the second as a number without leading zeros (for example, 12:15:5). Use %s if this is the only character in your user-defined numeric format.
ss	Displays the second as a number with leading zeros (for example, 12:15:05).
F	Displays fractions of seconds. For example ff will display hundredths of seconds, whereas ffff will display ten-thousandths of seconds. You may use up to seven f symbols in your user-defined format. Use %f if this is the only character in your user-defined numeric format.
T	Uses the 12-hour clock and displays an uppercase A for any hour before noon; displays an uppercase P for any hour between noon and 11:59 P.M. Use %t if this is the only character in your user-defined numeric format.
tt	Uses the 12-hour clock and displays an uppercase AM with any hour before noon; displays an uppercase PM with any hour between noon and 11:59 P.M.
y	Displays the year number (0-9) without leading zeros. Use %y if this is the only character in your user-defined numeric format.
yy	Displays the year in two-digit numeric format with a leading zero, if applicable.
yyy	Displays the year in four digit numeric format.
yyyy	Displays the year in four digit numeric format.
z	Displays the timezone offset without a leading zero (for example, -8). Use %z if this is the only character in your user-defined numeric format.
zz	Displays the timezone offset with a leading zero (for example, -08)
zzz	Displays the full timezone offset (for example, -08:00)

Appendix B – SOAPSonar Product Edition Feature Matrix

For the latest matrix comparison, please visit <http://www.crosschecknet.com/doc/SOAPSonar-ProductMatrix.pdf>

	Personal Edition	Standard Edition	Professional Edition	Server Edition
General				
Simple Navigation	X	X	X	X
WSDL Loading	X	X	X	X
OpenAPI Loading	X	X	X	X
HTTP Header Authoring	X	X	X	X
Complex OpenAPI support	X	X	X	X
Complex WSDL support		X	X	X
Enhanced XML Editing Interface		X	X	X
Automatic Data Population		X	X	X
Extensible Test Settings		X	X	X
Project Management				
Project Save, Merge, Import, and Export (allows Team Project Sharing)		X	X	X
WSDL Merge (services, operations, XSD schema)		X	X	X
Clone Test Suites, Test Groups, Test Cases, Test Settings		X	X	X
HP Quality Center Integration				
Run Test Suites from HP Quality Center			X	X
Report Test Results back into HP Quality Center			X	X
Test Flow Management				
Test Case Response/Request Chaining		X	X	X
Dynamic Response Element, Attribute, or Node Capture		X	X	X
Dynamic Response HTTP Header Value Capture		X	X	X
Key Generation				
Windows Integrated PKI		X	X	X
Key Import/Export		X	X	X
1024-4096 bit Keys		X	X	X

RSA, DSA, SHA1, SHA2, MD5 algorithms



Protocol Authentication

HTTP Basic, Kerberos, Digest



SSL X509 Client Auth (with integrated PKI)



Amazon AWSv4



WS-Security and Identity Management

WS-Security Signatures



WS-Security Encryption



WS-Security Username Token



WS-Security X509 Token



WS-Security SAML Token



WS-Security Kerberos Token



SOAP with Attachments (DIME and MIME)



General Dynamic Tasks

WS-Addressing



XSL Transformation



Arbitrary String Replacement



Arbitrary XML Insertion



Compliance

Design Time WSI-BP 1.1 Compliance



WSDL Region WS-I Assertion Violation Highlighting



Dynamic XSD-Mutation™ Active WSI-BP 1.1 Compliance



Vulnerability Assessment

Dynamic XSD-Mutation™ Automated Vulnerability Suites



Dynamic XSD-Mutation™ Web Service API Boundary Testing



Author Custom Auto Vulnerability Definitions (AVD)



Risk Mitigation Report Scoring



Extensible Criteria Libraries



Reporting

Detailed Statistics Logging	X	X	X	X
Graphs	X	X	X	X
Export Reports in PDF, XML, CSV formats		X	X	X
Export Grid Data to Excel, HTML		X	X	X
Automation Data Sources				
CSV File			X	X
Excel			X	X
MySQL			X	X
SQL Server 2005			X	X
Oracle			X	X
Generic ODBC Connection Support			X	X
Dynamic Success Criteria Evaluation				
Automation Data Source Column Variables			X	X
Runtime Variables			X	X
Baseline Regression Testing				
Generate Test Suite baseline response set			X	X
Configure XML Diff Criteria from Baseline			X	X
Publish HTML XML Diff Baseline vs. Actual Test Report			X	X
Extend and Manage Regression Baseline versions			X	X
Extensibility Plug-Ins				
Command-Line Interface			X	X
Request SOAP DLL Plug-in Interface			X	X
Response SOAP DLL Plug-in Interface			X	X
Success Criteria DLL Plug-in Interface			X	X
Performance Loading				
50 Virtual Users (concurrent loading engines)				X
SLA Rate Throughput Throttling				X
Distributed Loading (agents)				X
Performance Diagnostics				
Error Tracing Load Diagnostics				X

Index

- A.E.T. SafeSign, 196
- Abstract Types, 42
- Add Test Delay Action, 70
- Array Allocation, 41
- Array Capture using Runtime Variables, 96
- Automation Features, 224
- AVD Library, 175
- Baseline Regression, 153
- Batch File Test Case, 52
- Building Tests, 38
- Command-Line, 193
- Compliance Mode, 177
- Compliance Suite Properties, 125
- Conditional Testing, 120
- Create a Test Suite, 118
- Data Source, 102
- Database Query Action, 69
- Disable Automatic Test Dependency Detection, 101
- Distributed Performance Agent Loading, 169
- Dynamic Arrays from Data Source, 112
- Dynamic URI, 98
- EMS, 189
- Entry Recall, 44
- File Manipulation Action, 70
- Getting Started, 18
- HP QC Linked Data Sources, 104
- HTTP Decompression, 55
- Iteration Mode Data Sources, 102
- Java Keystores, 109
- JMS, 191
- JScript Inline Functions, 67
- JSON Test Case, 52
- JUnit Integration, 213
- Key Alias Management, 110
- Large File Streaming Response, 73
- Logging, 181
- Map Fields to Data Source, 112
- Message Providers, 184
- MQ, 184
- MTOM and MIME Large File Streaming, 60
- Performance Mode, 160
- Performance Mode Transaction Tracing, 162
- Performance Suite Properties, 123
- PFX Files, 110
- Popup Window Editor, 106
- Project Import, 128
- Project Management, 127, 216
- Project Settings, 31
- Purge Message Queue Action, 71
- QA Mode, 132
- QA Suite Properties, 122
- Quality Center Integration, 198, 206
- Reporting, 181
- Request Plugin, 66
- Request URI Variables, 98
- Resolved Variable Test Iteration Viewer, 107
- Response Data View Formats, 72
- REST Test Case, 54
- Run Modes, 131
- Run View, 117
- Runtime Dependency Variables, 93
- Runtime Variable - Include Encoded XML Setting, 95
- Runtime Variable – Mirror Value in Global Variable setting, 95
- Runtime Variable – Store runtime values in List Variable, 94
- Runtime Variable – Update Memory Table, 94
- Schema Fields View Filtering, 44
- Schema Validator, 106
- Screen Layout Docking, 71
- Send Email Action, 71
- SmartCard Keys, 196
- SmartCard Security Tasks, 196
- SmartCard Setup, 196
- SmartCard SSL X509 Authentication, 196
- SOAP Generator, 40, 47
- SOAP Header Authentication, 62, 64
- SOAP Header Authentication – Dynamic Tokens, 63
- SOAP Test Case, 39, 75, 114, 115
- SOAPSonar Editions, 11
- SOAPSonar GUI, 19
- Sorting WSDL Operations Alphabetically, 33
- Success Criteria – Attachment MD5, 145
- Success Criteria - Database Query, 150
- Success Criteria - DLL Plugin, 148
- Success Criteria - Dynamic Variables, 151
- Success Criteria - File Analysis, 150
- Success Criteria - HTTP Code, 144
- Success Criteria - HTTP Header Match, 144
- Success Criteria - Response Time, 146
- Success Criteria - Schematron Validation, 147
- Success Criteria - SOAP Fault Match, 145
- Success Criteria - SOAP Match, 145
- Success Criteria - VBScript, 146
- Success Criteria – XPath Array Handling Options, 139
- Success Criteria - XPath Compare Node Fragment Array to Database Query, 141
- Success Criteria – XPath Match, 134
- Success Criteria - XSD Schema Validation, 147
- Success Criteria Rules, 133
- Test Case - Attachments, 59
- Test Case - HTTP Headers, 56
- Test Case - Protocol Authentication, 57
- Test Case - Request Tasks, 61
- Test Case - STUB Tests, 56
- Test Case Chaining, 93, 99

Test Case Request, 55
Test Case Response, 72
Test Case Variables, 76
Test Configuration Tools, 105
Test Flow Mapping Data Sources, 103
Test Suite Post-Run Tasks, 120
Test Suite Pre-Run Tasks, 119
UDDI, 110
Update Global Variable Action, 70
Update Memory Table Task, 70
Upgrade a WSDL, 129
Variables - Automatic Data Function, 85
Variables - Automation Data Source, 88
Variables - Context Function, 77
Variables - Global, 80
Variables – Memory Table, 82
Variables – Project Global, 81
VBScript Inline Functions, 67
Vulnerability Mode, 171
Vulnerability Suite Properties, 126
Windows Keystore, 109
WLS, 187
WSDL Capture, 29, 30
WSDL Parsing Settings, 33
WSDL Project SOAP Version, 129
WS-Security, 63
WS-Trust Test Case, 54
XML Test Case, 47
XPath Success Criteria Rule Evaluation Modes,
136
XSD Mutation, 173
XSLT Transformation, 64, 65, 66